The Complexity of Explicit Construction Problems

Hanlin Ren

Christ Church College

University of Oxford

A thesis submitted for the degree of

Doctor of Philosophy

Trinity 2025

Abstract

Explicit constructions of pseudorandom objects play a central role in theoretical computer science. Unfortunately, for many properties of interest, while a randomly chosen object would satisfy them with high probability, deterministically constructing such objects remains notoriously challenging. For example, Erdős famously showed that a random graph is almost always a *Ramsey graph*, yet explicitly constructing Ramsey graphs has been a long-standing open problem.

Recently, several explicit construction problems have seen progress through *complexity-theoretic ideas*. Unlike previous ad-hoc methods tailored to individual cases, these complexity-theoretic approaches offer **systematic solutions** applicable to **broad families** of explicit construction problems. Central to this line of research is the *Range Avoidance* problem (Avoid), which encapsulates a wide array of explicit construction problems: Deterministically solving Avoid would simultaneously resolve many open questions in explicit construction. What are the strengths and limitations of these complexity-theoretic techniques, and how can understanding the complexity of Avoid illuminate these questions?

This thesis examines the complexity of explicit construction problems, with a particular focus on the Range Avoidance problem. We uncover a strong connection between explicit construction problems and fundamental questions in complexity theory, making progress in both domains. Specifically, our contributions include:

- **Algorithmic Method for Range Avoidance:** We generalise Williams's Algorithmic Method (2011)—originally developed for proving circuit lower bounds—to solve the Range Avoidance problem. Our results demonstrate that this method applies not only to circuit lower bounds but also to explicit construction problems in general. Consequently, we derive new complexity lower bounds and develop novel algorithms for special cases of Avoid. Building on these techniques, we further show that a slight extension of the Algorithmic Method fully characterises, i.e., is *both necessary and sufficient* for, proving circuit lower bounds for E^{NP}.

- **Unconditional constructions:** We present new *unconditional* results in explicit constructions. In particular, we devise an infinitely-often pseudodeterministic polynomial-time algorithm for finding prime numbers. We also introduce a new algorithm for solving the Range Avoidance problem. The latter result yields near-optimal circuit lower bounds for the complexity classes $\Sigma_2\mathsf{E}$ and $\mathsf{S}_2\mathsf{E}/_1$, resolving a 40-year-old open question from Kannan (1982). These results are obtained through a novel "iterative win-win" method, which is likely to have broader applications in complexity theory.

- **Additional results:** In addition, we study the related *Heavy Range Avoidance* problem, uncovering its connections to uniform lower bounds and derandomisation. We also present new hardness results for the Range Avoidance problem under Rudich's demi-bits conjectures, which have implications in proof complexity as well.

Acknowledgements

First and foremost, I would like to thank my advisor Rahul Santhanam for basically everything: his guidance, his deep insights in the field, his support and encouragement, our long walks at Magdalen's Deer Park where we exchanged a lot of ideas and observations, the countless many meetings where Rahul develops and adjusts my intuitions and guides me to more fruitful directions, the many pieces of advice Rahul gives to me about how to become a better researcher, and the moments when I feel down and hopeless and then Rahul cheer me up. With Rahul I am never "stuck" at research: whenever I feel a lack of progress, just go to Rahul's office and talk about my failures, then Rahul will turn these failures into new ideas and directions to explore! When I look back it is still hard to believe how much I have grown since I met Rahul, both as a complexity theorist and as a human being.

I am also grateful to Lijie Chen—another person without whose support I would hardly have achieved anything in complexity theory. The papers we wrote together, [CR22][1] and [CLO+23, CHR24], are not only my proudest works but also turning points in my life. The project that resulted in [CR22] was a light in my darkest times and became my main *raison d'être* during that time. We would meet again three years later at Simons where we tried really hard to design pseudodeterministic algorithms for Range Avoidance—while calling those algorithms "circuit lower bounds." In addition, I really admire Lijie for being a great mentor, as can be seen from the number and later achievements of the students he has worked with. I previously thought that the great names in the older generation are inspirations for the younger generation; but Lijie inspires people *barely* younger than himself.

I also want to thank Igor C. Oliveira for our collaborations and time spent together. Igor's hospitality makes my every visit to Warwick a delightful and enjoyable journey. It is hard to overstate how much I benefit from the intense exchange of ideas in Igor's office, where the desk fan spins quietly and whiteboard wipes accumulate in the bin. In fact, Igor's office is one of the places where I get struck by new ideas and make progress the most number of times.

I am grateful to my undergraduate mentor at Tsinghua University, Ran Duan, for bringing me into the beautiful world of theoretical computer science and graph algorithms. From a complete newbie in TCS research to a doctoral-student-to-be deciding between algorithms and complexity, my undergraduate life became colourful because of Ran's guidance and patience, along with the haunting open problems and beautiful insights he shared with me.

I also want to thank Shuichi Hirahara for all the discussions and our friendship during the years and for hosting me as a research intern at National Institute of Informatics. Besides Shuichi, I also want to thank Mikito Nanashima and Nobutaka Shimizu for all the ideas shared

[1] The conference version of [CR22] was published in STOC'20.

and all the time spent with me. You were my best friends in Tokyo. I am still looking forward to our next lunch at "the Japanese restaurant" (Shuichi, you know which one I mean :-)).

I also want to thank Rahul Ilango and Zhenjian Lu for our long-term collaboration spanning multiple projects. Rahul is a unique type of person with a unique approach to difficult problems and a unique sense of humour. I am also grateful to Rahul for taking photos of mine, both good and bad.[2] Zhenjian is both a cool person and a warm person, and has taught me many things related or unrelated to research. I feel so lucky to have the chance of working with people like them.

I had the precious opportunity of working with the (undergraduate) visitors at Oxford and Warwick: Yeyuan Chen, Yizhi Huang, Jiatu Li, Jingyi Lyu, and Zhikun Wang. I want to thank especially Zhikun for frequently inviting me to the St. Annes library and for tolerating my long-winded complaint about how terrible human beings are (especially at proving lower bounds). I also want to thank Jiatu for tirelessly attempting to teach me bounded arithmetic. It is a privilege to work and live with brilliant young minds like them.

I also want to thank all my collaborators besides those mentioned above: Michal Garlík, Svyatoslav Gryaznov, Yong Gu, Jiawei Li, Yuhao Li, Iddo Tzameret, Yichuan Wang, and Yan Zhong. Many of my projects would be impossible without your ideas and efforts. A special thanks to the Computational Complexity Conference (CCC) where many of my collaborations begin inadvertently—my project with Jiawei and Yuhao [LLR24] began at CCC'23 and my project with Yichuan and Yan [RWZ26] began at CCC'24.

I also want to thank everyone who had supported or helped me at various time in my DPhil journey—a partial list includes Robert Andrews, Levente Bodnar, Bruno Cavalar, Yijia Chen, Yilei Chen, Kuan Cheng, Mahdi Cheraghchi, Matthew Gray, Siyao Guo, Stefan Grosser, Alexandros Hollender, Valentine Kabanets, Antonina Kolokolova, Nutan Limaye, Ian Mertz, Jan Pich, Ninad Rajgopal, Srikanth Srinivasan, Roei Tell, Ryan Williams, and I'm sure that this list is far from being complete. In addition, I'm grateful to Stanislav Živný and Valentine Kabanets for reading my thesis and providing valuable comments and feedbacks.

I am also grateful to many friends in Oxford for their companionship during different phases of my DPhil life and for our shared memories. I want to thank Zepeng Cao, Yishun Lu, Xiao Su, Zhengyi Xiao, Yishan Xu, and Yuwen Yan for being my closest friends in my first year. I want to thank Paul Chang, Shanshan Hua, and Yilong Yang for organising regular badminton games. I want to thank Luzhi Chen for sharing and talking about VOCALOID music with me when I'm stressful, and for making Curry Udon together. I'm also grateful for a musician (who wishes to stay anonymous) and their four songs, which accompanied me through the most difficult times.

Finally, I want to thank my Dad and Mom for their unconditional support, even though they might not understand what I've been doing.

[2]The good one is available at https://tinyurl.com/h4n1inphoto.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

"How difficult could it be to find hay in a haystack?"

Howard Karloff [AB09, Chapter 21]

1.1 Explicit Construction Problems

The Prime Number Theorem implies that a significant fraction of integers are prime: If one samples an n-bit integer uniformly at random (i.e., from the interval $[2^{n-1}, 2^n)$), then the probability that it is prime is at least $\Omega(1/n)$. Therefore, to find a large prime number, it suffices to generate a few random integers and test each of them for primality. Perhaps surprisingly, it is unknown if the same task can be accomplished efficiently without using randomness: current *deterministic* methods for generating n-bit primes either rely on strong number-theoretic or complexity-theoretic assumptions (such as Cramér's conjecture [Cra36] or $\mathsf{P} = \mathsf{BPP}$ [IW97]), or require exponential ($2^{\Omega(n)}$) time [LO87, TCH12, BHP01].[1]

It turns out that the above example of finding prime numbers is just one of numerous difficult "explicit construction" problems, where a random object is very likely to satisfy a certain desired property, but it appears very difficult to find any object satisfying the same property *deterministically*:

- In 1959, Erdős [Erd59] famously introduced the *probabilistic method* to show that a random graph is likely to be a *Ramsey graph*. Since then, it has been a notorious open problem to find an *explicit construction* of Ramsey graphs: the best known explicit constructions achieve parameters that are *significantly weaker* than those obtained by random graphs [FW81, Alo98, Gro00, Bar06].

  Ramsey graphs are closely related to *two-source dispersers* and *two-source extractors* in the study of pseudorandomness. Although there has been substantial progress in explicitly constructing such objects [CG88, Raz05, Bou05, BKS+10, BRSW12, Li12, CZ19, Li16, BDT16, CL16, Coh16a, Coh16b, Coh17, Mek17, Li17, Li19, Lew19, Li23], the challenge of

[1] In fact, the problem of fast deterministic generation of primes was exactly the focus of the Polymath 4 project. This project establishes an improved algorithm for counting the *parity* of the number of primes in an interval [TCH12], but despite much effort, it did not improve the $2^{n/2+o(1)}$ time bound established in [LO87] for deterministically *generating* an n-bit prime. See https://michaelnielsen.org/polymath/index.php?title=Finding_primes (Accessed: Aug 28, 2025) for more details.

constructing an explicit two-source extractor with parameters matching those of random functions remains wide open.

- Valiant [Val77] showed, again via the probabilistic method, that a random matrix is very likely to be a *rigid matrix*. Explicit constructions of rigid matrices would have major consequences in complexity theory, with applications to circuit complexity [Val77, Pud94, AW17], communication complexity [Raz89, Wun12, Lok01, Pud94], and beyond [Lok09]. Yet, despite decades of work, the best known explicit constructions fall far short of the rigidity parameters achieved by random matrices, and thus have not been strong enough to yield the desired implications [Fri93, SSS97, GT18, VK21, Ram20].

- Random linear codes achieve a rate-distance trade-off known as the *Gilbert–Varshamov bound* [Gil52, Var64] with high probability. Yet, despite extensive work, constructing such codes explicitly has remained a longstanding open problem in coding theory [Jus72, AGHP92, ABN+92, NN93, BT11, BT13, Ta-17].

- Perhaps the most dramatic example is that of *circuit lower bounds*. More than 75 years ago, Shannon [Sha49] showed that most Boolean functions on n inputs require circuits of size $\Omega(2^n/n)$. Yet the strongest known lower bounds for any explicit function remain extremely weak: only $3n$ over the complete basis [KM65, Sch74, Sto77, Pau77, Blu84, DK11] and $5n$ over the De Morgan basis [Sch76, Zwi91, LR01, IM02]. Two recent breakthrough works used intricate case analysis to push the bound slightly further, achieving $3.1n - o(n)$ over the complete basis [FGHK16, LY22]. Even more embarrassingly, we are still unable to rule out the possibility that E^{NP}—the class of problems solvable in $2^{O(n)}$ time with an NP oracle—admits linear-size circuits. Arora and Barak's textbook [AB09] described circuit lower bounds as "complexity theory's Waterloo."

1.2 Complexity-Theoretic Constructions

Most previous explicit construction algorithms rely on ad-hoc insights tailored to the particular construction problems at hand. For example, the seminal zig-zag construction of expander graphs [RVW02] involves studying the spectral properties of the "zig-zag product" of two graphs in terms of the spectral properties of the original graphs. As a result, we are often stuck at many seemingly simple explicit construction problems (such as constructing rigid matrices) since they do not seem amenable to "ad-hoc analysis."

In contrast, there are a few notable (counter-)examples where construction algorithms are driven by general ideas from complexity theory rather than problem-specific tricks:

- **Pseudodeterministic constructions for primes.** Oliveira and Santhanam [OS17b] showed unconditionally that there is a subexponential-time *pseudodeterministic* algorithm that constructs primes infinitely often. A pseudodeterministic algorithm [GG11] is a randomised algorithm that, with high probability, outputs a fixed *canonical answer*, independent of the internal randomness. It is easy to see that the naïve randomised algorithm for constructing primes using rejection sampling is *not* pseudodeterministic: Under different choices of internal randomness, the algorithm is likely to output different primes.

Notably, the algorithm of Oliveira and Santhanam uses very few properties of primality: only that there are a lot of primes (the Prime Number Theorem) and that primality can be decided in deterministic polynomial time [AKS04]. Instead, the algorithm relies on complexity-theoretic ideas such as derandomisation [IW01, TV07] and win-win analysis. As a consequence, the algorithm solves the explicit construction problem (pseudodeterministically) not only for primes, but for any *dense property in* P.

- **Rigid matrices with an NP oracle.** Alman and Chen [AC19] showed how to construct rigid matrices with parameters much better than previously known, with the aid of *an* NP *oracle*. That is, they designed an $\mathsf{FP}^{\mathsf{NP}}$ algorithm that, given 1^n (the length-n unary string) as input, outputs an $n \times n$ matrix that is rigid (in some parameter regime). Previously, it was unclear how to construct such matrices even with an NP oracle.

  Perhaps what is more interesting is their technical insight: They treated low-rank matrices as a (non-standard) *circuit class* $\mathscr{C}$ and observed that constructing a rigid matrix is equivalent to proving an (average-case) *circuit lower bound* against $\mathscr{C}$. Then, they invoked Williams's seminal "Algorithmic Method" [Wil13a, Wil14] for proving circuit lower bounds. The Algorithmic Method needs a "circuit-analysis" algorithm for $\mathscr{C}$, which follows from previous work [CW21].[2] The only property of matrix rigidity used by [AC19] is the existence of such a "circuit-analysis" algorithm for $\mathscr{C}$.

Compared to "ad-hoc" algorithms, algorithms based on complexity-theoretic ideas are able to solve an *entire class* of explicit construction problems (e.g., all dense properties in P) and make progress on many difficult problems that otherwise resist attack. However, one drawback is that these algorithms only achieve weaker notions of explicitness: compared to standard, deterministic polynomial-time constructions, [OS17b] only achieves *pseudodeterministic* constructions, and [AC19] requires an NP oracle. When can we leverage complexity theory to solve explicit construction problems? Which notions of "explicitness" can we hope to achieve through such methods? Is there a deeper "complexity theory" underlying these explicit construction problems?

1.3 The Range Avoidance Problem

It turns out that the following total search problem, known as the *Range Avoidance problem* (Avoid), plays a central role in the study of explicit construction problems:

Problem 1.3.1 (Range Avoidance Problem)**.** Given the description of a circuit $C : \{0,1\}^n \to \{0,1\}^\ell$, where $\ell > n$, output any string $y \in \{0,1\}^\ell$ that is not in the range of C. That is, for every $x \in \{0,1\}^n$, $C(x) \neq y$.

The *dual weak pigeonhole principle* [Kra01b, Jeř04] states that if N pigeons are placed into M holes where $M \geq 2N$, then there is an empty hole. This principle implies that Avoid is a *total* problem, i.e., it always has a valid solution. As a natural example of total search problems in functional $\Sigma_2\mathsf{P}$ ($\mathsf{TF}\Sigma_2\mathsf{P}$), this problem was also studied in [KKMP21] under the name 1-Empty.[3] Indeed, it is easy to see that Avoid belongs to (the function version of) $\Sigma_2\mathsf{P}$, but it

[2] The conference version of [CW21] appeared in SODA'2016.

[3] "Empty" stands for "empty pigeonhole principle"; the constant 1 means that the input circuit has stretch at least one bit, i.e., $\ell \geq n + 1$.

is unknown whether it is in FNP. We may try to solve Avoid by guessing a string $y \in \{0,1\}^\ell$ as an answer, but it seems unclear how to verify that y is not in the range of C without using a universal quantifier.

It was pointed out by Korten [Kor21] that the Range Avoidance problem nicely captures the complexity of explicit constructions. Most explicit construction problems can be rephrased as a *unary total search* problem: Given 1^n as input, find a valid object of size n. It was shown in [Kor21, Section 3] that many explicit construction problems reduce to Avoid, including Ramsey graphs, rigid matrices, circuit lower bounds, and many more.[4] In fact, for combinatorial objects whose existence is proven via the probabilistic method, it tends to be the case that constructing such objects reduces to the Range Avoidance problem.

Example 1.3.2. Consider, for example, the problem of proving circuit lower bounds. Fix a size threshold such as $s(n) := 2^{n/2}$, we want to solve the following (unary) total search problem: Given 1^N as input where $N := 2^n$, find the length-N truth table of any function $f : \{0,1\}^n \to \{0,1\}$ that cannot be computed by circuits of size $s(n)$.

Let $\mathsf{TT} : \{0,1\}^{O(s \log s)} \to \{0,1\}^{2^n}$ denote the function that takes as input the description of a size-s circuit, and outputs the truth table of this circuit. (Here TT stands for *"truth table"*.) If we could solve Avoid on the particular instance TT, then we could find a truth table $tt \in \{0,1\}^{2^n}$ without size-s circuits, therefore proving a circuit lower bound. More precisely, solving Avoid for TT in polynomial time is equivalent to proving a circuit lower bound for E, and solving Avoid for TT in $\mathsf{FP}^{\mathsf{NP}}$ is equivalent to proving a circuit lower bound for E^{NP}.

Perhaps surprisingly, the main result of [Kor21] is that proving circuit lower bounds is *the hardest explicit construction*: There is a P^{NP}-reduction from Avoid to the total search problem defined in Example 1.3.2 for any $s(n) = 2^{\Omega(n)}$. (As pointed out in [Kor21], the same result was proven in the language of bounded arithmetic by Jeřábek [Jeř04].) In this regard, the study of explicit construction sheds light on the (metamathematical) difficulty of circuit complexity: Proving circuit lower bounds is hard because it is *as hard as* solving a wide class of explicit construction problems, many of which (e.g., Ramsey graphs, rigid matrices, and so on) seem unrelated at first glance.

1.4 Our Contributions: A Bird's-Eye View

In this thesis, we investigate the complexity of explicit construction problems, with a particular emphasis on the Range Avoidance problem. Our study reveals an *intimate and bidirectional* connection between explicit construction and complexity theory: Insights in complexity theory often lead to advances in explicit constructions, and, in turn, these advances yield consequences back to complexity theory as well!

An Algorithmic Method for explicit construction. First, motivated by the success of the Algorithmic Method in constructing rigid matrices [AC19, BHPT24, CLW20, CL21, HV21], we study the effectiveness of the Algorithmic Method for general explicit construction problems.

[4] Primality appears to be an exception. It remains unknown whether there is a polynomial-time reduction from finding prime numbers to Avoid [PWW88, Kor22].

In Chapter 3, we develop a version of the Algorithmic Method for solving the Range Avoidance problem. While the traditional Algorithmic Method derives circuit lower bounds from "non-trivial" circuit-analysis algorithms, our Algorithmic Method solves the Range Avoidance problem with an NP oracle given a (different kind of) "non-trivial" circuit-analysis algorithm. Using circuit-analysis algorithms that are implicit in the literature [Wil18c], we obtain unconditional $\mathsf{FP}^{\mathsf{NP}}$ algorithms for a special case of Range Avoidance for ACC^0 circuits. This both recovers the best known ACC^0 circuit lower bounds [CLW20] and implies new lower bounds.

Our results suggest a reinterpretation of the intuition behind [AC19], which treats low-rank matrices as a "circuit class" and applies the Algorithmic Method to prove circuit lower bounds. We propose that the Algorithmic Method is fundamentally a technique for explicit construction problems—proving circuit lower bounds is just one of its applications.

We take a step back and consider the Algorithmic Method for proving circuit lower bounds in Chapter 4. We first prove a slight extension of the Algorithmic Method: circuit-analysis algorithms, even with "E^{NP} preprocessing", imply circuit lower bounds. Next, we show that this is in fact the *complete* Algorithmic Method that *characterises* circuit lower bounds for E^{NP}: for circuit classes $\mathscr{C}$ satisfying mild technical conditions, E^{NP} is hard against $\mathscr{C}$ if and only if such circuit-analysis algorithms with E^{NP}-preprocessing exist. Curiously, while this result itself is entirely within the realm of circuit complexity, the most natural way of deriving it seems to be via explicit constructions and the Range Avoidance problem. We view this characterisation of circuit lower bounds for E^{NP} as a "gift" from explicit constructions to circuit complexity.

The iterative win-win method. Next, we present new unconditional results in explicit construction and circuit lower bounds:

- In Chapter 6, we present a *polynomial-time*, infinitely-often, pseudodeterministic construction of primes, improving upon the prior subexponential-time algorithm of [OS17b].
- In Chapter 7, we show that the complexity classes $\Sigma_2\mathsf{E}$ and $\mathsf{S}_2\mathsf{E}/_1$ cannot be computed by circuits of size $2^n/n$. Previously, only super-polynomial size lower bounds for these classes were known [Kan82, MVW99, CCHO05, Cai07].

Both results rely on a novel *iterative win-win method*. Previous results only achieved sub-optimal bounds: [OS17b] obtained subexponential-time constructions (ideally we would like polynomial-time constructions) and [Kan82] proved super-polynomial circuit lower bounds (ideally we would like exponential lower bounds). A common reason for the inefficiency of these results is that their proofs use a win-win analysis that involves two cases; the structure of the win-win analysis prevents us from obtaining optimal bounds in both cases. In fact, this proof strategy yields so-called "half-exponential" bounds [MVW99].

Instead, in Chapter 6 and Chapter 7, we use a more refined win-win analysis involving *not two, but* $O(\log n)$ *many cases*. A careful analysis of this iterative win-win approach gives optimal bounds on each of the $O(\log n)$ cases.

We also remark that by the main result of [Kor21], the new circuit lower bound in Chapter 7 implies pseudodeterministic constructions with an NP oracle for a wide range of objects, including Ramsey graphs, rigid matrices, optimal linear codes, and more. These results further

illustrate the strong synergy between circuit complexity and explicit constructions: advances in one area would catalyse progress in the other.

Avoiding heavy elements. In Chapter 8, we study a variant of the Range Avoidance problem which we call the *Heavy Avoidance* problem (`Heavy-Avoid`). In this problem, given a distribution $\mathcal{D}$ over $\{0,1\}^n$ (described by a circuit sampling from $\mathcal{D}$) and a parameter $\delta \geq 1/\mathrm{poly}(n) \gg 2^{-n}$, the goal is to output any string $x \in \{0,1\}^n$ that is sampled from $\mathcal{D}$ with probability at most δ. Clearly, this problem is total and can be solved in $\mathsf{SearchBPP}$. Does it admit deterministic algorithms? Or is it complete for prBPP?

We investigate both questions above and show that both are connected to long-standing open problems in complexity theory. Just like Avoid is intimately connected to circuit lower bounds, we show that `Heavy-Avoid` is also intimately connected to lower bounds against *uniform probabilistic circuits*; in fact, deterministic algorithms for certain versions of `Heavy-Avoid` are equivalent to such lower bounds. Then, we study the connection between `Heavy-Avoid` and derandomisation. Leveraging recent advances in derandomisation [CT21a, LP23], we show that `Heavy-Avoid` is prRP-hard under very weak, non-black-box reductions. We also study whether `Heavy-Avoid` is prBPP-complete under more standard types of reductions, but in general this remains an intriguing open question.

Hardness of Range Avoidance. Finally, in Chapter 9, we study the hardness of Avoid with respect to deterministic (and nondeterministic) algorithms. Ilango, Li, and Williams [ILW23] showed that, assuming the existence of subexponentially-secure indistinguishability obfuscations ($i\mathcal{O}$) [BGI+12, GGH+16, JLS21] and that $\mathsf{NP} \neq \mathsf{coNP}$, there is no deterministic algorithm that solves Avoid in polynomial time. This result may seem surprising, since there is a trivial randomised algorithm for Avoid: simply output a random string and it is not in the range of the input circuit with good probability. Building on this, Chen and Li [CL24] showed that under certain subexponential assumptions in nondeterministic cryptography with a public-key flavour (i.e., "Cryptomania" [Imp95]), there is no *nondeterministic* algorithm that solves Avoid in polynomial time.

The main result in Chapter 9 strengthens these findings: we show that the existence of *demi-bits generators* [Rud97] implies the non-existence of *nondeterministic* polynomial-time algorithms solving the Range Avoidance problem. This improves upon the previous results [ILW23, CL24] in two aspects: First, the assumption we use is of "Minicrypt" flavour and is arguably weaker than those in [CL24]. Second, our results do not require any subexponential hardness assumption.

As observed in [RSW22], the hardness of Avoid against nondeterministic algorithms is connected to the theory of *proof complexity generators* [Kra25], hence our results have implications in proof complexity as well. In Chapter 9, we further explore the consequences of our results in proof complexity. A highlight is that we show how to build proof complexity generators from demi-bits generators—even though, a priori, the definition of the former seemed much stronger than that of the latter.

1.5 List of Papers

The results in Chapter 3, Chapter 4, and Chapter 5 are from the following two papers:

- Hanlin Ren, Rahul Santhanam, and Zhikun Wang

  On the Range Avoidance Problem for Circuits [RSW22]

  Proceedings of *63rd IEEE Symposium on Foundations of Computer Science (FOCS 2022)*.

- Yeyuan Chen, Yizhi Huang, Jiatu Li, and Hanlin Ren

  Range Avoidance, Remote Point, and Hard Partial Truth Tables via Satisfying-Pairs Algorithms [CHLR23]

  Proceedings of *55th Annual ACM Symposium on Theory of Computing (STOC 2023)*.

The results in Chapter 6 are from the paper:

- Lijie Chen, Zhenjian Lu, Igor C. Oliveira, Hanlin Ren, and Rahul Santhanam

  Polynomial-Time Pseudodeterministic Construction of Primes [CLO+23]

  Proceedings of *64th IEEE Symposium on Foundations of Computer Science (FOCS 2023)*.

The results in Chapter 7 are from the paper:

- Lijie Chen, Shuichi Hirahara, and Hanlin Ren

  Symmetric Exponential Time Requires Near-Maximum Circuit Size [CHR24]

  Proceedings of *56th Annual ACM Symposium on Theory of Computing (STOC 2024)*.

Building on [CHR24], Zeyong Li [Li24] presents a simplified and improved circuit lower bound for $\mathsf{S}_2\mathsf{E}$. Li's proof gets rid of the win-win arguments completely, and implies a *truly-uniform* and *almost-everywhere* version of the lower bound: The hard language lies in $\mathsf{S}_2\mathsf{E}$ (instead of $\mathsf{S}_2\mathsf{E}/_1$, i.e., without the advice bit) and requires near-maximum circuit size for every input length (instead, our hard language in [CHR24] is only hard on infinitely many input lengths). The combined version of [CHR24] and [Li24] is published in *Journal of the ACM*, 2025 [CHLR25].

The results in Chapter 8 are from the paper:

- Zhenjian Lu, Igor C. Oliveira, Hanlin Ren, and Rahul Santhanam

  On the Complexity of Avoiding Heavy Elements [LORS24]

  Proceedings of *65th IEEE Symposium on Foundations of Computer Science (FOCS 2024)*.

The results in Chapter 9 are from the paper:

- Hanlin Ren, Yichuan Wang, and Yan Zhong

  Hardness of Range Avoidance and Proof Complexity Generators from Demi-Bits [RWZ26]

  To appear in proceedings of *17th Innovations in Theoretical Computer Science Conference (ITCS 2026)*.

Independently and concurrently, Ilango [Ila25] proved the same main result (that demi-bits generators imply hardness of Avoid) using different proof techniques.

The following papers are also coauthored during the author's doctoral studies, but are not directly related to the topic of this thesis (i.e., explicit constructions), hence are not included in this thesis:

- Yizhi Huang, Rahul Ilango, and Hanlin Ren

  NP-Hardness of Approximating Meta-Complexity: A Cryptographic Approach [HIR23]

  Proceedings of *55th Annual ACM Symposium on Theory of Computing (STOC 2023)*.

  SIAM Journal of Computing, 2025.

- Shuichi Hirahara, Zhenjian Lu, and Hanlin Ren

  Bounded Relativization [HLR23]

  Proceedings of *38th Computational Complexity Conference (CCC 2023)*.

- Noah Fleming, Stefan Grosser, Siddhartha Jain, Jiawei Li, Hanlin Ren, Morgan Shirley, and Weiqiang Yuan

  Total Search Problems in ZPP [FGJ+26]

  To appear in proceedings of *17th Innovations in Theoretical Computer Science Conference (ITCS 2026)*

- Lijie Chen, Yang Hu, and Hanlin Ren

  New Algebrization Barriers to Circuit Lower Bounds via Communication Complexity of Missing-String [CHR26]

  To appear in proceedings of *17th Innovations in Theoretical Computer Science Conference (ITCS 2026)*

- Jiawei Li, Yuhao Li, and Hanlin Ren

  Finding Bugs in Short Proofs: The Metamathematics of Resolution Lower Bounds [LLR24]

  Unpublished Manuscript.

- Michal Garlík, Svyatoslav Gryaznov, Hanlin Ren, and Iddo Tzameret

  The Weak Rank Principle: Lower Bounds and Applications [GGRT25]

  Unpublished Manuscript.

Chapter 2

Preliminaries

We use $\mathcal{U}_n$ to denote the uniform distribution over $\{0,1\}^n$. For a circuit $C : \{0,1\}^n \to \{0,1\}^\ell$, denote the range of C as

$$\mathrm{Range}(C) := \{C(x) : x \in \{0,1\}^n\}.$$

We use $\tilde{O}(f(n))$ to denote $f(n) \cdot (\log f(n))^{O(1)}$. The concatenation of the strings x and y is denoted by $x \circ y$.

Let S be a finite sample space and E be an event. We use $\Pr_{x \leftarrow S}[E]$ to denote the probability that E happens if x is sampled uniformly from S. Similarly, for a random variable Y, we use $\mathbb{E}_{x \leftarrow S}[Y]$ to denote the expectation of Y when x is sampled uniformly from S.

The *relative Hamming weight* of a string $x \in \{0,1\}^\ell$, denoted as $\delta(x)$, is the fraction of indices $i \in [\ell]$ such that $x_i = 1$. For two strings $x, y \in \{0,1\}^\ell$ of equal length, the *relative Hamming distance* between x and y, denoted as $\delta(x,y)$, is the fraction of indices $i \in [\ell]$ for which $x_i \neq y_i$. A string x is said to be γ-far from (resp. γ-close to) a string y if $\delta(x,y) \geq \gamma$ (resp. $\delta(x,y) < \gamma$). We say $x \in \{0,1\}^n$ is γ-far from $L \subseteq \{0,1\}^n$ if x is γ-far from every $y \in L$; otherwise x is γ-close to L. For a vector $\vec{u} \in \mathbb{R}^n$ and an integer $d \geq 1$, the ℓ_d *norm* of $\vec{u}$ is

$$\|\vec{u}\|_d := \left(\mathop{\mathbb{E}}_{i \leftarrow [n]} [|u_i|^d] \right)^{1/d}.$$

A function $f : \mathbb{N} \to \mathbb{N}$ is said to be *good* if there is a Turing machine such that given n in binary, it runs in time $\mathrm{poly}(\log n, \log f(n))$ and outputs $f(n)$ in binary.

A circuit class $\mathscr{C}$ is said to be *typical* if it contains the identity circuit and is closed under negations and projections. More precisely, (1) every function that always outputs its input bits is computable by a constant size $\mathscr{C}$ circuit; (2) for any $\mathscr{C}$ circuit C of size s and projection proj, both $\neg C$ and $C \circ \mathsf{proj}$ have $\mathscr{C}$ circuits of size $\mathrm{poly}(s)$, and the descriptions of these circuits can be computed in $\mathrm{poly}(s)$ time.

2.1 Circuit Classes

Throughout this thesis, the *size* of a circuit is defined as the number of *wires* (instead of gates) in the circuit. We will use the following (single-output) circuit classes.

- AC^0_d refers to depth-d circuits with AND and OR gates of unbounded fan-in, and NOT

gates of fan-in 1. We define $\mathsf{AC}^0 := \bigcup_{d\in\mathbb{N}} \mathsf{AC}^0_d$.

- $\mathsf{AC}^0_d[m]$ refers to depth-d circuits with AND, OR, and $\mathsf{MOD}[m]$ gates of unbounded fan-in, and NOT gates of fan-in 1. A $\mathsf{MOD}[m]$ gate outputs 1 if and only if m does not divide the number of 1 in its inputs. We define $\mathsf{AC}^0[m] := \bigcup_{d\in\mathbb{N}} \mathsf{AC}^0_d[m]$. Furthermore, we define $\mathsf{ACC}^0 := \bigcup_{m\in\mathbb{N}} \mathsf{AC}^0[m]$.
- $\mathsf{CC}^0_d[m]$ refers to depth-d circuits with only $\mathsf{MOD}[m]$ gates of unbounded fan-in. We define $\mathsf{CC}^0[m] := \bigcup_{d\in\mathbb{N}} \mathsf{CC}^0_d[m]$ and $\mathsf{CC}^0 := \bigcup_{m\in\mathbb{N}} \mathsf{CC}^0[m]$.
- NC^0_d refers to constant-size circuits such that the output depends on at most d input bits. We define $\mathsf{NC}^0 := \bigcup_{d\in\mathbb{N}} \mathsf{NC}^0_d$.
- Assume that $\mathsf{F} \in \{\mathsf{AND}, \mathsf{OR}, \mathsf{XOR}, \mathsf{MOD}[m], \dots\}$ is a gate, we define an F circuit as a circuit with only an F gate fed by some (or all) of the input bits. In particular, we define an F_d circuit as a circuit with an F gate of fan-in at most d fed by some (or all) of the input bits.

We define SYM as the class of any symmetric Boolean function, i.e., $f : \{0,1\}^n \to \{0,1\}$ such that $f(x) = g(x_1 + x_2 + \cdots + x_n)$ for some function g.

Suppose that $\mathscr{C}_1$ and $\mathscr{C}_2$ are circuit classes, we denote $\mathscr{C}_1 \circ \mathscr{C}_2$ as the *composition* of these two classes: the input bits feed an n-input m-output $\mathscr{C}_2$ circuit C_2, and the m output bits of C_2 feed an m-input single-output $\mathscr{C}_1$ circuit C_1. For instance, a $\mathsf{SYM} \circ \mathsf{ACC}^0$ circuit contains a symmetric output gate whose inputs are ACC^0 circuits.

For a circuit class $\mathscr{C}$, we use $\mathscr{C}[s]$ to represent the sub-class of $\mathscr{C}$ circuits of size at most s.

2.2 The Computational Models

We need to deal with two (nondeterministic) computational models: standard multi-tape Turing machines (TMs) and Random Access Machines (RAMs). The difference between TMs and RAMs is the following: for each work tape of an RAM, there is a corresponding address tape such that the head of the work tape is always in the cell whose index is the content of the address tape [GS89]. We need to be careful about the machine model we are using. For example, our ACC^0-Satisfying-Pairs algorithm runs in the RAM model, but the highly-efficient PCPP [BGH+05] works in the TM model.

Fortunately, we can use the following result to simulate one machine model by the other efficiently. Let $T(n)$ be a good function, we use $\mathsf{NTIME}_{\mathsf{TM}}[T(n)]$ and $\mathsf{NTIME}_{\mathsf{RAM}}[T(n)]$ to denote the set of languages computable in nondeterministic $T(n)$ time on a multi-tape Turing machine and an RAM respectively. Then we have:

Theorem 2.2.1 ([GS89])**.** $\bigcup_{c\geq 1} \mathsf{NTIME}_{\mathsf{TM}}[n\log^c n] = \bigcup_{c\geq 1} \mathsf{NTIME}_{\mathsf{RAM}}[n\log^c n]$.

By a padding argument, for some absolute constant $c \geq 1$, for every good function $T(n)$,

$$\mathsf{NTIME}_{\mathsf{TM}}[T(n)] \subseteq \mathsf{NTIME}_{\mathsf{RAM}}[T(n)\log^c T(n)],$$
$$\text{and } \mathsf{NTIME}_{\mathsf{RAM}}[T(n)] \subseteq \mathsf{NTIME}_{\mathsf{TM}}[T(n)\log^c T(n)].$$

2.3 Machines That Take Advice

Let C be a complexity class and $f : \mathbb{N} \to \mathbb{N}$. Denote $\mathsf{C}/_f$ the class of functions computable by a C machine with $f(n)$ bits of advice [KL80]. More formally, a language L is in $\mathsf{C}/_f$ if there is another language $L' \in \mathsf{C}$ and a sequence of "advice" strings $\left\{\alpha_n \in \{0,1\}^{f(n)}\right\}_{n\in\mathbb{N}}$ such that

$$\forall n \in \mathbb{N}, x \in \{0,1\}^n, x \in L \iff (1^n, x, \alpha_n) \in L'.$$

For example, $\mathsf{P}/_{\mathrm{poly}}$ denotes the class of languages computable in polynomial time given advice strings of polynomial length, which is equal to the class of languages with polynomial-size circuits [Pip79, KL80].

In particular, this thesis will deal with the following complexity classes with advice:

- In Chapter 3, we use $\mathsf{NTIMEGUESS}[T(n), n/10]/_{(n/10)}$ to denote the class of languages computed by a nondeterministic machine in $T(n)$ time, using $n/10$ nondeterministic bits and $n/10$ advice bits. (See Theorem 3.3.1.)
- In Chapter 7, we prove a circuit lower bound for the class $\mathsf{S_2E}/_1$, the class of languages computable in symmetric exponential time with one bit of advice.
- One of the assumptions used in Chapter 9 is the existence of demi-bits generators secure against $\mathsf{AM}/_{O(1)}$, the class of Arthur–Merlin adversaries with $O(1)$ advice bits (see Section 9.2.2 for a precise definition).

2.4 Error-Correcting Codes

An *error-correcting code* with *message length* n, *rate* r, and *relative distance* δ is a function $\mathsf{Enc} : \{0,1\}^n \to \{0,1\}^{rn}$ such that for every pair of distinct $x_1, x_2 \in \{0,1\}^n$, the Hamming distance between $\mathsf{Enc}(x_1)$ and $\mathsf{Enc}(x_2)$ is at least $\delta \cdot rn$. It is said to *correct* γ *fraction of errors* if there is a function $\mathsf{Dec} : \{0,1\}^{rn} \to \{0,1\}$ such that for every y that is γ-close to $\mathsf{Enc}(x)$ for some $x \in \{0,1\}^n$, $\mathsf{Dec}(y) = x$.

We need the following standard construction of error-correcting codes.

Theorem 2.4.1 ([Spi96])**.** *There is a* $\mathrm{GF}(2)$*-linear error-correcting code* $(\mathsf{Enc}, \mathsf{Dec})$ *with a constant rate and constant relative distance that can correct a constant fraction of errors. Moreover, both* Enc *and* Dec *are uniformly computable in linear time.*

2.5 Probabilistically Checkable Proofs of Proximity

We introduce Probabilistically Checkable Proofs of Proximity (PCPPs) [BGH+06] and the two properties of PCPPs that will be useful for us: *rectangularity* and *smoothness*.

In what follows, a *pair language* is simply a subset of $\{0,1\}^* \times \{0,1\}^*$. For an instance (z, x) of a pair language, we treat z as the *explicit input* (which the PCPP verifier can read entirely) and x as the *implicit input* (which the PCPP verifier could only read a few bits). For example, Circuit-Eval is a pair language with two inputs, i.e., a circuit C and an input x, and the task

is to evaluate $C(x)$. A PCPP verifier for Circuit-Eval knows the input circuit C entirely but can only access a few bits of x.

2.5.1 Basic Definitions

Definition 2.5.1 (PCP of Proximity Verifiers)**.** Let $r = r(n)$, $q = q(n)$, $\ell = \ell(n)$, $d = d(n)$ be good functions and $L \subseteq \{0,1\}^* \times \{0,1\}^*$ be a pair language. A PCPP verifier VPCPP for L with *proof length* ℓ, *randomness complexity* r, *decision complexity* d, and *query complexity* q is a tuple of Turing machines $(V_{\mathsf{type}}, V_{\mathsf{index}}, V_{\mathsf{dec}})$ that verify a proof $\pi \in \{0,1\}^\ell$ of the statement $(z, x) \in L \cap \{0,1\}^* \times \{0,1\}^n$ in the following fashion.

- It randomly samples a $\mathsf{seed} \in \{0,1\}^r$ and generates

$$(\mathsf{itype}[1], \mathsf{itype}[2], \ldots, \mathsf{itype}[q]) \leftarrow V_{\mathsf{type}}(\mathsf{seed}, z),$$
$$(i[1], i[2], \ldots, i[q]) \leftarrow V_{\mathsf{index}}(\mathsf{seed}, z).$$

  For every $j \in [q]$, $\mathsf{itype}[j] \in \{\mathsf{input}, \mathsf{proof}\}$ determines the type of the j-th query: If $\mathsf{itype}[j] = \mathsf{input}$, the j-th query probes the $i[j]$-th bit of the "implicit input" x; otherwise (i.e., $\mathsf{itype}[j] = \mathsf{proof}$), the j-th query probes the $i[j]$-th bit of the proof π.

- Let $\mathsf{ans}_1, \ldots, \mathsf{ans}_q$ be the answers to the queries defined above, we say VPCPP accepts (z, x, π), denoted by $\mathsf{VPCPP}^{x \circ \Pi}(z, \mathsf{seed}) = 1$, if and only if $V_{\mathsf{dec}}(\mathsf{seed}, z, \mathsf{ans}_1, \ldots, \mathsf{ans}_q)=1$. The machine V_{dec} is said to be the *decision predicate* of VPCPP, and has circuit complexity at most $d(n)$.

We may represent the "implicit input" x as $\Pi_{\mathsf{input}} : [n] \to \{0,1\}$ and the proof π as $\Pi_{\mathsf{proof}} : [\ell] \to \{0,1\}$ to emphasise that they are given as oracles to VPCPP. We sometimes denote the outputs of V_{type} and V_{index} as I and denote the answers $(\mathsf{ans}_1, \ldots, \mathsf{ans}_q)$ as $(\Pi_{\mathsf{input}} \circ \Pi_{\mathsf{proof}})|_I$.

We will also consider the PCPP verifier of *pure* languages (i.e. the first part z of any input is always the empty string). In such case, we simply omit all the z in the definition above.

Definition 2.5.2 (PCP of Proximity)**.** Let $s = s(n)$ and $\delta = \delta(n)$ be good functions, $L \subseteq \{0,1\}^* \times \{0,1\}^*$ be a pair language, and $\mathsf{VPCPP} = (V_{\mathsf{type}}, V_{\mathsf{index}}, V_{\mathsf{dec}})$ be a PCPP verifier for L. We say VPCPP is a PCPP verifier for L with *completeness error* $1 - c$, *soundness error* s, and *proximity parameter* δ if the following two conditions hold for every $(z, x) \in \{0,1\}^* \times \{0,1\}^n$.

- **(Completeness).** If $(z, x) \in L$, then there is a proof $\pi \in \{0,1\}^\ell$ such that VPCPP accepts (z, x, π) with probability at least c.

- **(Soundness).** Denote $L(z)$ to be the set of $y \in \{0,1\}^n$ such that $(z, y) \in L$. If x is δ-far from $L(z)$, then for every proof $\pi \in \{0,1\}^\ell$, VPCPP accepts (z, x, π) with probability at most s.

For most of the constructions of PCPPs, the completeness error can be made 0, which means that for $(z, x) \in L$, there is a proof such that the verifier accepts with probability 1. Therefore, we assume that the completeness error of a PCPP is 0 when it is not specified.

We need to define a stronger version of soundness called *robust soundness* as follows, as an intermediate step for constructing PCPPs with nice parameters.

Definition 2.5.3 (Robust PCP of Proximity [BGH+06])**.** Let $s = s(n)$, $\delta = \delta(n)$, and $\rho = \rho(n)$ be good functions, $L \subseteq \{0,1\}^* \times \{0,1\}^*$ be a pair language, and $\mathsf{VPCPP} = (V_{\mathsf{type}}, V_{\mathsf{index}}, V_{\mathsf{dec}})$ be a PCPP verifier for L. We say VPCPP is a robust PCPP verifier for L with *robust soundness error* s, *robustness parameter* ρ, and *proximity parameter* δ if it satisfies the completeness property of PCPP and the following *robust soundness property*.

- **(Robust Soundness).** The following holds for every $(z, x) \in \{0,1\}^* \times \{0,1\}^n$. Denote $L(z)$ to be the set of $y \in \{0,1\}^n$ such that $(z, y) \in L$. If x is δ-far from $L(z)$, then for every proof $\pi \in \{0,1\}^\ell$, with probability at least $1 - s$ over the random bits seed, the answer $(\mathsf{ans}_1, \ldots, \mathsf{ans}_q)$ of the queries of VPCPP is ρ-far from being accepted (i.e. we need to flip at least a ρ fraction of the bits of the answers $(\mathsf{ans}_1, \ldots, \mathsf{ans}_q)$ to make the verifier accept).

2.5.2 Rectangular PCPs of Proximity

Roughly speaking, a rectangular PCPP verifier [BHPT24] treats the input as an $H_{\mathsf{input}} \times W_{\mathsf{input}}$ matrix and the proof as an $H_{\mathsf{proof}} \times W_{\mathsf{proof}}$ matrix, and generates the query indices in a "rectangular" fashion. In particular, the random seed is split into two parts denoted as $\mathsf{seed.row}$ and $\mathsf{seed.col}$ respectively, and there are two algorithms V_{row} and V_{col} such that:

- V_{row} takes $\mathsf{seed.row}$ as input and generates $\mathsf{irow}[1], \ldots, \mathsf{irow}[q]$;
- V_{col} takes $\mathsf{seed.col}$ as input and generates $\mathsf{icol}[1], \ldots, \mathsf{icol}[q]$;
- The final indices of the queries $i[1], \ldots, i[q]$ are defined as $i[j] := (\mathsf{irow}[j] - 1) \cdot W + \mathsf{icol}[j]$, where $W = W_{\mathsf{input}}$ or $W = W_{\mathsf{proof}}$ depending on the the type of the j-th query.

In other words, the row verifier V_{row} (resp. the column verifier V_{col}) takes the row randomness $\mathsf{seed.row}$ (resp. the column randomness $\mathsf{seed.col}$) and generates the row indices (resp. the column indices) of the queries. Ideally, a rectangular PCPP should satisfy the following properties:

- **(Perfect Rectangularity).** The row randomness $\mathsf{seed.row}$ and column randomness $\mathsf{seed.col}$ are independent random bits (i.e. the row and column query indices are independent).
- **(Randomness-Oblivious Type Predicate).** The type predicate V_{type}, which determines the types of the queries (i.e. whether a query is to the input or the proof oracle), does not depend on the row and column random seeds.
- **(Randomness-Oblivious Decision Predicate).** The decision predicate V_{dec}, which decides whether to accept the proof given the answers to the queries, does not depend on the row and column random seeds.

However, as in [BHPT24], we do not know how to construct such rectangular PCPPs. Nevertheless, we could construct a weaker version where the row and column randomness are almost independent, and the dependencies of the decision and type predicates on the random seeds are relatively simple. In particular, the random seed of an *almost rectangular* PCPP is partitioned into three parts: $\mathsf{seed.row}$, $\mathsf{seed.col}$, and (a short portion) $\mathsf{seed.shared}$ and the following properties are satisfied:

- **(Almost Rectangularity).** V_{row} takes $\mathsf{seed.row}$ and $\mathsf{seed.shared}$ as inputs and generates the row query indices $(\mathsf{irow}[1], \mathsf{irow}[2], \ldots, \mathsf{irow}[q])$. Similarly, V_{col} takes $\mathsf{seed.col}$ and $\mathsf{seed.shared}$ as inputs and generates the column query indices $(\mathsf{icol}[1], \mathsf{icol}[2], \ldots, \mathsf{icol}[q])$.

- **(Randomness-Oblivious Type Predicate).** The type predicate V_{type} only depends on $\mathsf{seed.shared}$.
- **(Randomness-Oblivious Decision Predicate).** The decision predicate V_{dec} only depends on $\mathsf{seed.shared}$ and p additional *parity-check* bits, where each parity-check bit pc_i is a PARITY function over (a subset of indices in) $(\mathsf{seed.row}, \mathsf{seed.col})$.

We formally define such rectangular PCPPs as follows (for simplicity, we only define rectangular PCPPs for pure languages).

Definition 2.5.4 (Rectangular PCPPs with Randomness-Oblivious Predicates)**.** Let $L \subseteq \{0,1\}^*$ be a language, $H_{\mathsf{input}} = H_{\mathsf{input}}(n)$, $W_{\mathsf{input}} = W_{\mathsf{input}}(n)$, $H_{\mathsf{proof}} = H_{\mathsf{proof}}(n)$, $W_{\mathsf{proof}} = W_{\mathsf{proof}}(n)$, $r_{\mathsf{row}} = r_{\mathsf{row}}(n)$, $r_{\mathsf{col}} = r_{\mathsf{col}}(n)$, $r_{\mathsf{shared}} = r_{\mathsf{shared}}(n)$, and $p = p(n)$ be good functions such that $H_{\mathsf{input}} \cdot W_{\mathsf{input}} = O(n)$. A PCPP verifier VPCPP is said to be an almost rectangular PCPP with row randomness r_{row}, column randomness r_{col}, and shared randomness r_{shared}, that has a *randomness-oblivious predicate* (ROP) with parity-check complexity p, if the following hold.

- **(Randomness).** The randomness complexity is $r = r_{\mathsf{row}} + r_{\mathsf{col}} + r_{\mathsf{shared}}$ and the random seed can be partitioned into three independent parts: *row randomness* $\mathsf{seed.row} \in \{0,1\}^{r_{\mathsf{row}}}$, *column randomness* $\mathsf{seed.col} \in \{0,1\}^{r_{\mathsf{col}}}$, and *shared randomness* $\mathsf{seed.shared} \in \{0,1\}^{r_{\mathsf{shared}}}$.
- **(Query Pattern).** There are algorithms V_{type}, V_{row}, and V_{col} running in deterministic poly($|\mathsf{seed}|$) time that generates the queries in a rectangular fashion. More specifically:
  - $(\mathsf{itype}[1], \ldots, \mathsf{itype}[q]) \leftarrow V_{\mathsf{type}}(\mathsf{seed.shared})$, where $\mathsf{itype}[j] \in \{\mathsf{input}, \mathsf{proof}\}$ for all $j \in [q]$;
  - $(\mathsf{irow}[1], \ldots, \mathsf{irow}[q]) \leftarrow V_{\mathsf{row}}(\mathsf{seed.row}, \mathsf{seed.shared})$;
  - $(\mathsf{icol}[1], \ldots, \mathsf{icol}[q]) \leftarrow V_{\mathsf{col}}(\mathsf{seed.col}, \mathsf{seed.shared})$;
  - For every $j \in [q]$, the index of the j-th query is $i[j] := \mathsf{irow}[j] \cdot W + \mathsf{icol}[j]$, where $W = W_{\mathsf{input}}$ if $\mathsf{itype}[j] = \mathsf{input}$ and $W = W_{\mathsf{proof}}$ otherwise.

  Like normal PCPP verifiers, the j-th query is to the $i[j]$-th bit of the input if $\mathsf{itype}[j] = \mathsf{input}$, and is to the $i[j]$-th bit of the proof if $\mathsf{itype}[j] = \mathsf{proof}$. Note that since $H_{\mathsf{input}} \cdot W_{\mathsf{input}}$ may be larger than n, the query to the input is not well-defined when $i[j] > n$. In such case, we define the answer to be $\perp$.
- **(Decision Predicate).** There are algorithms V_{dec} and V_{pc} running in deterministic poly($|\mathsf{seed}|$) time such that the following holds.
  - The algorithm $V_{\mathsf{dec}}(\mathsf{seed.shared})$ generates a circuit $\mathsf{VDec} : \{0,1,\perp\}^{p+q} \to \{0,1\}$.
  - The algorithm $V_{\mathsf{pc}}(\mathsf{seed.shared})$ generates p XOR gates (i.e., GF(2)-linear functions) $pc_1, \ldots, pc_p : \{0,1\}^{r_{\mathsf{row}}+r_{\mathsf{col}}} \to \{0,1\}$.

  Assume that $(\mathsf{ans}_1, \ldots, \mathsf{ans}_q) \in \{0,1,\perp\}^q$ are the answers to the queries. For every $i \in [p]$, we denote $pc_i := pc_i(\mathsf{seed.row}, \mathsf{seed.col})$. The PCPP verifier accepts the proof if

$$\mathsf{VDec}(\mathsf{ans}_1, \ldots, \mathsf{ans}_q, pc_1, \ldots, pc_p) = 1.$$

  The *decision complexity* of this PCPP verifier is said to be the circuit complexity of V_{dec}.

Remark 2.5.5. For clarity, we now present the streamlined procedure of how a rectangular PCPP with randomness-oblivious predicate works:

1. Sample shared randomness $\mathsf{seed.shared} \in \{0,1\}^{r_{\mathsf{shared}}}$. Based on it,
   (a) Construct a decision predicate circuit $\mathsf{VDec} \leftarrow V_{\mathsf{dec}}(\mathsf{seed.shared})$.
   (b) Construct the parity-checks functions $(pc_1, \ldots, pc_p) \leftarrow V_{\mathsf{pc}}(\mathsf{seed.shared})$. Here, each pc_i is a PARITY function over (a subset of indices in) seed.row and seed.col.
   (c) Compute query types
   $$(\mathsf{itype}[1], \mathsf{itype}[2], \ldots, \mathsf{itype}[q]) \leftarrow V_{\mathsf{type}}(\mathsf{seed.shared}).$$
2. Sample row randomness $\mathsf{seed.row} \in \{0,1\}^{r_{\mathsf{row}}}$. Compute row indices
   $$(\mathsf{irow}[1], \mathsf{irow}[2], \ldots, \mathsf{irow}[q]) \leftarrow V_{\mathsf{row}}(\mathsf{seed.row}, \mathsf{seed.shared}).$$
3. Sample column randomness $\mathsf{seed.col} \in \{0,1\}^{r_{\mathsf{col}}}$. Compute column indices
   $$(\mathsf{icol}[1], \mathsf{icol}[2], \ldots, \mathsf{icol}[q]) \leftarrow V_{\mathsf{col}}(\mathsf{seed.col}, \mathsf{seed.shared}).$$
4. Compute randomness parity checks
   $$\mathrm{PC} := (pc_1(\mathsf{seed.row}, \mathsf{seed.col}), , \ldots, pc_p(\mathsf{seed.row}, \mathsf{seed.col})).$$
5. Compute $\mathsf{ans} := (\mathsf{ans}_1, \mathsf{ans}_2, \ldots, \mathsf{ans}_q)$ as in Definition 2.5.4.
6. Output the result of the computation $\mathsf{VDec}(\mathsf{ans}, \mathrm{PC})$.

In the Algorithmic Method, we also care about the circuit complexity of computing the query indices from the random seed. In fact, our rectangular PCPPs will have the *lowest* possible circuit complexity: the query indices are computable by a *projection* (i.e., NC^0 circuit of locality 1). More formally:

Definition 2.5.6. We say that the query indices of a rectangular PCPP *can be computed by (polynomial-time) projections* if for every seed.shared, the functions V^{row} and V^{col} (which maps seed.row and respectively seed.col to the row/column parts of query indices) are projections over seed.row and respectively seed.col, and moreover these projections can be computed in polynomial time given seed.shared.

2.5.3 Smooth PCP of Proximity

Apart from rectangularity, we also want our PCPPs to be *smooth*: each location is probed with equal probability.[1] The formal definition is as follows.

Definition 2.5.7 (Smooth PCPPs for Pure Languages)**.** Let $r = r(n)$, $q = q(n)$ be good functions, $L \subseteq \{0,1\}^*$ be a language, and $\mathsf{VPCPP} = (V_{\mathsf{type}}, V_{\mathsf{index}}, V_{\mathsf{dec}})$ be a PCPP verifier for L with randomness complexity r. We say VPCPP is a *smooth PCPP verifier* if for all locations $\mathsf{loc}_1, \mathsf{loc}_2$ in the proof oracle, over a uniformly random $\mathsf{seed} \leftarrow \{0,1\}^r$ and a uniformly random index $j \leftarrow [q]$, loc_1 and loc_2 are probed by VPCPP with equal probability in the j-th query.

[1] In some literature (e.g. [Par21]), the smoothness of PCPPs is defined differently: the queries to both the input oracle and the proof oracle need to be smooth, i.e., each location in the input (resp. the proof) is queried with equal probability. Here, we only require the queries to the proof oracle to be smooth and pose no requirement on the query distribution over the input oracle.

Smooth PCPPs can be viewed as PCPPs that can tolerate errors in the proof: since all the locations in the proof are queried with equal probability, a slightly corrupted version of a correct proof is still likely to be accepted, as shown in the following lemma.

Lemma 2.5.8. *Let $q = q(n)$, $\ell = \ell(n)$, $s = s(n)$ be good functions, $L \subseteq \{0,1\}^*$ be a language, and* VPCPP *be a smooth PCPP verifier for L with soundness error s, proof length ℓ, and query complexity q. Assume that $x \in L \cap \{0,1\}^n$ and $\pi \in \{0,1\}^\ell$ is a correct proof for $x \in L$, i.e.,* $\mathsf{VPCPP}^{x \circ \pi}(\mathsf{seed})$ *accepts with probability 1 over* $\mathsf{seed} \leftarrow \{0,1\}^r$. *Then for every π' such that the relative Hamming distance between π' and π is at most ε,* $\mathsf{VPCPP}^{x \circ \pi'}(\mathsf{seed})$ *accepts with probability at least $1 - q \cdot \varepsilon$ over* $\mathsf{seed} \leftarrow \{0,1\}^r$.

Proof. We say a location $i \in [\ell]$ of the proof oracle is *bad* if $\pi[i] \neq \pi'[i]$. Let B_j be the event that the j-th query of VPCPP probes a bad location in the proof. By the smoothness, we know that

$$\Pr_{\mathsf{seed} \leftarrow \{0,1\}^r, j \leftarrow [q]}[B_j] \leq \varepsilon.$$

By a union bound, we can see that

$$\Pr_{\mathsf{seed} \leftarrow \{0,1\}^r}[\exists j \in [q],\ B_j] \leq \sum_{j \in [q]} \Pr_{\mathsf{seed} \leftarrow \{0,1\}^r}[B_j] \leq q \cdot \varepsilon. \tag{2.1}$$

Denote E to be the event that there exists a $j \in [q]$ such that B_j happens. Then it follows that

$$\begin{aligned}
&\Pr_{\mathsf{seed} \leftarrow \{0,1\}^r}\left[\mathsf{VPCPP}^{x \circ \pi'}(\mathsf{seed}) \text{ rejects}\right] \\
\leq\ &\Pr_{\mathsf{seed} \leftarrow \{0,1\}^r}\left[\mathsf{VPCPP}^{x \circ \pi'}(\mathsf{seed}) \text{ rejects} \mid \neg E\right] + \Pr_{\mathsf{seed} \leftarrow \{0,1\}^r}[E] \\
\leq\ &0 + q \cdot e \\
=\ &q \cdot e,
\end{aligned}$$

where the second inequality follows from (2.1) and the perfect completeness of VPCPP. □

Note that smoothness can be defined for rectangular PCPPs, in which case each location in the proof matrix is probed with equal probability. This further means that each row (resp. column) index is queried by the row (resp. column) verifier with equal probability.

Remark 2.5.9. A stronger definition of smoothness is as follows: for every fixed $i \in [q]$, condition on the i-th query probing the proof oracle, the i-th query is uniformly random over the proof oracle. By randomly permuting the q queries, we can make a smooth PCPP satisfy this stronger definition of smoothness. In particular, if we have a smooth and rectangular PCPP, we can make it satisfy this stronger definition of smoothness by adding $O(q \log q)$ bits in the shared randomness for a random permutation over the q queries.

2.5.4 Our Constructions

In this thesis, we provide two new constructions of rectangular PCPPs. For solving the Range Avoidance problem and constructing worst-case hard partial truth tables, we need a rectangular PCPP with query complexity 3 or 2 (depending whether perfect completeness is required); for

solving the Remote Point problem and constructing average-case hard partial truth tables, we need a smooth and rectangular PCPP with query complexity $O(1)$.[2] These rectangular PCPPs are constructed in Chapter 5.

Theorem 2.5.10 (3-Query and 2-Query Rectangular PCPPs)**.** *For every constant $\delta \in (0,1)$, there are constants $s_3 \in (0,1)$ and $0 < s_2 < c_2 < 1$ such that the following holds. Let $m = m(n)$, $T(n)$, $w_{\mathsf{proof}}(n)$, $w_{\mathsf{input}}(n)$ be good functions such that $1 \le m \le (\log T(n))^{0.1}$, $n \le T(n) \le 2^{\mathrm{poly}(n)}$, $w_{\mathsf{proof}}(n) \le \log T(n)$, and $w_{\mathsf{input}}(n) \le \log n$. Then there are good functions $h^{\mathsf{3q}}_{\mathsf{proof}}(n)$, $h^{\mathsf{2q}}_{\mathsf{proof}}(n)$, and $h_{\mathsf{input}}(n)$ satisfying*

$$h^{\mathsf{3q}}_{\mathsf{proof}}(n), h^{\mathsf{2q}}_{\mathsf{proof}}(n) = \log T(n) + \Theta(m \log\log T(n)) - w_{\mathsf{proof}}(n), \qquad \text{and}$$
$$h_{\mathsf{input}}(n) = \lceil \log n \rceil - w_{\mathsf{input}}(n),$$

such that the following holds.

Suppose that $w_{\mathsf{proof}}, h^{\mathsf{3q}}_{\mathsf{proof}}, h^{\mathsf{2q}}_{\mathsf{proof}} \ge (5/m)\log T(n)$, and for some absolute constant $C \ge 1$,

$$\frac{w_{\mathsf{input}}(n)}{w_{\mathsf{proof}}(n)}, \frac{h_{\mathsf{input}}(n)}{h^{\mathsf{3q}}_{\mathsf{proof}}(n)}, \frac{h_{\mathsf{input}}(n)}{h^{\mathsf{2q}}_{\mathsf{proof}}(n)} \le 1 - \frac{Cm\log\log T(n)}{\log T(n)}.$$

Let $W_{\mathsf{proof}}(n) := 2^{w_{\mathsf{proof}}(n)}$, $H^{\mathsf{3q}}_{\mathsf{proof}}(n) := 2^{h^{\mathsf{3q}}_{\mathsf{proof}}(n)}$, $H^{\mathsf{2q}}_{\mathsf{proof}}(n) := 2^{h^{\mathsf{2q}}_{\mathsf{proof}}(n)}$, $W_{\mathsf{input}}(n) := 2^{w_{\mathsf{input}}(n)}$, and $H_{\mathsf{input}}(n) := 2^{h_{\mathsf{input}}(n)}$. Then $\mathsf{NTIME}[T(n)]$ has:

- *a rectangular PCP of proximity V_{3q} with perfect completeness, soundness error s_3, an $H^{\mathsf{3q}}_{\mathsf{proof}}(n) \times W_{\mathsf{proof}}(n)$ proof matrix and an $H_{\mathsf{input}}(n) \times W_{\mathsf{input}}(n)$ input matrix;*
- *a rectangular PCP of proximity V_{2q} with completeness error $1 - c_2$, soundness error s_2, an $H^{\mathsf{2q}}_{\mathsf{proof}}(n) \times W_{\mathsf{proof}}(n)$ proof matrix and an $H_{\mathsf{input}}(n) \times W_{\mathsf{input}}(n)$ input matrix.*

Other parameters of V_{3q} and V_{2q} are specified in Table 2.1.

Furthermore, given the randomness $\mathsf{seed} \in \{0,1\}^r$, the total number of queries and parity-check bits is at most 3 for V^{3q} and 2 for V^{2q}, and the decision predicate $\mathsf{VDec} \leftarrow V_{\mathsf{dec}}(\mathsf{seed.shared})$ of the rectangular PCPP verifier is an OR of the input bits (including queries and parity-check bits) or their negations for every $\mathsf{seed.shared}$. Also, the query indices of V_{3q} and V_{2q} can be computed by projections.

Theorem 2.5.11 (Smooth and Rectangular PCPP)**.** *For all constants $\delta \in (0,1)$ and $s \in (0,1)$, there is a constant $q \ge 1$ such that the following holds. Let $m = m(n)$, $T(n)$, $w_{\mathsf{proof}}(n)$, $w_{\mathsf{input}}(n)$ be good functions such that $1 \le m(n) \le (\log T(n))^{0.1}$, $n \le T(n) \le 2^{\mathrm{poly}(n)}$, $w_{\mathsf{proof}}(n) \le \log T(n)$, and $w_{\mathsf{input}}(n) \le \log n$. Then there are good functions $h_{\mathsf{proof}}(n)$ and $h_{\mathsf{input}}(n)$ satisfying*

$$h_{\mathsf{proof}}(n) := \log T(n) + \Theta(m \log\log T(n)) - w_{\mathsf{proof}}(n), \qquad \text{and}$$
$$h_{\mathsf{input}}(n) := \lceil \log n \rceil - w_{\mathsf{input}}(n),$$

such that the following holds.

[2] Unfortunately, our smooth PCPP requires a large (although constant) number of queries, because of the arguments in Section 5.1.4.

PCPP Verifier	$V^{3\mathsf{q}}$	$V^{2\mathsf{q}}$
Completeness error	0	$1 - c_2$
Soundness error	s_3	s_2
Proximity parameter	δ	
Row randomness	$h^{3\mathsf{q}}_{\mathsf{proof}} - (5/m)\log T(n)$	$h^{2\mathsf{q}}_{\mathsf{proof}} - (5/m)\log T(n)$
Column randomness	$w_{\mathsf{proof}} - (5/m)\log T(n)$	
Shared randomness	$(10/m)\log T(n) + O(\log\log T(n) + m\log m)$	
Query complexity	3	2
Parity check complexity		
Decision complexity	$\mathrm{poly}(\log\log T)$	

Table 2.1: Parameters of the PCPPs constructed in Theorem 2.5.10.

Suppose that $w_{\mathsf{proof}}, h_{\mathsf{proof}} \geq (5/m)\log T(n)$*, and that for some absolute constant* $C \geq 1$*,*

$$\frac{w_{\mathsf{input}}(n)}{w_{\mathsf{proof}}(n)}, \frac{h_{\mathsf{input}}(n)}{h_{\mathsf{proof}}(n)} \leq 1 - \frac{Cm^2 \log\log T(n)}{\log T(n)}.$$

Let $W_{\mathsf{proof}}(n) := 2^{w_{\mathsf{proof}}(n)}$*,* $H_{\mathsf{proof}}(n) := 2^{h_{\mathsf{proof}}(n)}$*,* $W_{\mathsf{input}}(n) := 2^{w_{\mathsf{input}}(n)}$*, and* $H_{\mathsf{input}}(n) := 2^{h_{\mathsf{input}}(n)}$*. Then* $\mathsf{NTIME}[T(n)]$ *has a smooth and rectangular PCP of proximity with an* $H_{\mathsf{input}}(n) \times W_{\mathsf{input}}(n)$ *input matrix and an* $H_{\mathsf{proof}}(n) \times W_{\mathsf{proof}}(n)$ *proof matrix, with query indices computable by projections, and whose other parameters are specified in Table 2.2.*

Soundness error	s
Proximity parameter	δ
Row randomness	$r_{\mathsf{row}} := h_{\mathsf{proof}} - (5/m)\log T(n)$
Column randomness	$r_{\mathsf{col}} := w_{\mathsf{proof}} - (5/m)\log T(n)$
Shared randomness	$r_{\mathsf{shared}} := (10/m)\log T(n) + O(\log\log T(n) + m\log m)$
Query complexity	$q = O_{s,\delta}(1)$
Parity check complexity	
Decision complexity	$\mathrm{poly}(T(n)^{1/m})$

Table 2.2: Parameters of the PCPP constructed in Theorem 2.5.11.

Remark 2.5.12 (Comparison with [BHPT24]). Our rectangular PCP of proximity differs from the rectangular PCP in [BHPT24] in the following ways.

- The biggest difference is that our construction is a smooth PCP of *proximity*. As a result, the input is also treated as a matrix, and its query pattern is also rectangular.
- The input matrix size and the proof matrix size in our rectangular PCPP are flexible, while the proof matrix in [BHPT24] is $\sqrt{m} \times \sqrt{m}$. It is easy to make the proof matrix size flexible, but more care needs to be taken for the input matrix. (See Section 5.1.2 where we artificially define a bijection called bin_{H^m}.) This is quite important as in our application, we need the input matrix width to be as small as possible!

Chapter 3

Range Avoidance via Satisfying-Pairs

3.1 Introduction

3.1.1 The Algorithmic Method

Building on his previous work [Wil13a], Williams [Wil14] famously proved that $\mathsf{NEXP} \not\subseteq \mathsf{ACC}^0$, the first non-uniform lower bound against the circuit class ACC^0. This lower bound is proved by designing a "non-trivial" satisfiability algorithm for ACC^0 circuits and then showing that such algorithms imply lower bounds against ACC^0. Indeed, the only property of ACC^0 that Williams uses is that ACC^0-SAT has a non-trivial algorithm; the algorithm-to-lower-bound connection works for any circuit class satisfying some mild technical conditions.

Let $\mathscr{C}$ be a circuit class and $\varepsilon(n) \in (0,1)$ be a parameter. Given as input a $\mathscr{C}$ circuit $C : \{0,1\}^n \to \{0,1\}$, we consider the following circuit-analysis problems:

Circuit-Analysis Problems

- ($\mathscr{C}$-SAT) decide whether there is an input $x \in \{0,1\}^n$ such that $C(x) = 1$;
- ($\mathscr{C}$-$\mathsf{GapUNSAT}_\varepsilon$) distinguish between the case that $\Pr_{x \leftarrow \{0,1\}^n}[C(x) = 1] \geq \varepsilon(n)$ and that C is unsatisfiable;
- ($\mathscr{C}$-$\#\mathsf{SAT}$) count the number of satisfying assignments of C;
- ($\mathscr{C}$-$\mathsf{CAPP}_\varepsilon$) estimate the quantity $\Pr_{x \leftarrow \{0,1\}^n}[C(x) = 1]$ within additive error ε.[a]

[a] CAPP stands for "circuit acceptance probability problem."

Clearly, all problems above can be solved in deterministic $2^n \cdot \mathrm{poly}(|C|)$ time by brute force. We say a deterministic algorithm for these problems is *non-trivial* if its time complexity is $2^n/n^{\omega(1)}$, i.e., slightly less than the brute-force time bound. The Algorithmic Method states that such algorithms imply circuit lower bounds against $\mathscr{C}$:

Theorem 3.1.1 (Informal; see [Wil14])**.** *Let $\mathscr{C}$ be a "nice" circuit class. If $\mathscr{C}$-SAT admits a deterministic algorithm running in $2^n/n^{\omega(1)}$ time, then $\mathsf{NEXP} \not\subseteq \mathscr{C}$.*

Since there exists a deterministic algorithm for ACC^0-SAT running in $2^{n-n^{\Omega(1)}}$ time [Wil14], it follows that $\mathsf{NEXP} \not\subseteq \mathsf{ACC}^0$.

There has been a long line of subsequent developments of the Algorithmic Method, both improving the algorithms and tightening the connection between algorithms and circuit lower bounds [SW13, BV14, Wil16, Wil18c, COS18, MW20, Wil18b, Che19, CW19b, VW20, Vio20, CR22, CLW20, CL21, CLLO21, Che23]. A recent highlight is the following result proved in [CLW20]:

Theorem 3.1.2 ([CLW20], Informal)**.** *There is a language in* E^{NP} *that does not have subexponential size* ACC^0 *circuits on almost every input length.*

Thinking of circuit lower bounds as explicit construction problems, [CLW20] gave an $\mathsf{FP}^{\mathsf{NP}}$-explicit construction of hard truth tables against subexponential size ACC^0 circuits. When formulated in the language of Range Avoidance, Theorem 3.1.2 becomes:

Theorem 3.1.3 (Theorem 3.1.2, Reformulated as Range Avoidance)**.** *Let* $s(n) := 2^{n^{o(1)}}$ *and* $\mathsf{TT}_{\mathsf{ACC}^0} : \{0,1\}^{O(s(n)\log s(n))} \to \{0,1\}^{2^n}$ *be the circuit that takes as input the description of a size-*$s(n)$ ACC^0 *circuit and outputs its truth table. Then, there is an* $\mathsf{FP}^{\mathsf{NP}}$ *algorithm for solving* Avoid *on the instance* $\mathsf{TT}_{\mathsf{ACC}^0}$*.*

Recently, the Algorithmic Method has found applications to another problem: constructing rigid matrices. Alman and Chen [AC19] showed how to construct rigid matrices in $\mathsf{FP}^{\mathsf{NP}}$ with parameters much better than previously known constructions; their results were later improved by [BHPT24, CLW20, HV21, CL21]. The key insight in [AC19] is to treat low-rank matrices as a special type of *circuit class*; thus, the task of constructing rigid matrices reduces to proving average-case circuit lower bounds against this class.

Given the success of the Algorithmic Method, it is natural to ask the following question:

Question 3.1.4. *Under which conditions can the Algorithmic Method be used to solve general explicit construction problems, such as* Avoid*?*

3.1.2 An Algorithmic Method for Range Avoidance

In this chapter, we present a version of the Algorithmic Method for solving Avoid. This method requires a non-trivial algorithm for the *Satisfying Pairs* problem:[1]

Problem 3.1.5 ($\mathscr{C}$-Satisfying-Pairs)**.** Let N, M, s, n be parameters. Given as inputs N $\mathscr{C}$ circuits $C_1, C_2, \ldots, C_N : \{0,1\}^n \to \{0,1\}$ of size s each and M strings $x_1, x_2, \ldots, x_M \in \{0,1\}^n$, compute or estimate

$$\Pr_{i \leftarrow [M], j \leftarrow [N]}[C_j(x_i) = 1]. \tag{3.1}$$

Analogous to the circuit-analysis problems such as SAT and CAPP defined in Section 3.1.1, we define the decisional and counting versions of the Satisfying Pairs problem as follows:

Satisfying Pairs Problems

- ($\mathscr{C}$-Satisfying-Pairs) decide whether (3.1) > 0;
- (Gap_ε-$\mathscr{C}$-Satisfying-Pairs) distinguish between (3.1) $= 0$ and (3.1) $> \varepsilon$;
- (#$\mathscr{C}$-Satisfying-Pairs) compute (3.1) exactly;
- ($\mathsf{Approx}_\varepsilon$-$\mathscr{C}$-Satisfying-Pairs) estimate (3.1) within additive error ε.

[1] We remark that our definition of $\mathscr{C}$-Satisfying-Pairs is different from the fine-grained complexity literature (e.g., [AHWW16, CW19a]). The input of the $\mathscr{C}$-Satisfying-Pairs problem defined in [AHWW16, CW19a] consists of a circuit $C(-,-)$ and two sets of input strings $\{a_i\}$ and $\{b_j\}$, and one wants to compute or approximate the number of pairs (i,j) such that $C(a_i, b_j) = 1$; in our $\mathscr{C}$-Satisfying-Pairs problem, we receive as input a list of circuits $\{C_i\}$ and a list of inputs $\{x_j\}$, and we want to compute or approximate the number of pairs (i,j) such that $C_i(x_j) = 1$. The new definition fits our purpose better. We also remark that for circuit classes that can "evaluate themselves" (such as $\mathsf{AC}^0, \mathsf{ACC}^0$, and TC^0), these two definitions are computationally equivalent.

We consider the regime where the input length n and the circuit size s are much smaller than N and M. In such case, a deterministic algorithm for $\mathscr{C}$-Satisfying-Pairs is said to be *non-trivial* if it runs in time $NM/\log^{\omega(1)}(NM)$.

Remark 3.1.6. The circuit-analysis problems that arise in the Algorithmic Method are special cases of Satisfying Pairs problems. For instance, we can solve #SAT of the circuit C by solving #Satisfying-Pairs with $N = 2^{n/2}$ and $M = 2^{n/2}$, where the inputs $(x_1, x_2, \ldots, x_M)$ consists of all strings of length $n/2$, and the circuits are $\{C_y : y \in \{0,1\}^{n/2}\}$, where $C_y(x) := C(x \circ y)$. This reduction shows that a non-trivial algorithm for Satisfying Pairs problem implies a non-trivial algorithm for the corresponding circuit-analysis problem.

Satisfying Pairs with P^{NP} preprocessing. In fact, our results also hold for Satisfying Pairs algorithms with P^{NP} *preprocessing* on circuits. Such an algorithm consists of two phases: First, the algorithm receives N $\mathscr{C}$ circuits $C_1, C_2, \ldots, C_N$, is allowed to preprocess them in some fixed polynomial $T = N^{O(1)}$ time deterministically with an NP oracle and produce a "data structure" DS of length at most T. Then the algorithm receives M inputs $x_1, x_2, \ldots, x_M$ and needs to output an estimation of (3.1) in non-trivial (i.e., $MN/\log^{\omega(1)}(MN)$) time, with the aid of DS.

Remark 3.1.7. The preprocessing phase might seem strange at first sight (especially since it is allowed to use an NP oracle), so we provide an example to motivate it. Suppose that $N \gg n$ (i.e., the number of circuits is much larger than the input length of these circuits) and we want to solve $\mathsf{Approx}_\varepsilon$-Satisfying-Pairs. It is easy to show (using a Chernoff bound and a union bound) that there is a subset $S \subseteq [N]$ of size $|S| = O(n\varepsilon^{-2})$ such that for every $x \in \{0,1\}^n$,

$$\left| \Pr_{i \leftarrow [N]}[C_i(x) = 1] - \Pr_{i \leftarrow S}[C_i(x) = 1] \right| \leq \varepsilon.$$

If for some polynomial T, such a subset S can be found in deterministic time $T(n)$ with an NP oracle, then there is a non-trivial algorithm for $\mathsf{Approx}_\varepsilon$-Satisfying-Pairs with P^{NP} preprocessing: In the preprocessing phase, given $\mathscr{C}$ circuits $C_1, C_2, \ldots, C_N$, we simply calculate S; in the query phase, given inputs $x_1, x_2, \ldots, x_M \in \{0,1\}^n$, we calculate

$$\Pr_{i \leftarrow S, j \leftarrow [M]}[C_i(x_j) = 1].$$

Since this algorithm takes $O(nM\varepsilon^{-2})$ time, it is non-trivial as long as $N \gg (n\varepsilon^{-2})^{1+\Omega(1)}$. In fact, a similar example will play an important role in Chapter 4 and characterise the "complete" Algorithmic Method for proving circuit lower bounds.

The main result of this chapter is that non-trivial algorithms for Satisfying Pairs imply $\mathsf{FP}^{\mathsf{NP}}$ algorithms for Avoid. Furthermore, this is true even if the Satisfying Pairs algorithm has a P^{NP} preprocessing phase.

Theorem 3.1.8 (Theorem 3.4.2, Informal)**.** *Let $\mathscr{C}$ be a typical circuit class and $\mathscr{C}' := \mathsf{OR}_2 \circ \mathscr{C}$.*[2] *Suppose that for every constant $\varepsilon > 0$ there is a non-trivial algorithm for $\mathsf{Approx}_\varepsilon$-$\mathscr{C}'$-Satisfying-Pairs with P^{NP} preprocessing, then $\mathscr{C}$-Avoid with certain parameters can be solved in $\mathsf{FP}^{\mathsf{NP}}$.*

[2] Here, $\mathsf{OR}_d \circ \mathscr{C}$ refers to the composition of a single fan-in-d OR gate being the output gate of the circuit and (at most) d $\mathscr{C}$ circuits feeding the top OR gate.

A note on the stretch functions. The above informal statement omitted the stretch of the $\mathscr{C}$ circuits for simplicity. Actually, even assuming the best possible algorithms for Satisfying Pairs, Theorem 3.1.8 could only solve the Range Avoidance problem for $\mathscr{C}$ circuits with stretch $\ell(n) = n^{1+\varepsilon}$. We refer to Theorem 3.4.2 for the precise statement.

3.1.3 Extension: The Remote Point Problem

The Algorithmic Method is extremely good at proving *average-case* circuit lower bounds [CR22, CLW20, CL21]. In this subsection, we define the *Remote Point* problem, which is an "average-case" analogue of the Range Avoidance problem, and present an Algorithmic Method for it. Here, for two strings $x, y \in \{0,1\}^n$, their *relative Hamming distance* is defined as the fraction of indices where x and y differ, formally $\delta(x,y) := \frac{1}{n}|\{i \in [n] : x_i \neq y_i\}|$.

Problem 3.1.9 (Remote Point Problem ($\mathscr{C}$-Remote-Point)). Given the description of a circuit $C : \{0,1\}^n \to \{0,1\}^\ell$ and a parameter $\delta > 0$, where each output bit of C is a $\mathscr{C}$ circuit, output any string $y \in \{0,1\}^\ell$ that is δ-far from the range of C. That is, for every $x \in \{0,1\}^n$, $\delta(C(x), y) \geq \delta$.

By Chernoff bound, if $\delta < 1/2 - c\sqrt{n/\ell}$ for some absolute constant $c > 0$, then a random length-ℓ string is a valid solution for Remote-Point with high probability. Therefore, the challenge is to find deterministic algorithms for Remote-Point.

It is not hard to see that $\mathscr{C}$-Remote-Point for the truth table generator TT corresponds to average-case circuit lower bounds. In particular, the regime where δ is a small constant corresponds to proving "weak" average-case lower bounds (e.g. [COS18, Che19]), and the regime where δ is close to $1/2$ (say, $\delta = 1/2 - 1/n$) corresponds to proving "strong" average-case lower bounds (e.g. [CR22, CLW20]).[3]

The remote point problem was also discussed in [KKMP21]. Indeed, an important special case of the problem has been studied by Alon, Panigrahy, and Yekhanin [APY09], namely the case that C is a linear transformation over GF(2). In other words, we are given a linear code $C : \{0,1\}^n \to \{0,1\}^\ell$ and we want to find a string that is far from every codeword. They introduced this problem as an intermediate step towards constructing rigid matrices.

It is already quite hard to solve this special case deterministically. Alon, Panigrahy, and Yekhanin [APY09] designed a polynomial-time algorithm for XOR-Remote-Point when $\ell > 2n$ and $\delta = O(\log n/n)$. For slightly larger δ, say $\delta = 0.1$, no deterministic algorithm is known even with an NP oracle. Arvind and Srinivasan [AS10] showed that for certain parameters, a polynomial-time algorithm for XOR-Remote-Point implies a polynomial-time algorithm for AC^0-Partial-Hard (defined later in Section 3.1.4).

In this chapter, we also extend our Algorithmic Method to the Remote Point problem. (Below, recall that a circuit class is *typical* if it contains the identity circuit and is closed under negations and projections.)

Theorem 3.1.10 (Theorem 3.5.1, Informal). *Let $\mathscr{C}$ be a typical circuit class and $\mathscr{C}' := \mathsf{AND}_{O(1)} \circ \mathscr{C}$. Suppose that there is a non-trivial algorithm for $\mathsf{Approx}_\varepsilon$-$\mathscr{C}'$-Satisfying-Pairs for every*

[3] Typically, a strong average-case lower bound states that certain problems cannot be $(1/2+1/s)$-approximated by size-s circuits. Suppose $\mathsf{TT} : \{0,1\}^n \to \{0,1\}^\ell$ is the truth table generator, then n is roughly the size of the circuit (i.e., $n \approx s$). In this regard, strong average-case circuit lower bounds correspond to Remote-Point where $\delta = 1/2 - 1/n$.

*constant $\varepsilon > 0$, then $\mathscr{C}$-*Remote-Point *with certain parameters can be solved in* $\mathsf{FP}^{\mathsf{NP}}$.

In particular, suppose for every constant $\varepsilon > 0$, there is a non-trivial algorithm for $\mathsf{Approx}_{\varepsilon}$*-$\mathscr{C}'$-*Satisfying-Pairs *for $N = \text{quasi-poly}(n)$ $\mathscr{C}'$-circuits of size $O(s)$ and $M = \text{quasi-poly}(n)$ inputs of length $n \cdot \text{polylog}(n)$; then for some stretch function $\ell = \text{quasi-poly}(n)$, there is an* $\mathsf{FP}^{\mathsf{NP}}$ *algorithm for $\mathscr{C}$-*Remote-Point *that takes as input a circuit $C : \{0,1\}^n \to \{0,1\}^\ell$ where each output bit of C is a $\mathscr{C}$-circuit of size s, and outputs a y that is 0.49-far from $\text{Range}(C)$.*

Our framework provides Remote-Point algorithms for the regime corresponding to "strong average-case lower bounds" [CR22], i.e., the distance between the output y and $\text{Range}(C)$ is close to $1/2$. In fact, the distance can be as large as $1/2 - 1/\text{poly}(n)$ given an Approx-$\mathscr{C}$-Satisfying-Pairs algorithm with a small enough error. (see Theorem 3.5.1 for details).

The stretch of circuits for which we can solve Remote-Point via Theorem 3.1.10 is worse than that for Theorem 3.1.8: Even assuming the best possible Satisfying Pairs algorithms, we could only solve Remote-Point for circuits with *quasi-polynomial* stretch. This is because we use an approximate list-decodable code with *linear-sum* decoders in [CLW20] which has a quasi-polynomial rate. It is an interesting open problem to improve the stretch of Remote-Point that can be solved by our framework, possibly by designing new linear-sum decodable codes with a better rate; see, e.g., [CL21].

3.1.4 Extension: Hard Partial Truth Tables

Besides Avoid and Remote-Point, we also consider the following problem that generalises the task of proving circuit lower bounds (in a different way):

Problem 3.1.11 (Hard Partial Truth Tables against $\mathscr{C}$ ($\mathscr{C}$-Partial-Hard))**.** Given a list of input strings $z_1, z_2, \ldots, z_\ell \in \{0,1\}^n$ and a parameter s, find a list of output bits $b_1, b_2, \ldots, b_\ell \in \{0,1\}$ such that the partial function defined by $\{(z_i, b_i)\}_{i \in [\ell]}$ cannot be computed by $\mathscr{C}$ circuits of size s. In other words, for every size-s $\mathscr{C}$ circuit C, there exists an index $i \in [\ell]$ such that $C(z_i) \neq b_i$.

It is easy to see that $\mathscr{C}$-Partial-Hard generalises the problem of proving circuit lower bounds against $\mathscr{C}$. Indeed, if we take $\ell := 2^n$ and $z_1, z_2, \ldots, z_\ell$ be an enumeration of length-n strings, then $\mathscr{C}$-Partial-Hard becomes exactly the problem of proving circuit lower bounds against $\mathscr{C}$. It is also easy to see that when $\ell > O(s \log s)$, this problem reduces to Avoid: given the input $(z_1, z_2, \ldots, z_\ell)$, we can construct a circuit $\mathsf{TT}' : \{0,1\}^{O(s \log s)} \to \{0,1\}^\ell$ which takes the description of a $\mathscr{C}$ circuit C as input, and outputs the concatenation of $C(z_1), C(z_2), \ldots, C(z_\ell)$. Finding a non-output of TT' is equivalent to finding a solution of $\mathscr{C}$-Partial-Hard.

This problem was introduced by Arvind and Srinivasan [AS10] under the name "circuit lower bounds with help functions." Let $h_1, h_2, \ldots, h_n : \{0,1\}^m \to \{0,1\}$ denote a sequence of *help functions*, $\mathscr{C}$ be a circuit class, and $s \in \mathbb{N}$ be a size parameter. The goal is to construct the truth table of a function $f : \{0,1\}^m \to \{0,1\}$ that is hard to compute for size-s $\mathscr{C}$ circuits, even when the circuit has access to these help functions. Formally, for any size-s circuit $C : \{0,1\}^n \to \{0,1\}$, there exists an input $x \in \{0,1\}^m$ such that

$$C(h_1(x), h_2(x), \ldots, h_n(x)) \neq f(x).$$

This problem is equivalent to Partial-Hard with $\ell = 2^m$ inputs of length n, namely for every

$x \in \{0,1\}^m$, there is an input $h_1(x) \circ h_2(x) \circ \cdots \circ h_n(x) \in \{0,1\}^n$ in the Partial-Hard instance.

This problem appears to be very hard. Neither [AS10] nor we are aware of an efficient deterministic solution for $\mathscr{C} = \mathsf{AC}^0$ with (say) $\ell, s \in$ quasi-poly(n). That is, although exponential-size lower bounds against AC^0 are known [Ajt83, FSS84, Yao85, Hås89], we do not have any idea about how to prove such a lower bound for partial functions. Even when $\mathscr{C}$ is the class of *polynomial-size DNF*, to the best of our knowledge, there is no known deterministic algorithm for $\mathscr{C}$-Partial-Hard.

Besides being a natural problem itself, $\mathscr{C}$-Partial-Hard also arises when we study the closure of non-uniform complexity classes (under reductions). Recall that AC^0 denotes the class of languages computable by a *non-uniform* family of polynomial-size constant-depth circuits; in particular, AC^0 contains undecidable languages such as unary versions of the halting problem. A language L Turing-reduces to some language in AC^0 if and only if $L \in \mathsf{P}/_{\text{poly}}$ [Pip79], thus proving $\mathsf{EXP} \not\leq_T^p \mathsf{AC}^0$ is likely beyond current techniques. But what about *mapping reducibility*? Can we show that $\mathsf{EXP} \not\leq_m^p \mathsf{AC}^0$? It turns out that a deterministic algorithm for AC^0-Partial-Hard implies that $\mathsf{EXP} \not\leq_m^p \mathsf{AC}^0$ [AS10, Theorem 5]. Of course, there is nothing special with AC^0, and it can be replaced by other non-uniform classes. Therefore, $\mathscr{C}$-Partial-Hard sheds light on ruling out many-one reducibility of EXP (and other complexity classes) to non-uniform classes.

We also define the average-case version of $\mathscr{C}$-Partial-Hard, which requires us to construct partial functions that are *average-case* hard against $\mathscr{C}$:

Problem 3.1.12 (Average-Case Hard Partial Truth Tables against $\mathscr{C}$ ($\mathscr{C}$-Partial-AvgHard))**.** Given a list of input strings $z_1, z_2, \ldots, z_\ell \in \{0,1\}^n$ and parameters s, δ, find a list of output bits $b_1, b_2, \ldots, b_\ell \in \{0,1\}$ such that the partial function defined by $\{(z_i, b_i)\}_{i \in [\ell]}$ is δ-far from being computable by $\mathscr{C}$ circuits of size s. In other words, for every size-s $\mathscr{C}$ circuit C, there are at least $\delta\ell$ indices $i \in [\ell]$ such that $C(z_i) \neq b_i$.

Similar to the frameworks for Avoid and Remote-Point, we can solve Partial-Hard and Partial-AvgHard via non-trivial algorithms for Satisfying-Pairs.

Theorem 3.1.13 (Informal)**.** *Let $\mathscr{C}$ be a typical circuit class.*

- *Suppose that there is a non-trivial algorithm for $\mathsf{Approx}_\varepsilon$-$\mathscr{C}'$-Satisfying-Pairs for every $\varepsilon > 0$ and $\mathscr{C}' := \mathsf{OR}_2 \circ \mathscr{C}$, then $\mathscr{C}$-Partial-Hard with certain parameters can be solved in $\mathsf{FP}^{\mathsf{NP}}$.*
- *Suppose that there is a non-trivial algorithm for $\mathsf{Approx}_\varepsilon$-$\mathscr{C}''$-Satisfying-Pairs for every $\varepsilon > 0$ and $\mathscr{C}'' := \mathsf{AND}_{O(1)} \circ \mathscr{C}$, then $\mathscr{C}$-Partial-AvgHard with certain parameters can be solved in $\mathsf{FP}^{\mathsf{NP}}$.*

These results are proved using essentially the same approach as the framework for Avoid and Remote-Point; consequently, the trade-off between parameters for Satisfying-Pairs and Partial-Hard (resp. Partial-AvgHard) is similar to that for Avoid (resp Remote-Point). We omit the details and refer the readers to Theorem 3.6.2 and Theorem 3.7.1.

Remark 3.1.14. It is not surprising to have a unified framework for Avoid and Partial-Hard (and their average-case analogues Remote-Point and Partial-AvgHard), since Avoid and Partial-Hard can be considered as the *dual* problem of each other. Let $\mathsf{Eval} : \{0,1\}^{O(s \log s)} \times \{0,1\}^n \to \{0,1\}$ be the circuit-evaluation function that takes a circuit C of size s and an input of

length n, and outputs $C(x)$. We can interpret Avoid and Partial-Hard as follows:

- **(Avoid).** Given size-s circuits $C_1, C_2, \ldots, C_\ell$, find $y_1, y_2, \ldots, y_\ell \in \{0,1\}$ such that for every $x \in \{0,1\}^n$, there is an $i \in [\ell]$ such that $\mathsf{Eval}(C_i, x) \neq y_i$.
- **(Partial-Hard).** Given inputs $x_1, x_2, \ldots, x_\ell \in \{0,1\}^n$, find $y_1, y_2, \ldots, y_\ell \in \{0,1\}$ such that for every size-s circuit C, there is an $i \in [\ell]$ such that $\mathsf{Eval}(C, x_i) \neq y_i$.

Clearly, Avoid and Partial-Hard are essentially the same problem on the table $\mathsf{Eval}(\cdot,\cdot)$ with the rows and columns being exchanged.

3.1.5 Unconditional Results

By slightly adapting the technique introduced by Williams [Wil18c] to design non-trivial #SAT algorithms for ACC^0 circuits (which uses an earlier quasi-polynomial size simulation of $\mathsf{SYM} \circ \mathsf{ACC}^0$ circuits by $\mathsf{SYM} \circ \mathsf{AND}$ circuits [BT94, AG91]), we obtain a non-trivial algorithm for #ACC^0-Satisfying-Pairs:

Theorem 3.1.15. *For every constants m, ℓ, c, there is a constant $\varepsilon \in (0,1)$ such that the following holds. Let $n := 2^{\log^\varepsilon N}$ and $s := 2^{\log^c n}$. There is a deterministic algorithm running in $\tilde{O}((N/n)^2)$ time that given N strings $x_1, x_2, \ldots, x_N \in \{0,1\}^n$ and N $\mathsf{AC}^0_\ell[m]$ circuits $C_1, C_2, \ldots, C_N : \{0,1\}^n \to \{0,1\}$ of size s, outputs the number of pairs $(i,j) \in [N] \times [N]$ such that $C_i(x_j) = 1$.*

The $\mathsf{FP}^{\mathsf{NP}}$ algorithm for ACC^0-Remote-Point and ACC^0-Partial-AvgHard follows from this algorithm together with Theorem 3.1.10 and Theorem 3.1.13.

Theorem 3.1.16 (ACC^0-Remote-Point $\in \mathsf{FP}^{\mathsf{NP}}$)**.** *There is a constant $c_u \geq 1$ such that for every constant $d, m \geq 1$, there is a constant $c_{\mathsf{str}} := c_{\mathsf{str}}(d,m) \geq 1$, such that the following holds.*

Let $n < s(n) \leq 2^{n^{o(1)}}$ be a size parameter, $\varepsilon := \varepsilon(n) \geq 2n^{-c_u}$ be an error parameter and $\ell := \ell(n) \geq 2^{\log^{c_{\mathsf{str}}} s}$ be a stretch function, then there is an $\mathsf{FP}^{\mathsf{NP}}$ algorithm that takes as input a circuit $C : \{0,1\}^n \to \{0,1\}^\ell$, where each output bit of C is computed by an $\mathsf{AC}^0_d[m]$ circuit of size s, and outputs a string y that is $(1/2 - \varepsilon)$-far from $\mathrm{Range}(C)$.

Theorem 3.1.17 (ACC^0-Partial-AvgHard $\in \mathsf{FP}^{\mathsf{NP}}$)**.** *There is a constant $c_u \geq 1$ such that for every constants $d, m \geq 1$, there is a constant $c_{\mathsf{str}} := c_{\mathsf{str}}(d,m) \geq 1$, such that the following holds.*

Let $n < s(n) \leq 2^{n^{o(1)}}$ be a size parameter, $\varepsilon := \varepsilon(n) \geq 2n^{-c_u}$ be an error parameter and $\ell := \ell(n) \geq 2^{\log^{c_{\mathsf{str}}} s}$ be a stretch function, then there is an $\mathsf{FP}^{\mathsf{NP}}$ algorithm that given inputs $x_1, \ldots, x_\ell \in \{0,1\}^n$, it outputs a string $y \in \{0,1\}^\ell$ such that for any $s(n)$-size $\mathsf{AC}^0_d[m]$ circuit C, y is $(1/2 - \varepsilon)$-far from $C(x_1) \circ \cdots \circ C(x_\ell)$.

It is worth noting that the ACC^0-Remote-Point algorithm here recovers the best known almost-everywhere average-case circuit lower bounds against ACC^0 [CLW20]. This is done by considering the special case where the input circuit is the truth table generator $\mathsf{TT} : \{0,1\}^{O(s \log s)} \to \{0,1\}^{2^n}$ that prints the truth table of a given ACC^0 circuit (see Section 3.8.2).

Corollary 3.1.18. *For every constants $d, m \geq 1$, there is an $\varepsilon > 0$ and a language $L \in \mathsf{E}^{\mathsf{NP}}$ such that L_n cannot be $(1/2 + 2^{-n^\varepsilon})$-approximated by $\mathsf{AC}^0_d[m]$ circuits of size 2^{n^ε}, for all sufficiently large n.*

Following the observation of Arvind and Srinivasan [AS10], the $\mathsf{FP}^{\mathsf{NP}}$ algorithm for ACC^0-Partial-AvgHard can be used to prove *unconditionally* that E^{NP} cannot be mapping reduced to languages decidable by small-size *non-uniform* families of ACC^0 circuits.[4] To the best of our knowledge, this is the first unconditional result ruling out the mapping reducibility from uniform classes to non-trivial non-uniform classes.

Corollary 3.1.19. *Let $d, m \in \mathbb{N}$ be constants, $\mathsf{AC}^0_d[m]$ denote the class of languages computable by a* non-uniform *family of polynomial-size $\mathsf{AC}^0_d[m]$ circuits. Then, there is a language $L^{\mathsf{hard}} \in \mathsf{E}^{\mathsf{NP}}$ that does not have polynomial-time mapping reductions to any language in $\mathsf{AC}^0_d[m]$.*

3.2 Technical Overview

In this subsection, we present an overview of the proof of Theorem 3.1.8.

It would be helpful to review the Algorithmic Method for proving E^{NP} lower bounds. Let $L^{\mathsf{hard}} \in \mathsf{NTIME}[2^n] \setminus \mathsf{NTIME}[o(2^n)]$ be a hard language constructed by the nondeterministic time hierarchy theorem [Zák83]. Let V be the PCP verifier of [BGH+06]; here V is an oracle circuit $V^{(-)} : \{0,1\}^r \to \{0,1\}$. This oracle circuit takes PCP randomness as input (so the input length is $r := n + O(\log n)$) and receives the PCP proof as the oracle.

For a proof oracle $\pi : \{0,1\}^r \to \{0,1\}$, denote $p_{\mathsf{acc}}(\pi) := \Pr_{\mathsf{seed} \leftarrow \{0,1\}^r}[V^{\pi}(\mathsf{seed}) \text{ accepts}]$. The PCP theorem guarantees that for every input $x \in \{0,1\}^\star$:

- If $x \in L^{\mathsf{hard}}$, then there is a proof oracle π such that $p_{\mathsf{acc}}(\pi) = 1$.
- If $x \notin L^{\mathsf{hard}}$, then for every proof oracle π, we have $p_{\mathsf{acc}}(\pi) \leq 0.01$.

Now, suppose that for every input $x \in L^{\mathsf{hard}}$, there is a proof oracle π such that $p_{\mathsf{acc}}(\pi) = 1$, and in addition, *$\pi$ can be computed by a $\mathscr{C}$ circuit.* (Call this assumption the "easy-witness assumption".) Moreover, suppose that the GapUNSAT problem for $V^{\mathscr{C}}$ can be solved in $2^r/r^{\omega(1)} < o(2^n)$ time. Then there is a faster nondeterministic algorithm for L^{hard} as follows. Given an input x, we first guess a circuit $\mathscr{C}$ that computes a valid proof oracle π, and use the GapUNSAT algorithm to distinguish between the case that $p_{\mathsf{acc}}(\pi) = 1$ and that $p_{\mathsf{acc}}(\pi) \leq 0.01$.

By the nondeterministic time hierarchy theorem, the above speed-up algorithm has to be incorrect. Therefore, our "easy-witness assumption" has to be false, i.e., there is an input $x \in L^{\mathsf{hard}}$ which does not have valid PCP proofs computable by a small $\mathscr{C}$ circuit.

A naïve attempt. Given a circuit $C : \{0,1\}^n \to \{0,1\}^\ell$, our goal is to find a non-output of C in $\mathsf{FP}^{\mathsf{NP}}$. Again, let $L^{\mathsf{hard}} \in \mathsf{NTIME}[\ell] \setminus \mathsf{NTIME}[o(\ell)]$ be the hard language constructed by the nondeterministic time hierarchy.[5] Our "easy-witness" assumption now becomes:

Assumption 3.2.1. For every $x \in L^{\mathsf{hard}}$, there is a PCP proof for x that is in the range of C.

Now we design a faster nondeterministic algorithm M_{fast} that tries to solve L^{hard}. For every PCP randomness $\mathsf{seed} \in \{0,1\}^r$, let $Q^{\mathsf{seed}} : \{0,1\}^\ell \to \{0,1\}$ be the circuit that takes a PCP

[4] In fact, it suffices to have an $\mathsf{FP}^{\mathsf{NP}}$ algorithm for ACC^0-Partial-Hard (which is a trivial consequence of an $\mathsf{FP}^{\mathsf{NP}}$ algorithm for ACC^0-Partial-AvgHard) for this application.

[5] Note that we have not specified the input length for L^{hard}. We only know that L^{hard} is in non-deterministic ℓ time on this input length. This important issue will be discussed later.

proof $\pi \in \{0,1\}^\ell$ as input[6] and outputs whether the verifier accepts π when given seed as the randomness. Suppose we guess $w \in \{0,1\}^n$ such that $C(w)$ is the PCP proof for $x \in L^{\mathsf{hard}}$, then it suffices to estimate $\Pr_{\mathsf{seed}}[Q^{\mathsf{seed}}(C(w)) = 1]$ to verify this PCP proof.

However, there is a serious problem with this approach: The description length of C is already $\Omega(\ell)$, therefore it is impossible to estimate $\Pr[Q^{\mathsf{seed}}(C(w)) = 1]$ in $o(\ell)$ time.

Idea 1: Making copies. Our first idea is simple but crucial: we pick a large enough number $H = \mathrm{poly}(\ell)$ and make H copies of C. That is, instead of the Range Avoidance problem for C, we consider the Range Avoidance problem for the circuit

$$C^H(x_1, x_2, \ldots, x_H) = (C(x_1), C(x_2), \ldots, C(x_H)).$$

There is a simple $\mathsf{FP}^{\mathsf{NP}}$ reduction from avoiding C to avoiding C^H. Suppose $y = (y_1, \ldots, y_H)$ is not in the range of C^H, then we can use the NP oracle to check whether each y_i is in the range of C and pick the first y_i that is not. Hence, it suffices to solve the Range Avoidance problem for C^H.

Now, let $L^{\mathsf{hard}} \in \mathsf{NTIME}[H \cdot \ell] \setminus \mathsf{NTIME}[o(H \cdot \ell)]$. Let $Q^{\mathsf{seed}} : \{0,1\}^{H \cdot \ell} \to \{0,1\}$ be the circuit that accepts a PCP proof $\pi \in \{0,1\}^{H \cdot \ell}$ if and only if the PCP verifier accepts it given seed as the PCP randomness. After guessing $w \in \{0,1\}^{nH}$ (which corresponds to the PCP proof $C^H(w)$), It suffices to estimate $p_{\mathsf{est}} := \Pr_{\mathsf{seed}}[Q^{\mathsf{seed}}(C^H(w)) = 1]$. The good news is that we only need $\approx \ell \ll H \cdot \ell$ bits to describe the circuit C^H, thus, at least in principle, it could be possible to estimate p_{est} in less than $H \cdot \ell$ time.

But how do we *actually* estimate p_{est}? It seems likely that we need to exploit some special properties of the PCP verifier. What property should our PCP have?

Idea 2: Rectangular PCP. Our second idea is to use *rectangular* PCPs [BHPT24]. In this overview, let us assume the PCP is *perfectly* rectangular, which means that the PCP proof π is formatted as an $H \times \ell$ matrix and the PCP randomness seed is partitioned into two parts: $\mathsf{seed.row}$ and $\mathsf{seed.col}$. More importantly, the row index of each query only depends on $\mathsf{seed.row}$ and the column index of each query only depends on $\mathsf{seed.col}$. Correspondingly, our easy-witness assumption becomes that every row of π is in the range of C.

Suppose the easy-witness assumption holds and there are strings $w_1, w_2, \ldots, w_H$ such that the i-th row of π is equal to $C(w_i)$. Each $\mathsf{seed.row}$ corresponds to q rows $r_1, r_2, \ldots, r_q$ such that $V^\pi(\mathsf{seed.row}, -)$ will only access these rows of π. Hence, we define the following input corresponding to $\mathsf{seed.row}$:

$$\mathsf{Input}_{\mathsf{seed.row}} := (w_{r_1}, w_{r_2}, \ldots, w_{r_q}).$$

Similarly, each $\mathsf{seed.col}$ corresponds to q columns $c_1, c_2, \ldots, c_q$ such that $V^\pi(-, \mathsf{seed.col})$ will only access these columns of π. Let $Q^{\mathsf{seed.col}} : \{0,1\}^{q\ell} \to \{0,1\}$ be the circuit that takes the q rows (determined by $\mathsf{seed.row}$) of π as inputs and outputs whether the verifier accepts when the

[6] The length of the PCP proofs is slightly larger than ℓ; using efficient PCPs [BGH+05, BS08, Din07], one can achieve PCP proof length $\ell \cdot \mathrm{polylog}(\ell)$. In this informal exposition, we do not distinguish between the time complexity of L^{hard} and the length of its PCP proofs.

column randomness is $\mathsf{seed.col}$. We define the circuit

$$C^{\mathsf{seed.col}}(w_1, w_2, \ldots, w_q) := Q^{\mathsf{seed.col}}(C(w_1), C(w_2), \ldots, C(w_q)).$$

After guessing $w_1, w_2, \ldots, w_H \in \{0,1\}^n$ such that the i-th row of π is equal to $C(w_i)$, we need to test whether the verifier accepts. Note that for each $\mathsf{seed} = (\mathsf{seed.row}, \mathsf{seed.col})$, $V^\pi(\mathsf{seed})$ accepts if and only if $C_{\mathsf{seed.col}}(\mathsf{Input}^{\mathsf{seed.col}}) = 1$. Therefore, it suffices to solve an instance of Satisfying Pairs with $2^{|\mathsf{seed.row}|}$ many inputs and $2^{|\mathsf{seed.col}|}$ many circuits. The time complexity of M_{fast} is

$$2^{|\mathsf{seed.row}|} \cdot 2^{|\mathsf{seed.col}|}/(|\mathsf{seed.row}| \cdot |\mathsf{seed.col}|)^{\omega(1)} \leq (H\ell)/\log^{\omega(1)}(H\ell).$$

The "right" time hierarchy theorem. The above Range Avoidance algorithm is only correct on infinitely many input lengths. This is because the nondeterministic time hierarchy in [Zák83] only works infinitely often, i.e., for any $\mathsf{NTIME}[o(H\ell)]$ machine M, L^{hard} and M only disagree on infinitely many input lengths.

To obtain an almost-everywhere algorithm, we follow the ideas of [CLW20]. The crucial observation is that M_{fast} does not guess too many nondeterministic bits. (In the case of the Algorithmic Method, it only guesses a small circuit encoding the PCP proof; in our case, it only guesses $Hn \ll H\ell$ bits.) There is an almost-everywhere nondeterministic time hierarchy against such machines [FS16]. Let $\mathsf{NTIMEGUESS}[T(N), g(N)]$ denote the class of languages decidable by a nondeterministic machine running in $T(N)$ time and guessing $g(N)$ bits. Then:

Theorem 3.2.2 ([FS16])**.** *Let $T(N)$ be a time-constructible function such that $N \leq T(N) \leq 2^{\mathrm{poly}(N)}$. There is a language $L^{\mathsf{hard}} \in \mathsf{NTIME}[T(N)] \setminus \text{i.o.-}\mathsf{NTIMEGUESS}[o(T(N)), N/10]$.*

Since we need to guess Hn bits, we set the input length to be $N := 10Hn$. We also set $T(N)$ to be a slightly super-linear function such that $T(10Hn) \approx H\ell$.

There is a small issue: M_{fast} needs to access the circuit C. More generally, if our Satisfying Pairs algorithm has P^{NP} preprocessing, then M_{fast} needs to access the "data structure" DS produced in the preprocessing phase. Fortunately, the above NTIME hierarchy theorem also holds against machines with $N/10$ advice bits, and we can hardwire C or DS as nonuniform advice.

Theorem 3.2.3. *Let $T(N)$ be a time-constructible function such that $N \leq T(N) \leq 2^{\mathrm{poly}(N)}$. There is a language $L^{\mathsf{hard}} \in \mathsf{NTIME}[T(N)] \setminus \text{i.o.-}\mathsf{NTIMEGUESS}[o(T(N)), N/10]/\underline{(N/10)}$.*

Our $\mathsf{FP}^{\mathsf{NP}}$ algorithm needs one more ingredient from [CLW20]: a *refuter* for Theorem 3.2.3. Given 1^N and the code of the machine M_{fast} that attempts to compute L^{hard}, as well as the $N/10$ advice bits, if M_{fast} runs in $o(T(N))$ time and uses at most $N/10$ nondeterministic bits, then the refuter finds an input $x \in \{0,1\}^N$ such that $M_{\mathsf{fast}}(x) \neq L^{\mathsf{hard}}(x)$. The refuter runs in polynomial time with access to an NP oracle.

Our $\mathsf{FP}^{\mathsf{NP}}$ algorithm for Avoid works as follows. We first compute (the code of) the machine M_{fast}; recall that it is a machine in $\mathsf{NTIMEGUESS}[o(T(N)), N/10]/_{(N/10)}$. (This includes running the P^{NP} preprocessing phase of our Satisfying Pairs algorithm if there is one, and hardwiring the circuit C and the DS it produces into the code of M_{fast}.) Then we use the refuter to find

an input $x_{\mathsf{hard}} \in \{0,1\}^N$ such that $M_{\mathsf{fast}}(x_{\mathsf{hard}}) \neq L^{\mathsf{hard}}(x_{\mathsf{hard}})$. It follows that in any valid proof matrix of $x_{\mathsf{hard}} \in L^{\mathsf{hard}}$, there is some row that is not in the range of C. We can then simply use the NP oracle to pick the first such row.

Rectangular PCP of proximity. There is another issue: the PCP verifier depends on the input x_{hard}. As x_{hard} might depend on DS (recall that x_{hard} is found by the refuter, which takes the code of M_{fast} as input, and M_{fast} needs to hardcode DS), we cannot preprocess the circuits $\{C^{\mathsf{seed.col}}\}$ before we know x_{hard}.

Our solution is to use a rectangular PCP of *proximity* (henceforth rectangular PCPP). Recall that a PCPP verifier can only query a small number of bits in both the proof oracle and the input oracle. (As it does not even have time to read the whole input, its query pattern does not depend on it.) In a rectangular PCPP, the input oracle is also accessed in a rectangular fashion. There are three predicates V_{type}, V_{row}, and V_{col}:[7]

- V_{type}, without looking at seed, outputs q symbols, where each symbol is either input or proof.
- V_{row} reads $\mathsf{seed.row}$ and outputs q row indices $r_1, r_2, \ldots, r_q$.
- V_{col} reads $\mathsf{seed.col}$ and outputs q column indices $c_1, c_2, \ldots, c_q$.
- For each query $i \in [q]$, if the i-th symbol is input, then the i-th query asks the (r_i, c_i)-th entry of the input matrix; if the i-th symbol is proof, then the i-th query asks the (r_i, c_i)-th entry of the proof matrix.

We now revise our speed-up algorithm M_{fast} for L^{hard} using rectangular PCPPs. Given an input $x \in \{0,1\}^N$,[8] we still guess $w_1, w_2, \ldots, w_H$ and construct the PCPP proof matrix π whose i-th row is $C(w_i)$. Also, the input is treated as an $H' \times W'$ matrix[9]; let x_i be the i-th row of the input matrix. Now we estimate the probability that $V^{x,\pi}(\mathsf{seed})$ accepts, where V is the PCPP verifier with oracle access to x and π.

- Each $\mathsf{seed.row}$ corresponds to q_{proof} rows in the proof matrix and q_{input} rows in the input matrix, where $q_{\mathsf{proof}} + q_{\mathsf{input}} = q$, and the output of $V^{x,\pi}(\mathsf{seed.row}, -)$ only depends on these rows. Let these indices be $i_1, \ldots, i_q$, we define

$$\mathsf{Input}_{\mathsf{seed.row}} := (w_{i_1}, \ldots, w_{i_{q_{\mathsf{proof}}}}, x_{i_{q_{\mathsf{proof}}+1}}, \ldots, x_{i_q}).$$

- For each $\mathsf{seed.col}$, let $Q^{\mathsf{seed.col}} : \{0,1\}^{q_{\mathsf{proof}} \cdot \ell + q_{\mathsf{input}} \cdot W'} \to \{0,1\}$ be the circuit that takes these rows as inputs and outputs whether the verifier accepts them when the column randomness

[7] Note that we consider perfect rectangularity here for simplicity. In an almost rectangular PCPP, the randomness also contains a short part denoted as $\mathsf{seed.shared}$ that is read by every predicate. In particular, V_{type} depends on $\mathsf{seed.shared}$, V_{row} depends on $\mathsf{seed.row}$ and $\mathsf{seed.shared}$, and V_{col} depends on $\mathsf{seed.col}$ and $\mathsf{seed.shared}$.

[8] Note that a PCPP could only distinguish between $x \in L$ and x being *far* from L. Thus, we need to apply an error-correcting code to the input. For simplicity, we still use x to denote the encoded input.

[9] A technicality here is that we want to set W' to be as small as possible, as the size of $C'_{\mathsf{seed.col}}$ is proportional to W'. It turns out that we can achieve $W' = n \cdot \mathrm{polylog}(\ell)$.

is $\mathsf{seed.col}$.[10] We define

$$C^{\mathsf{seed.col}}(w_1, \ldots, w_{q_{\mathsf{proof}}}, x_1, \ldots, x_{q_{\mathsf{input}}}) = Q^{\mathsf{seed.col}}(C(w_1), \ldots, C(w_{q_{\mathsf{proof}}}), x_1, \ldots, x_{q_{\mathsf{input}}}).$$

Note that the description of $C^{\mathsf{seed.col}}$ itself does not depend on x_{hard} now.

It follows that

$$\Pr_{\mathsf{seed}}[V^{x,\pi}(\mathsf{seed}) = 1] = \Pr_{\mathsf{seed.row}}\left[\Pr_{\mathsf{seed.col}}\left[C^{\mathsf{seed.col}}(\mathsf{Input}_{\mathsf{seed.row}}) = 1\right]\right],$$

which, by our Satisfying Pairs algorithms, can be estimated in time

$$2^{|\mathsf{seed.row}|} \cdot 2^{|\mathsf{seed.col}|}/(|\mathsf{seed.row}| \cdot |\mathsf{seed.col}|)^{\omega(1)} < H\ell/\log^{\omega(1)}(H\ell).$$

Finally, our $\mathsf{FP}^{\mathsf{NP}}$ avoidance algorithm is the same as before, except that we use the rectangular PCPP in the code of M_{fast}.

3.2.1 Extensions to Remote Point and Hard Partial Truth Tables

Remote Point Problem. Our start point is the following reduction from Remote-Point to Avoid. Suppose that $C : \{0,1\}^n \to \{0,1\}^\ell$ is the input circuit. Let $\mathsf{Enc} : \{0,1\}^{\ell'} \to \{0,1\}^\ell$ be the encoding procedure of an error-correcting code, and $\mathsf{Dec} : \{0,1\}^\ell \to \{0,1\}^{\ell'}$ be the corresponding decoding procedure, where Dec can correct a δ fraction of errors. Define the circuit $C'(x) := \mathsf{Dec}(C(x))$, and let z be any string not in the range of C', then $\mathsf{Enc}(z)$ is $(1-\delta)$-far from $\mathrm{Range}(C)$. To see this, assume for contradiction that $\mathsf{Enc}(z)$ is $(1-\delta)$-close to some $C(x)$, then $\mathsf{Dec}(C(x))$ should return exactly z, contradicting that z is a non-output of C'.

Suppose that the function Dec can be implemented in the circuit class $\mathscr{C}_{\mathsf{Dec}}$, then this is a reduction from $\mathscr{C}$-Remote-Point to $(\mathscr{C}_{\mathsf{Dec}} \circ \mathscr{C})$-Avoid. Therefore, we would like the complexity of $\mathscr{C}_{\mathsf{Dec}}$ to be as small as possible. There are decoders that tolerate a small constant fraction of errors in AC^0 [GGH+07], so it might be possible to implement $\mathscr{C}_{\mathsf{Dec}}$ in AC^0. However, when δ is very close to $1/2$ (say $\delta = 1/2 - \varepsilon$), we enter the list-decoding regime where $\mathscr{C}_{\mathsf{Dec}}$ seems to need the power of majority [GR08]. Can we solve $\mathscr{C}$-Remote-Point without invoking any circuit-analysis algorithms for $\mathsf{MAJ} \circ \mathscr{C}$?

Fortunately, the required techniques already appeared in previous works on the Algorithmic Method for proving strong average-case circuit lower bounds. In [CLW20], they provided an error-correcting code that corrects a $1/2 - \varepsilon$ fraction of errors, where the decoder $\mathsf{Dec}_{\mathsf{CLW}}$ can be implemented as a *linear sum*, i.e., each output is a linear combination of the input bits.[11] Intuitively, this means that we can reduce $\mathscr{C}$-Remote-Point to $(\mathsf{Sum} \circ \mathscr{C})$-Avoid, where Sum denotes the layer of $\mathsf{Dec}_{\mathsf{CLW}}$. Using the framework for Range Avoidance established above, it suffices to solve the Satisfying-Pairs problem for $\mathsf{Sum} \circ \mathscr{C}$ circuits.[12] But it is easy to

[10] Actually, $Q^{\mathsf{seed.col}}$ also depends on $O(q)$ *parity-check* bits. We ignore this technical detail in the overview.

[11] [CLW20] stated this result as a non-standard XOR lemma in their Appendix A. We re-prove it in the form of error-correcting codes in Section 3.3.2.

[12] We made a simplification here. Actually, we need to solve Satisfying-Pairs for $\mathsf{NC}^0 \circ \mathsf{Sum} \circ \mathscr{C}$ circuits. Using the distributive property, we can push the NC^0 circuits below the Sum layer, thus it suffices to solve Satisfying-Pairs for $\mathsf{Sum} \circ \mathsf{NC}^0 \circ \mathscr{C}$ circuits. In this informal exposition, we may assume that $\mathscr{C}$ is closed under top NC^0 gates, which means that a Satisfying-Pairs algorithm for $\mathsf{Sum} \circ \mathscr{C}$ now suffices.

see that Satisfying-Pairs for $\mathsf{Sum} \circ \mathscr{C}$ circuits directly reduces to Satisfying-Pairs for $\mathscr{C}$ circuits! Therefore, the error-correcting code in [CLW20] allows us to use an algorithm for $\mathscr{C}$-Satisfying-Pairs to directly solve $\mathscr{C}$-Remote-Point, with little or no circuit complexity overhead.

The above discussion omitted several important technical details:

- It turns out that $\mathsf{Dec}_{\mathsf{CLW}}$ is only an *approximate* list-decoding algorithm: given a corrupted codeword that is $(1/2 - \varepsilon)$-close to the correct codeword, we can only recover a message that is δ-close to the correct message (instead of perfectly recovering the correct message).

  This drawback is handled by *smooth PCPPs* [Par21], which has the property that any slightly corrupted version of a correct proof is still accepted with good probability. In fact, we actually need a *smooth and rectangular PCPP*, which we construct in Section 5.1. We remark that [CLW20] also encountered this difficulty; they got around it by combining a PCP and a PCPP for Circuit-Eval. It is not clear how to generalise this strategy to our case.

- Another technical complication is that $\mathsf{Dec}_{\mathsf{CLW}}$ outputs *real values* instead of Boolean values. It is only guaranteed that the decoded message is close to the original message *in ℓ_1-norm*. Consequently, after guessing the PCPP proof, we also need to verify that it is "close to Boolean", This difficulty also appears in [CLW20]; however, we need to carefully define what it means by "close to Boolean" in our case.

- Since $\mathsf{Dec}_{\mathsf{CLW}}$ works in the list-decoding regime, it also receives an advice string (specifying the index of the codeword in the list). In the above discussion, we omitted the advice string to highlight the main ideas. It turns out that the dependency of the decoder on the advice string *cannot* be captured by linear sums. This is why we define an ad-hoc "linear sum" circuit class (see Section 3.3.2) that receives both an input and an advice string and computes a linear combination over the input, where the "linear combination" depends on the advice. It turns out that we need the dependency on the advice to be *local* (see Section 3.3.2 for details), which is fortunately satisfied by the XOR-Lemma-based code in [CLW20].

Another reduction via succinct dictionaries. We mention that there is another reduction from Remote-Point to Avoid which appears in [Kor21, GLW22]. Let $C : \{0,1\}^n \to \{0,1\}^\ell$ be a circuit, $y \in \{0,1\}^\ell$ be a string that is not δ-far from $\mathrm{Range}(C)$. Then we can find a string $x \in \{0,1\}^n$ and a "noise" string $e \in \{0,1\}^m$ of relative Hamming weight at most δ such that $y = C(x) \oplus e$, where $\oplus$ refers to bit-wise XOR. Consider the circuit $C'(x, e) := C(x) \oplus e$. To solve the remote point problem for C, it suffices to solve the Range Avoidance problem for C'. Using a "succincter" dictionary to represent e [Păt08], [GLW22] managed to show that this reduction also preserves circuit complexity, and in particular reduces NC^1-Remote-Point to NC^1-Avoid.

A drawback of this approach is that it only reduces Remote-Point to Range Avoidance instances with a small stretch. Indeed, suppose C' is a circuit from n' inputs to ℓ outputs, and $\delta = \Omega(1)$, then

$$n' \geq |\Pi(e)| \geq \log \binom{\ell}{\delta\ell} = \Omega(\ell).$$

In contrast, our algorithmic method could not solve Range Avoidance instances with such a small stretch ($\ell = c \cdot n$ for some constant c), even with the best possible algorithms for Satisfying-Pairs. Therefore, we do not use this approach here.

Hard Partial Truth Tables. There is a simple reduction from Partial-Hard to Avoid. Suppose we are given strings $x_1, x_2, \ldots, x_N$. Let TT' be the circuit that receives a size-s circuit C as input, and outputs the concatenation of $C(x_1)$, $C(x_2)$, $\ldots$, $C(x_N)$. If $N > O(s \log s)$, then the circuit TT' is stretching. It is also easy to see that solving the Range Avoidance of TT' is equivalent to solving the Partial-Hard problem.

In Section 3.6, we essentially combine this reduction with the frameworks in Section 3.4. In other words, we could have reduced Partial-Hard to Avoid in a black-box way and derived the main results in Section 3.6. However, this reduction only reduces $\mathscr{C}$-Partial-Hard to $\mathscr{C}'$-Avoid, where $\mathscr{C}'$ is any circuit class that can solve $\mathscr{C}$-Eval in the following sense: for every fixed input x, there is a $\mathscr{C}'$ circuit C' that takes as input the description of a $\mathscr{C}$ circuit C, and outputs $C(x)$. For most circuit classes of interest (e.g., $\mathscr{C} \in \{\mathsf{AC}^0, \mathsf{ACC}^0, \mathsf{NC}^1, \mathsf{P}_{/\mathrm{poly}}\}$), we could simply let $\mathscr{C}' = \mathscr{C}$; however, this is not necessarily true for more refined circuit classes (such as $\mathscr{C} = \mathsf{ACC} \circ \mathsf{THR}$). We choose to derive the main results in Section 3.6 from scratch instead of reducing it to Section 3.4, partly because we also want our framework to hold for these more refined circuit classes.

3.3 Preliminaries

3.3.1 An Almost-Everywhere NTIME Hierarchy with a Refuter

We need the almost-everywhere NTIME hierarchy against bounded nondeterminism [FS16], which has an $\mathsf{FP}^{\mathsf{NP}}$ refuter as shown in [CLW20]. Let $T(n), G(n)$ be good functions, we define $\mathsf{NTIME}[T(n)]$ to be the class of languages decidable by nondeterministic Turing machines in $T(n)$ time, and $\mathsf{NTIMEGUESS}_{\mathsf{RAM}}[T(n), G(n)]$ to be the class of languages decidable by nondeterministic Random-Access Turing Machines (RAMs) in $T(n)$ time with $G(n)$ nondeterministic bits.

Theorem 3.3.1 ([FS16, CLW20])**.** *Let c be a large universal constant, $T : \mathbb{N} \to \mathbb{N}$ be a good function such that $n \log^{c+1} n \le T(n) \le 2^{\mathrm{poly}(n)}$. There is a language*

$$L^{\mathsf{hard}} \in \mathsf{NTIME}[T(n)] \setminus \text{i.o.-}\mathsf{NTIMEGUESS}_{\mathsf{RAM}}[T(n)/\log^c T(n), n/10]/_{(n/10)}.$$

Moreover, there is an algorithm $\mathcal{R}$ (the "refuter") such that the following holds.

(Input) *$\mathcal{R}$ receives three inputs $(1^n, M, \alpha)$, where M is a nondeterministic RAM and $\alpha \in \{0,1\}^{n/10}$ is an advice string. It is guaranteed that M runs in $T(n)/\log^c T(n)$ time and uses at most $n/10$ nondeterministic bits; moreover, the description length of M is $O(1)$.*

(Output) *For every fixed M, every sufficiently large n, and every advice $\alpha \in \{0,1\}^{n/10}$, $\mathcal{R}(1^n, M, \alpha)$ outputs a string $x \in \{0,1\}^n$ such that $M(x; \alpha) \neq L^{\mathsf{hard}}(x)$.*

(Complexity) *$\mathcal{R}$ runs in $\mathrm{poly}(T(n))$ time with adaptive access to an NP oracle.*

We provide a proof of Theorem 3.3.1 here for completeness; in particular, we show that the time hierarchy and the refuter could also deal with $N/10$ advice bits. (This essentially follows from the original proofs in [FS16, CLW20].)

We need the following binary search algorithm.

Lemma 3.3.2 ([CLW20, Lemma 4.4]). *There is an algorithm A satisfying the following.*

- **Input.** *A is given an explicit integer $n \geq 2$ (written in binary form) as input, together with oracle access to a list $(a_1, a_2, \ldots, a_n) \in \{0,1\}^n$ such that $a_1 \neq a_n$.*
- **Output.** *An index $p \in [1, n-1]$ such that $a_p \neq a_{p+1}$.*
- **Efficiency.** *A runs in $O(\log n)$ time and makes at most $O(\log n)$ queries to the list.*

Proof of Theorem 3.3.1. We first define L^{hard}. Let $x \in \{0,1\}^n$ be the input of L^{hard}, we parse it into $x := (\langle M \rangle, \alpha, w, x_{\mathsf{rest}})$. Here, $\langle M \rangle$ is the description of a nondeterministic RAM M, $\alpha \in \{0,1\}^{n/10}$ is the advice string for M, $w \in \{0,1\}^{n/10}$ is a witness of M, and x_{rest} denotes the rest input bits. We interpret M as a nondeterministic RAM that guesses at most $n/10$ nondeterministic bits and runs in at most $T' := T/\log^c T$ time; if the nondeterminism or time complexity exceeds the corresponding bounds, we force M to reject.

(i) If M accepts the input $(\langle M \rangle, \alpha, 0^{n/10}, x_{\mathsf{rest}})$ with witness w and advice α, then $L^{\mathsf{hard}}(x) = 0$.

(ii) Otherwise, if $w = 1^{n/10}$, then $L^{\mathsf{hard}}(x) = 1$.

(iii) Otherwise, let $w+1$ be the lexicographically next string after w, $L^{\mathsf{hard}}(x) = 1$ if and only if M accepts $(\langle M \rangle, \alpha, w+1, x_{\mathsf{rest}})$ with advice α.

Since a nondeterministic RAM of time complexity $T' = T/\log^c T$ can be simulated by a nondeterministic TM of time complexity $o(T)$, it follows that $L^{\mathsf{hard}} \in \mathsf{NTIME}_{\mathsf{TM}}[T(n)]$.

Before describing the refuter, it is instructive to understand why L^{hard} does not admit a nondeterministic RAM algorithm with time T', $n/10$ nondeterministic bits, and $n/10$ advice bits. Let M be such an algorithm and $\alpha \in \{0,1\}^{n/10}$ be the corresponding advice string. For the sake of contradiction, suppose M computes L^{hard}. Fix an arbitrary x_{rest}. For a witness string $w \in \{0,1\}^{n/10}$, denote $x_w := (\langle M \rangle, \alpha, w, x_{\mathsf{rest}})$. Abusing notation, for an integer $0 \leq i < 2^{n/10}$, let w_i be the $(i+1)$-th lexicographically smallest string (with $w_0 = 0^{n/10}$ and $w_{2^{n/10}-1} = 1^{n/10}$), we also denote $x_i := x_{w_i}$.

- Suppose $M(x_0)$ accepts. Let w be the lexicographically smallest witness w such that M accepts x_0 on witness string w. It follows from (i) that $L^{\mathsf{hard}}(x_w) = 0$. However, for every $w' < w$, since M does not accept $x_{w'}$, by (iii) we have that

$$L^{\mathsf{hard}}(x_{w'}) = M(x_{w'+1}) = L^{\mathsf{hard}}(x_{w'+1}).$$

  It follows that $1 = M(x_0) = L^{\mathsf{hard}}(x_0) = L^{\mathsf{hard}}(x_w) = 0$, a contradiction.

- Suppose $M(x_0)$ rejects. Then M rejects x_0 on every possible witness, which means (i) never happens. It follows from (iii) that for every $w \in \{0,1\}^{n/10} \setminus \{1^{n/10}\}$,

$$L^{\mathsf{hard}}(x_w) = M(x_{w+1}) = L^{\mathsf{hard}}(x_{w+1}).$$

This is a contradiction as $L^{\mathsf{hard}}(x_0) = 0$ but $L^{\mathsf{hard}}(x_{1^{n/10}}) = 1$.

Now we describe our refuter. On input $(1^n, M, \alpha)$, our refuter $\mathcal{R}$ first uses the NP oracle to decide if $M(x_0)$ accepts.

- If $M(x_0)$ accepts, then it uses the NP oracle to find the lexicographically smallest w such that M accepts x_0 on witness string w. Consider the list

$$1 = M(x_0), M(x_1), \ldots, M(x_w), L^{\mathsf{hard}}(x_w) = 0.$$

  We use Lemma 3.3.2 to find two adjacent entries in the list that are different. This takes $O(\log(2^n)) = O(n)$ time with random access to the list. As random access to the list can be simulated by an NP oracle, this pair of entries can be found in polynomial time with an NP oracle. Now there are two cases:

  - (Case I) Suppose the two entries are $M(x_w)$ and $L^{\mathsf{hard}}(x_w)$. The refuter simply outputs x_w.
  - (Case II) Suppose the two entries are $M(x_{w'})$ and $M(x_{w'+1})$ where $w' < w$. Since M does not accept x_0 on witness string w' and $w' < 1^{n/10}$, (iii) applies to w'. Therefore $L^{\mathsf{hard}}(x_{w'}) = M(x_{w'+1}) \neq M(x_{w'})$, and the refuter can output $x_{w'}$.

- If $M(x_0)$ rejects, then consider the list

$$0 = M(x_0), M(x_1), \ldots, M(x_{1^{n/10}}), L^{\mathsf{hard}}(x_{1^{n/10}}) = 1.$$

  Again, we use Lemma 3.3.2 to find two adjacent entries in the list that are different, in polynomial time with an NP oracle. There are two cases:

  - (Case I) Suppose the two entries are $M(x_{1^{n/10}})$ and $L^{\mathsf{hard}}(x_{1^{n/10}})$. The refuter simply outputs $x_{1^{n/10}}$.
  - (Case II) Suppose the two entries are $M(x_w)$ and $M(x_{w+1})$ for some $w \in \{0,1\}^{n/10} \setminus \{1^{n/10}\}$. Since M does not accept x_0 (at all) and $w < 1^{n/10}$, (iii) applies to w. Therefore $L^{\mathsf{hard}}(x_w) = M(x_{w+1}) \neq M(x_w)$, and the refuter can output x_w. □

3.3.2 Linear Sum Circuits and Hardness Amplification with Them

We need an XOR lemma with "linear sum" decoders: given a corrupted codeword $\tilde{f}$ that is $(1/2 - \varepsilon)$-close to $\mathsf{Amp}(f)$, there is an affine transformation A such that $A(\tilde{f})$ is δ-close to f.

The actual definition of *linear sum circuits* is more involved for the following reason. Our XOR lemma works in the *list-decoding* regime, therefore it also receives an advice string α (i.e., the index in the list) and outputs the α-th decoded message in the list. When α is fixed, $A(\tilde{f}; \alpha)$ is simply an affine function over $\tilde{f}$; but the dependence on α can be more complicated. It turns out that we need an upper bound on the *locality* of the dependence on α, defined as follows.

Definition 3.3.3 (Linear Sum Circuits)**.** Let $x \in \{0,1\}^n$ and $\alpha \in \{0,1\}^a$ be two inputs. A *linear sum* circuit on input x with advice α is a function $C : \{0,1\}^n \times \{0,1\}^a \to \mathbb{R}^m$ of the

following form:

$$C(x,\alpha)_i = \sum_{k\in[A]} \mathsf{coeff}_k(\alpha)\cdot x_{\mathsf{idx}_k(\alpha,i)}.$$

Here, A is the *fan-in* of C. The circuit is described by two functions $\mathsf{coeff}_k(\alpha)$ and $\mathsf{idx}_k(\alpha,i)$; note that $\mathsf{coeff}_k(\alpha)$ does not depend on i. For technical convenience, we will also allow $\mathsf{idx}_k(\alpha,i)$ to take special values ZERO and ONE, where x_{ZERO} is always 0 and x_{ONE} is always 1.

Besides the fan-in A, the following complexity measures of C will also be important:

- We say the *coefficient sum* of C is at most U, if for every advice α, we have

$$\sum_{k\in[A]} |\mathsf{coeff}_k(\alpha)| \le U.$$

- We say that C has *locality* l, if for every fixed k, there is a subset S_k of l bits of α such that the functions $\mathsf{coeff}_k(\alpha)$ and $\mathsf{idx}_k(\alpha,i)$ only depends on $\alpha|_{S_k}$.

Example 3.3.4. Consider the following example (simplified from the proof of Theorem 3.3.5). Suppose the advice α consists of a list of strings $(\alpha_1,\alpha_2,\ldots,\alpha_{a'})$ where $a'\approx 1/\varepsilon^2$; given an index k, $\mathsf{coeff}_k(\alpha)$ only depends in α_k, and $\mathsf{idx}_k(\alpha,i)$ only depends on α_k and i. Suppose each α_k has length l, then regardless of the number a', the linear sum has locality l.

We need the following XOR lemma with linear sum decoders. The XOR lemma was proved in [Lev87, GNW11], and it was shown in [CLW20, Section A] to admit linear sum decoders. For completeness, we provide a proof below and verify the locality of the linear sum. Note that the XOR lemma is stated below as an approximately locally list-decodable code.

Theorem 3.3.5. *Let $N\in\mathbb{N}$, $0<\varepsilon,\delta<1/10$, $k:=O(\log(1/\varepsilon)/\delta)$, $\tilde{N}:=N^k$, and $a:=O(\log^2 N/(\varepsilon\delta)^2)$. There is an algorithm $\mathsf{Amp}:\{0,1\}^N\to\{0,1\}^{\tilde{N}}$ computable in deterministic $\mathrm{poly}(\tilde{N})$ time, and a linear sum circuit $C:\{0,1\}^{\tilde{N}}\times\{0,1\}^a\to\mathbb{R}^N$ such that the following hold.*

(List-decoding) *For every string $\tilde{f}\in\{0,1\}^{\tilde{N}}$ that is $(1/2-\varepsilon)$-close to $\mathsf{Amp}(f)$ for some hidden string f, there is an advice $\alpha\in\{0,1\}^a$, such that (1) for every $i\in[N]$, $C(\tilde{f},\alpha)_i\in[0,1]$; and (2) $\|C(\tilde{f},\alpha)-f\|_1\le\delta$.*

(Complexity) *The fan-in, coefficient sum, and locality of C are at most $O(\log N/(\varepsilon\delta)^2)$, $O(1/\varepsilon)$, and $\log\tilde{N}$ respectively.*

Proof. For simplicity, we identify a string f of length N and a Boolean function $f:[N]\to\{0,1\}$, where $f(x)$ outputs the x-th bit of f. For a string $f\in\{0,1\}^N$, denote $f^{\oplus k}$ to be the following string of length N^k. For each $(x_1,x_2,\ldots,x_k)\in[N]^k$, we have

$$f^{\oplus k}(x_1,x_2,\ldots,x_k) := \bigoplus_{i=1}^{k} f(x_i) = f(x_1)\oplus f(x_2)\oplus\cdots\oplus f(x_k),$$

where $\oplus$ denotes bitwise XOR. We simply let $\mathsf{Amp}(f):=f^{\oplus k}$.

The decoder. For a length-k vector $\vec{v}^{\perp} \in ([N] \cup \{\perp\})^k$ and $i \in [N]$, let $\vec{v}^{i}$ denote the vector where each $\perp$ in $\vec{v}^{\perp}$ is replaced by i. (In the decoder, we will only need the case where each $\vec{v}^{\perp}$ contains exactly one $\perp$, so $\vec{v}^{i}$ simply replaces that single $\perp$ by i.)

Let $\tilde{f} : [N]^k \to \{0,1\}$ be a codeword (treated as a Boolean function). We set $A' := O(\log N/(\varepsilon\delta)^2)$ and $r := \frac{(2+\delta)\varepsilon}{1-\delta}$. Our decoder will take a list of vectors $\vec{v}_1^{\perp}, \vec{v}_2^{\perp}, \ldots, \vec{v}_{A'}^{\perp} \in ([N] \cup \{\perp\})^k$ and a list of signs $\sigma_1, \sigma_2, \ldots, \sigma_{A'} \in \{0,1\}$ as advice. Intuitively, $\vec{v}_i^{\perp}$ denotes a segment of $\tilde{f}$ that has noticeable correlation with f, and σ_i denotes whether the correlation is positive or negative; our linear sum decoder uses the average of $\tilde{f}(\vec{v}_j^{i}) \oplus \sigma_j$ as a prediction of f_i.

More formally, given an input $i \in [N]$, the decoder outputs [13]

$$\mathsf{dec}(\tilde{f})_i := \frac{1}{r} \mathop{\mathbb{E}}_{j \leftarrow [A']} \Big[(\tilde{f}(\vec{v}_j^{i}) \oplus \sigma_j) - 1/2 \Big] + 1/2. \tag{3.2}$$

Correctness. We establish the correctness of this decoder by the following lemma.

Lemma 3.3.6. *Let $k \geq 1$, $\delta \in (0, 1/10)$, $\varepsilon := (1-\delta)^{k-1}(1/2-\delta)$, and $A' := O(\log N/(\varepsilon\delta)^2)$. For every string $\tilde{f} \in \{0,1\}^{N^k}$ that is $(1/2-\varepsilon)$-close to $f^{\oplus k}$ for some hidden string $f \in \{0,1\}^N$, there is a list of A' vectors $\vec{v}_1^{\perp}, \vec{v}_2^{\perp}, \ldots, \vec{v}_{A'}^{\perp} \subseteq ([N] \cup \{\perp\})^k$, and a list of signs $\sigma_1, \sigma_2, \ldots, \sigma_{A'} \in \{0,1\}$, such that (1) for every $i \in [N]$, $\mathsf{dec}(\tilde{f})_i \in [0,1]$; and (2) $\|\mathsf{dec}(\tilde{f}) - f\|_1 \leq \delta$.*

Proof. We use induction on k. Suppose $k = 1$, then one can verify by direct calculation that the lemma holds by setting $\vec{v}_1^{\perp} = (\perp)$ and $\sigma_1 = 0$. Now suppose $k > 1$ and the lemma holds for $k-1$.

Fix $i \in [N]$ and let $\vec{v}^{\perp} \in ([N] \cup \{\perp\})^k$ denote some vector whose first coordinate is $\perp$ and other coordinates are from $[N]$. Think of every coordinate of $\vec{v}^{\perp}$, except the first, is drawn independently and uniformly from $[N]$. Define

$$p_i := \Pr_{\vec{v}^{\perp} \leftarrow \{\perp\} \times [N]^{k-1}} \Big[\tilde{f}(\vec{v}^{i}) = f^{\oplus k}(\vec{v}^{i}) \Big].$$

Case I: Suppose there is some $i_0 \in [N]$ such that $|p_{i_0} - 1/2| > \varepsilon/(1-\delta)$. Let $b \in \{0,1\}$ be a bit, consider the sub-string $\tilde{f}' \in \{0,1\}^{N^{k-1}}$ such that $\tilde{f}'(\vec{v}^{\perp}) = \tilde{f}(\vec{v}^{i_0}) \oplus b$. Then, for some $b \in \{0,1\}$, $\tilde{f}'$ is $(1/2 - \varepsilon/(1-\delta))$-close to $f^{\oplus(k-1)}$.

By the induction hypothesis, there is a list of A' vectors $\vec{u}_1^{\perp}, \ldots, \vec{u}_{A'}^{\perp} \subseteq ([N-1] \cup \{\perp\}^k)$ and a list of signs $\sigma'_1, \ldots, \sigma'_{A'} \in \{0,1\}$ such that the vector dec' satisfies the conclusion of the lemma, where

$$\mathsf{dec}'_i := \frac{1}{r} \mathop{\mathbb{E}}_{j \leftarrow [A']} \Big[(\tilde{f}'(\vec{u}_j^{i}) \oplus \sigma'_j) - 1/2 \Big] + 1/2.$$

For each j, let $\vec{v}_j^{\perp}$ be the concatenation of i_0 and $\vec{u}_j^{\perp}$, and let $\sigma_j = \sigma'_j \oplus b$. We have that $\mathsf{dec}(\tilde{f})_i$ is exactly dec'_i and we are done.

[13] (3.2) is perhaps easier to understand when we change the basis from $\{0,1\}$ to $\{1,-1\}$; we choose the basis $\{0,1\}$ only to be consistent with other parts of this chapter. When we change the basis to $\{1,-1\}$, XOR becomes multiplication and the assertion "$a = b$" becomes simply $a \cdot b$. Thus (3.2) becomes

$$\mathsf{dec}(\tilde{f})_i = \frac{1}{r} \mathop{\mathbb{E}}_{j \leftarrow [A']} \Big[\tilde{f}(\vec{v}_j^{i}) \cdot \sigma_j \Big],$$

which is simply the average of all A' predictions, amplified by a factor of $1/r$.

Case II: Suppose for every $i \in [N]$, we have $|p_i - 1/2| \le \varepsilon/(1-\delta)$. Note that, since $\tilde{f}$ is $(1/2-\varepsilon)$-close to $f^{\oplus k}$, we have

$$\mathop{\mathbb{E}}_{i \leftarrow [N]}[p_i] \ge 1/2 + \varepsilon.$$

We sample each $\vec{v}_j^{\perp} \leftarrow \{\perp\} \times [N]^{k-1}$ independently at random. Let

$$\tilde{p}_i := \Pr_{j \leftarrow [A']}\Big[\tilde{f}(\vec{v}_j^{i}) = f^{\oplus k}(\vec{v}_j^{i})\Big].$$

Let $\eta := \frac{\varepsilon\delta}{2(1-\delta)}$. By a Chernoff bound, w.p. $1 - Ne^{-2\eta^2 t} > 0$, for every $i \in [N]$, we have $|p_i - \tilde{p}_i| \le \eta$. Let $\sigma_j := f^{\oplus(k-1)}((\vec{v}_j^{\perp})_{2\sim k}) = \bigoplus_{l=2}^{k} f((\vec{v}_j^{\perp})_l)$, then we have

$$\tilde{p}_i = \Pr_{j \leftarrow [t]}\Big[f_i = \tilde{f}(\vec{v}_j^{i}) \oplus \sigma_j\Big].$$

Note that

$$\begin{aligned}\mathsf{dec}(\tilde{f})_i &= \frac{1}{r}(\tilde{p}_i \cdot (f_i - 1/2) + (1-\tilde{p}_i)\cdot(1/2 - f_i)) + 1/2\\ &= \frac{(f_i - 1/2)(2\tilde{p}_i - 1)}{r} + 1/2.\end{aligned}$$

We first show that for every $i \in [N]$, $\mathsf{dec}(\tilde{f})_i \in [0,1]$. In fact,

$$\begin{aligned}|\mathsf{dec}(\tilde{f})_i - 1/2| &= \frac{1}{r}|(f_i - 1/2)(2\tilde{p}_i - 1)|\\ &= \frac{1}{r}|\tilde{p}_i - 1/2|\\ &\le \frac{1-\delta}{(2+\delta)\varepsilon}\left(\frac{\varepsilon}{1-\delta} + \eta\right)\\ &= 1/2.\end{aligned}$$

Then we show that

$$\|\mathsf{dec}(\tilde{f}) - f\|_1 = \mathop{\mathbb{E}}_{i \leftarrow [N]}[|\mathsf{dec}(\tilde{f})_i - f_i|] \le \delta.$$

This is because

$$\begin{aligned}&\mathop{\mathbb{E}}_{i \leftarrow [N]}[|\mathsf{dec}(\tilde{f})_i - f_i|]\\ =& \mathop{\mathbb{E}}_{i \leftarrow [N]}\left[\left|\frac{(f_i - 1/2)(2\tilde{p}_i - 1)}{r} + 1/2 - f_i\right|\right]\\ =& \mathop{\mathbb{E}}_{i \leftarrow [N]}\left[\left|\frac{\tilde{p}_i - 1/2}{r} - 1/2\right|\right]\\ =& 1/2 - \frac{\mathbb{E}_{i \leftarrow [N]}[\tilde{p}_i] - 1/2}{r} \qquad (3.3)\\ \le& 1/2 - \frac{\varepsilon - \eta}{r} \le \delta,\end{aligned}$$

where (3.3) is because $\frac{1}{r}|\tilde{p}_i - 1/2| \le 1/2$ for every i. The lemma follows. ◇

Complexity. It remains to determine the complexity of the decoder defined in (3.2). The advice string α contains the vectors $\vec{v}_1^{\perp}, \vec{v}_2^{\perp}, \ldots, \vec{v}_{A'}^{\perp}$ and the signs $\sigma_1, \sigma_2, \ldots, \sigma_{A'}$. It is clear that the fan-in is at most $A' + 1 = O(\log N/(\varepsilon\delta)^2)$. The coefficient sum is $O(1/r) = O(\varepsilon^{-1})$. Since the k-th term is

$$\mathsf{coeff}_k(\alpha) = (-1)^{\sigma_j}/(A'r), \quad \text{and} \quad \mathsf{idx}_k(\alpha, i) = \vec{v}_k^i,$$

it follows that each term only depends on $\log \tilde{N}$ bits of α. □

We will also use the notation $\mathsf{dec}_\alpha(f)$ to denote $C(f, \alpha)$, emphasising that dec_α is an affine transformation that depends on α.

3.3.3 A Stretch Reduction for Remote-Point and Partial-AvgHard

In our framework for solving Remote-Point (Section 3.5), for technical convenience, we only consider circuits $C : \{0,1\}^n \to \{0,1\}^{\ell(n)}$, where $\ell(n)$ is a certain stretch function. (For example, it might be the case that $\ell(n)$ is rounded to a power of 2 for every n.) In this section, we show that such an algorithm can also solve Remote-Point for circuits of larger stretches (such as $\frac{3}{2}\ell(n)$). This justifies that it is without loss of generality to only consider stretch functions that are *equal* to $\ell(n)$.

Lemma 3.3.7 (Stretch Reduction for Remote-Point)**.** *Let $\mathscr{C}$ be a typical circuit class and s be a size parameter. Suppose that $\mathscr{C}[s]$-Remote-Point with stretch $\ell'(n)$ and distance parameter $1/2 - \varepsilon'(n)$ admits an $\mathsf{FP}^{\mathsf{NP}}$ algorithm. Then for any stretch $\ell = \ell(n) \geq \ell'(n+1)/2$, $\mathscr{C}[s]$-Remote-Point with stretch $\ell(n)$ and distance parameter $1/2 - \varepsilon(n)$ also admits an $\mathsf{FP}^{\mathsf{NP}}$ algorithm, where $\varepsilon(n) := 2 \cdot \varepsilon'(n+1)$.*

Proof. Denote $\ell' := \ell'(n+1)$, $\varepsilon' := \varepsilon'(n+1)$, $\ell := \ell(n)$, and $\varepsilon := \varepsilon(n)$, and let $C : \{0,1\}^n \to \{0,1\}^\ell$ be an input circuit. If ℓ is a multiple of ℓ', we can split the ℓ-bit output of C into blocks of size ℓ' and add a dummy input bit to construct $m := \ell/\ell'$ circuits $C_1, C_2, \ldots, C_m : \{0,1\}^{n+1} \to \{0,1\}^{\ell'(n+1)}$ such that for every $x \in \{0,1\}^n$ and $b \in \{0,1\}$,

$$C(x) = C_1(x,b) \circ C_2(x,b) \circ \cdots \circ C_m(x,b).$$

Using the $\mathsf{FP}^{\mathsf{NP}}$ algorithm for $\mathscr{C}$-Remote-Point with stretch $\ell'(n+1)$ and error parameter $\varepsilon'(n+1)$, we can construct $y_1, y_2, \ldots, y_m \in \{0,1\}^{\ell'}$ such that each y_i is $(1/2 - \varepsilon')$-far from $\mathrm{Range}(C_i)$. It then follows that the concatenation $y_1 \circ y_2 \circ \cdots \circ y_m$ is $(1/2 - \varepsilon')$-far from $\mathrm{Range}(C)$.

We now consider the case where ℓ is not a multiple of ℓ'. Let $I : \{0,1\}^{n+1} \to \{0,1\}$ be defined as the projection $I(x) = x_{n+1}$, that is, it always outputs the last bit. For any t, let $I^{\otimes t} : \{0,1\}^{n+1} \to \{0,1\}^t$ denote the concatenation of t copies of I. Therefore, $\mathrm{Range}(I^{\otimes t}) = \{0^t, 1^t\}$. Since $\mathscr{C}$ is typical, we have $I^{\otimes t} \in \mathscr{C}$.

Let $M = k \cdot \ell'$ be the smallest multiple of ℓ' larger than ℓ, and $\bar{\ell} := M - \ell$. For a multi-output $\mathscr{C}$ circuit C, we define $\tilde{C} : \{0,1\}^{n+1} \to \{0,1\}^M$ as

$$\tilde{C}(x,b) = C(x) \circ I^{\otimes \bar{\ell}}(x,b),$$

where $x \in \{0,1\}^n$ and $b \in \{0,1\}$. Since $\tilde{C}$ is of input length $n+1$ and output length being a multiple of ℓ', we can get a remote point $s \in \{0,1\}^M$ in $\mathsf{FP}^{\mathsf{NP}}$ that is $(1/2-\varepsilon')$-far from $\mathrm{Range}(\tilde{C})$.

Let $s = s_1 \circ s_2$, where s_1 and s_2 has length ℓ and $\bar{\ell}$, respectively. We then prove that s_1 is $(1/2-\varepsilon)$ far from $\mathrm{Range}(C)$. Towards a contradiction, we assume that s_1 is not $(1/2-\varepsilon)$-far from $\mathrm{Range}(C)$. In other words, there is an $x \in \{0,1\}^n$ such that $\delta(C(x), s_1) < 1/2-\varepsilon$. By considering the Hamming weight of s_2 we know that there is a $b \in \{0,1\}$ such that $\delta(I^{\otimes\bar{\ell}}(x,b), s_2) \leq 1/2$. It then follows that

$$\begin{aligned}
\delta(s, \tilde{C}(x,b)) &= \delta(s_1 \circ s_2, C(x) \circ I^{\otimes\bar{\ell}}(x,b))) \\
&\leq \frac{\ell}{\ell+\bar{\ell}} \cdot \left(\frac{1}{2} - \varepsilon(n)\right) + \frac{\bar{\ell}}{\ell+\bar{\ell}} \cdot \frac{1}{2} \\
&\leq \frac{1}{2} - \frac{\ell}{\ell+\bar{\ell}} \cdot \varepsilon(n) \\
&< \frac{1}{2} - \varepsilon'(n+1).
\end{aligned}$$

This leads to a contradiction as s is $(1/2-\varepsilon'(n+1))$-far from $\mathrm{Range}(\tilde{C})$. □

Similar to Remote-Point, another average-case problem Partial-AvgHard can also be reduced to the instances with smaller stretch in the same way.

Lemma 3.3.8 (Stretch Reduction for Partial-AvgHard)**.** *Let $\mathscr{C}$ be a typical circuit class and s be a size parameter. Suppose that $\mathsf{NC}^0_2 \circ (\mathscr{C}[s])$-Partial-AvgHard with stretch $\ell'(n)$ and distance parameter $1/2-\varepsilon'(n)$ admits an $\mathsf{FP}^{\mathsf{NP}}$ algorithm, then for any stretch $\ell = \ell(n) \geq \ell'(n+1)/2$, $\mathscr{C}[s]$-Partial-AvgHard with stretch $\ell(n)$ and distance parameter $1/2-\varepsilon(n)$ admits an $\mathsf{FP}^{\mathsf{NP}}$ algorithm, where $\varepsilon(n) := 2 \cdot \varepsilon'(n+1)$.*

Proof Sketch. The proof of this lemma is similar to that of Lemma 3.3.7. Here we use the same notation as the proof of Lemma 3.3.7.

Let $X = \{x_1, \ldots, x_\ell\}$ denote input strings, and let $y_i := x_i \circ 0$. We create $\bar{\ell}$ copies of $0^n \circ 1$ and use $y_{\ell+1}, \ldots, y_M$ to denote these copies.

we solve $\mathsf{NC}^0_2 \circ \mathscr{C}$-Partial-AvgHard on $\{y_1, \ldots, y_M\}$ and get an average-case hard partial truth table $s = s_1 \circ s_2$ that is $(1/2-\varepsilon'(n+1))$-far from any truth table of $\mathsf{NC}^0_2 \circ \mathscr{C}$ circuit, where s_1 and s_2 has length ℓ and $\bar{\ell}$. Then we prove s_1 is a solution to the original problem. For some $\mathscr{C}$ circuit C, if s_1 is not $(1/2-\varepsilon(n))$ far from partial truth table of C on X, we can define $\tilde{C}_1, \tilde{C}_2 : \{0,1\}^n \times \{0,1\} \to \{0,1\}$ as $\tilde{C}_1(x;b) := C(x) \vee b, \tilde{C}_2(x;b) := C(x) \wedge (\neg b)$. Then one of $\tilde{C}_1$ and $\tilde{C}_2$ has partial truth table on $Y := \{y_1, \ldots, y_M\}$ not $(1/2-\varepsilon'(n+1))$ far from s, which leads to a contradiction. Therefore, s_1 has to be a solution. The analysis is similar to Lemma 3.3.7. □

3.3.4 Satisfying Pairs for $\mathsf{NC}^0_d \circ \mathscr{C}$

We show that satisfying pairs for $\mathsf{NC}^0_d \circ \mathscr{C}$ circuits can be reduced to the satisfying pairs of $\mathsf{AND}^0_d \circ \mathscr{C}$, $\mathsf{XOR}^0_d \circ \mathscr{C}$, or $\mathsf{OR}^0_d \circ \mathscr{C}$ via standard Fourier analysis (see, e.g., [CW19b, Section 4]). This will be beneficial for the unconditional results for weak circuit classes.

Theorem 3.3.9. *For every constants $\delta \in [0,1]$ and $d \geq 1$, there is a constant δ' such that the following holds. Let $N = N(n), M = M(n), n, s = s(n)$ be parameters, $\mathsf{C}_d \in \{\mathsf{AND}_d, \mathsf{OR}_d, \mathsf{XOR}_d\}$.*

Then $\#(\mathsf{NC}^0_d \circ \mathscr{C})$-Satisfying-Pairs (resp. Approx_δ-$(\mathsf{NC}^0_d \circ \mathscr{C})$-Satisfying-Pairs) with parameters (N, M, n, s) is $\tilde{O}(n)$-time Turing-reducible to $\#(\mathsf{C}_d \circ \mathscr{C})$-Satisfying-Pairs (resp. $\mathsf{Approx}_{\delta'}$-$(\mathsf{C}_d \circ \mathscr{C})$-Satisfying-Pairs) with parameters $(\Theta(N), M, n, s)$, as long as each input circuit of the $\#(\mathsf{NC}^0_d \circ \mathscr{C})$-Satisfying-Pairs (resp. Approx_δ-$(\mathsf{NC}^0_d \circ \mathscr{C})$-Satisfying-Pairs) problem are given explicitly as a top NC^0_d circuit C_{top} together with d circuits $C_1, C_2, \ldots, C_d \in \mathscr{C}$ feeding C_{top}.

Moreover, the oracle algorithm for $\#(\mathsf{NC}^0_d \circ \mathscr{C})$-Satisfying-Pairs (resp. Approx_δ-$(\mathsf{NC}^0_d \circ \mathscr{C})$-Satisfying-Pairs) only makes $O(1)$ non-adaptive queries to the $\#(\mathsf{C}_d \circ \mathscr{C})$-Satisfying-Pairs (resp. $\mathsf{Approx}_{\delta'}$-$(\mathsf{C}_d \circ \mathscr{C})$-Satisfying-Pairs) oracle.

Proof. Let N, M, n, s be the parameters. Suppose that we are given $C_1, C_2, \ldots, C_N \in \mathscr{C}[s]$ and $x_1, x_2, \ldots, x_M \in \{0,1\}^n$ as input. We assume that $C_1, C_2, \ldots, C_N$ share the same upper NC^0_d function computing $f : \{0,1\}^d \to \{0,1\}$, that is for every $i \in [N]$, $C_i \equiv f \circ D_i$ for some d-output $\mathscr{C}$ circuit D_i of size at most s. This is without loss of generality since there are at most $2^{2^d} = O(1)$ different NC^0_d functions and we can (approximately) count the number of satisfying pairs for each of these cases separately.

We first consider the case for $\mathsf{C}_d = \mathsf{AND}_d$. We use the basis $\{0,1\} \subseteq \mathbb{R}$ for Boolean values and write f as

$$f(x) = \sum_{S \subseteq [d]} \alpha_S \cdot \prod_{i \in S} x_i,$$

where each coefficient $\alpha_S \in [-2^d, 2^d] \cap \mathbb{Z}$. Note that we can compute the coefficients by writing the truth table of f in the canonical disjunctive normal form, represent x by x, $\neg x$ by $1 - x$, $\wedge$ by multiplication, and (disjoint) $\vee$ by addition, and then expanding the multi-linear polynomial using a brute-force algorithm in $O(1)$ time.

Let $\chi_S(x) := \prod_{i \in S} x_i$ for $S \subseteq [d]$. Then the number of $(i,j) \in [N] \times [M]$ such that $C_i(x_j) = 1$ is

$$\begin{aligned}
&\sum_{i \in [N]} \sum_{j \in [M]} f(D_i(x_j)) \\
&= \sum_{i \in [N]} \sum_{j \in [M]} \sum_{S \subseteq [d]} \alpha_S \cdot \chi_S(D_i(x_j)) \\
&= \sum_{S \subseteq [d]} \alpha_S \cdot \left[\sum_{i \in [N]} \sum_{j \in [M]} \chi_S(D_i(x_j)) \right] \\
&= \sum_{S \subseteq [d]} \alpha_S \cdot \left[\sum_{i \in [N]} \sum_{j \in [M]} \mathsf{AND}_{|S|} \circ D_i|_S(x_j) \right],
\end{aligned}$$

where $D_i|_S : \{0,1\}^{|S|} \to \{0,1\}$ representing the circuit obtained from D_i by restricting to the output bits in S. Then our algorithm is as follows: We enumerate all $S \subseteq [d]$ and count (resp. approximately count) the number A_S of satisfying pairs for circuits $\mathsf{AND}_{|S|} \circ D_1|_S, \ldots, \mathsf{AND}_{|S|} \circ D_N|_S$ and inputs $x_1, \ldots, x_M$, then we output the answer $\sum_{S \subseteq [d]} \alpha_S \cdot A_S$.

For $\mathsf{C}_d = \mathsf{XOR}_d$ and $\mathsf{C}_d = \mathsf{OR}_d$, we only need to write f as

$$f(x) = \sum_{S \subseteq [d]} \alpha'_S \cdot \bigoplus_{i \in S} x_i, \tag{3.4}$$

$$f(x) = \sum_{S \subseteq [d]} \alpha''_S \cdot \bigvee_{i \in S} x_i, \tag{3.5}$$

where $\alpha'_S, \alpha''_S \leq 2^{O(d)}$. Note that (3.4) can be obtained using the basis $\{\mathsf{true} := -1, \mathsf{false} := 1\}$ and (3.5) can be obtained using the basis $\{\mathsf{true} := 0, \mathsf{false} := 1\}$. □

3.4 Range Avoidance

Definition 3.4.1 (Algorithms for Satisfying-Pairs with P^{NP} Preprocessing on Circuits)**.** Let P be one of $\mathscr{C}$-Satisfying-Pairs, $\#\mathscr{C}$-Satisfying-Pairs, Approx_δ-$\mathscr{C}$-Satisfying-Pairs, and Gap_δ-$\mathscr{C}$-Satisfying-Pairs. A t-time algorithm for P with P^{NP} preprocessing of an ℓ-size data structure on circuits is a pair of algorithms (A_1, A_2) that solves P in two phases:

1. Given the circuits $C_1, C_2, \ldots, C_N : \{0,1\}^n \to \{0,1\}$ of size s, the polynomial-time algorithm A_1 with oracle access to a SAT oracle computes a string $\mathsf{DS} \in \{0,1\}^\ell$.
2. Given the inputs $x_1, x_2, \ldots, x_M \in \{0,1\}^n$ and the string $\mathsf{DS} \in \{0,1\}^\ell$, the algorithm A_2 solves P on the instance $(C_1, \ldots, C_N, x_1, \ldots, x_M)$ in time t.

In this subsection, we establish the connection between the Avoid and Satisfying-Pairs. The main result is the following theorem.

Theorem 3.4.2. *There are constants $\varepsilon > 0$ and $c_0 \geq 1$ such that the following holds. Let $0 < \eta < 1/2$ be a constant, $\ell(n) > n^{1+4\eta}$ be a good function. Let $\mathscr{C}[s]$ be a typical circuit class where $s = s(n)$ is a size parameter, and $\mathscr{C}'[2s] := \mathsf{OR}_2 \circ \mathscr{C}[s]$ (i.e. a $\mathscr{C}'$ circuit of size $2s$ refers to the OR of at most two $\mathscr{C}$ circuits of size s).*

Assumption: *Suppose that for some constant $c \geq 1$, there is an $(NM/\log^{c_0}(NM))$-time algorithm for $\mathsf{Approx}_\varepsilon$-$\mathscr{C}'$-Satisfying-Pairs with $N := \ell^{1-\eta} \cdot \mathrm{polylog}(\ell)$ circuits of size $2s(n)$ and $M := \ell^{c+1-\eta} \cdot \mathrm{polylog}(\ell)$ inputs of length $n \cdot \mathrm{polylog}(\ell)$, allowing a P^{NP} preprocessing of an N^c-size data structure on circuits.*

Conclusion: *Then there is an $\mathsf{FP}^{\mathsf{NP}}$ algorithm for $\mathscr{C}[s]$-Avoid with stretch $\ell(n)$.*

3.4.1 Proof of Theorem 3.4.2

Proof. Suppose that we are given a $\mathscr{C}$ circuit $C : \{0,1\}^n \to \{0,1\}^\ell$. Without loss of generality, we may assume ℓ is a power of 2 and $c \geq 2$. We set the following parameters:

$$\begin{aligned} m &:= 5(c+2)/\eta = O(1), \\ w_{\mathsf{proof}} &:= \log \ell, & W_{\mathsf{proof}} &:= 2^{w_{\mathsf{proof}}} = \ell, \\ h_{\mathsf{proof}} &:= (c+1)\log \ell, & H_{\mathsf{proof}} &:= 2^{h_{\mathsf{proof}}} = \ell^{c+1}, \\ n_{\mathsf{hard}} &:= 10 H_{\mathsf{proof}} \cdot n \cdot \mathrm{polylog}(\ell), \end{aligned}$$

$$T := H_{\mathsf{proof}} \cdot W_{\mathsf{proof}} / \log^{c_{\mathsf{tm}}}(\ell).$$

The constants ε, c_0, and c_{tm} will be determined later.

Let L^{hard} be the hard language constructed in Theorem 3.3.1. We use n_{hard} and T to denote the input length and the time complexity of L^{hard}, respectively, i.e.

$$L^{\mathsf{hard}} \in \mathsf{NTIME}_{\mathsf{TM}}[T] \setminus \text{i.o.-}\mathsf{NTIMEGUESS}_{\mathsf{RAM}}[T/\log^{c_{\mathsf{hard}}}(T), n_{\mathsf{hard}}/10]/_{(n_{\mathsf{hard}}/10)},$$

where c_{hard} is some large universal constant. Note that since $T = \ell^{c+2}/\mathrm{polylog}(\ell)$, $n_{\mathsf{hard}} = O(\ell^{c+1} \cdot n)$, $\ell > n^{1+4\eta}$, we can see that $n_{\mathsf{hard}}^{1+\Omega(1)} \le T \le n_{\mathsf{hard}}^2$, which satisfies the technical condition of Theorem 3.3.1.

We describe a nondeterministic RAM M^{PCPP} that runs in $T/\log^{c_{\mathsf{hard}}}(T)$ time, uses $n_{\mathsf{hard}}/10$ advice bits, guesses $n_{\mathsf{hard}}/10$ nondeterministic bits, and attempts to solve L^{hard} on n_{hard}-bit inputs. By the definition of L^{hard}, M^{PCPP} has to fail on some input $x \in \{0,1\}^{n_{\mathsf{hard}}}$ when n_{hard} is sufficiently large. Our goal is to design such an algorithm M^{PCPP} that (1) rejects every $x \notin L^{\mathsf{hard}}$, and (2) accepts every $x \in L^{\mathsf{hard}}$ *with an easy witness*. Thus, if M^{PCPP} fails on some input x, then $x \in L^{\mathsf{hard}}$ and it has only "hard witnesses", which will be exploited for finding a non-output of C.

Here, to define the inputs x "with an easy witness", we will need the 2-query rectangular PCPP in Theorem 2.5.10 for the following language

$$L^{\mathsf{enc}} := \{\mathsf{Encode}(x) : x \in L^{\mathsf{hard}}\},$$

where we fix an error-correcting code $(\mathsf{Encode}, \mathsf{Decode})$ as in Theorem 2.4.1. Let δ_{code} be the distance of the code. Suppose a string of length n_{hard} is encoded (via Encode) into a string of length $\tilde{n}_{\mathsf{hard}} := O(n_{\mathsf{hard}})$. We set the following parameters:

$$h_{\mathsf{input}} := \left(1 - \frac{\Theta(\log\log T)}{\log T}\right) h_{\mathsf{proof}}, \qquad H_{\mathsf{input}} := 2^{h_{\mathsf{input}}} = H_{\mathsf{proof}}/\mathrm{polylog}(\ell),$$
$$w_{\mathsf{input}} := \lceil \log \tilde{n}_{\mathsf{hard}} \rceil - h_{\mathsf{input}}, \qquad W_{\mathsf{input}} := 2^{w_{\mathsf{input}}} = n \cdot \mathrm{polylog}(\ell).$$

We assume without loss of generality that $\tilde{n}_{\mathsf{hard}} = H_{\mathsf{input}} \cdot W_{\mathsf{input}}$. (This can always be done by adding at most $W_{\mathsf{input}} \ll \tilde{n}_{\mathsf{hard}}$ dummy bits into the codeword of the error-correcting code, where the resulting code is still of constant rate and distance.)

We apply Theorem 2.5.10 to obtain a 2-query rectangular PCPP for L^{enc} with an $H_{\mathsf{input}} \times W_{\mathsf{input}}$ input matrix and an $\hat{H}_{\mathsf{proof}} \times W_{\mathsf{proof}}$ proof matrix, where

$$\hat{h}_{\mathsf{proof}} := \log T(n) + \Theta(m \log\log T(n)) - w_{\mathsf{proof}} = (c+1)\log \ell + \Theta(m \log\log \ell),$$
$$\hat{H}_{\mathsf{proof}} := 2^{\hat{h}_{\mathsf{proof}}} = \ell^{c+1} \cdot \mathrm{polylog}(\ell).$$

We can check that the technical conditions of Theorem 2.5.10 for the 2-query rectangular PCPP construction holds:

- $w_{\mathsf{proof}} \ge (5/m)\log T$: because $\frac{m \cdot w_{\mathsf{proof}}}{5 \log T} \ge 1/\eta \ge 1$.
- $\hat{h}_{\mathsf{proof}} \ge (5/m)\log T$: because $h_{\mathsf{proof}} > w_{\mathsf{proof}}$.

- $\frac{h_{\mathsf{input}}}{\hat{h}_{\mathsf{proof}}} \le 1 - \frac{Cm \log\log T}{\log T}$: because $\frac{h_{\mathsf{input}}}{h_{\mathsf{proof}}} = 1 - \frac{\Theta(\log\log T)}{\log T}$ and $h_{\mathsf{proof}} \le \hat{h}_{\mathsf{proof}}$.
- $\frac{w_{\mathsf{input}}}{w_{\mathsf{proof}}} \le 1 - \frac{Cm \log\log T}{\log T}$: Note that $h_{\mathsf{input}} = h_{\mathsf{proof}} - \frac{\Theta(\log\log T) h_{\mathsf{proof}}}{\log T} \ge (c+1)\log \ell - O(\log\log \ell)$. We have $\frac{w_{\mathsf{input}}}{w_{\mathsf{proof}}} = \frac{\lceil \log \tilde{n}_{\mathsf{hard}} \rceil - h_{\mathsf{input}}}{\log \ell} \le \frac{(c+1)\ell + \log n + O(\log\log \ell) - h_{\mathsf{input}}}{\log \ell} \le \frac{\log n + O(\log\log \ell)}{\log \ell} \le 1 - \Omega(1)$.

By Theorem 2.5.10, there is a PCPP verifier VPCPP for L^{enc} with oracle access to $\Pi := \mathsf{Enc}(x) \circ \pi$, where the input $\mathsf{Enc}(x)$ is treated as a matrix of size $H_{\mathsf{input}} \times W_{\mathsf{input}}$, and the proof π is treated as a matrix of size $\hat{H}_{\mathsf{proof}} \times W_{\mathsf{proof}}$. The PCPP verifier has the following parameters:

- completeness error $= 1 - c_{\mathsf{pcp}}$,
- soundness error $= s_{\mathsf{pcp}}$,
- proximity parameter $= \delta_{\mathsf{code}}/3$,
- query complexity ≤ 2,
- parity-check bits ≤ 2,
- total randomness $= r := \log T + O(\log\log T + m \log m)$,
- row randomness $= r_{\mathsf{row}} := \hat{h}_{\mathsf{proof}} - (5/m)\log T = (c + 1 - \eta)\log \ell + O(\log\log \ell)$,
- column randomness $= r_{\mathsf{col}} := w_{\mathsf{proof}} - (5/m)\log T = (1 - \eta)\log \ell + O(\log\log \ell)$,
- shared randomness $= r_{\mathsf{shared}} := (10/m)\log T + O(\log\log T) = 2\eta \log \ell + O(\log\log \ell)$.

Moreover, the total number of parity-check bits and queries is at most 2, and the decision predicate $\mathsf{VDec} \leftarrow V_{\mathsf{dec}}(\mathsf{seed.shared})$, which takes the parity-check bits and the answers to the queries as the input, is an OR of its input bits or their negations.

For an input $x \in L^{\mathsf{hard}} \cap \{0,1\}^{n_{\mathsf{hard}}}$, we say that x *has an easy witness* if there is a proof matrix π for the statement "$\mathsf{Encode}(x) \in L^{\mathsf{enc}}$" such that:

(completeness) $\Pr_{\mathsf{seed} \leftarrow \{0,1\}^r}[\mathsf{VPCPP}^{\mathsf{Encode}(x) \circ \pi}(\mathsf{seed}) \text{ accepts}] \ge c_{\mathsf{pcp}}$; and

(easiness) for every row π_i of π, there exists a string w_i such that $\pi_i = C(w_i)$.

Description of M^{PCPP}. Now we define M^{PCPP}, which is a non-deterministic algorithm that runs in $T/\log^{c_{\mathsf{hard}}} T$ time and takes at most $\ell^{c+1} \le n_{\mathsf{hard}}/10$ bits of advice. The goal of M^{PCPP} is to reject every $x \notin L^{\mathsf{hard}}$ and accept every $x \in L^{\mathsf{hard}}$ with easy witness when appropriate advice is given.

On input $x \in \{0,1\}^{n_{\mathsf{hard}}}$, we guess $\hat{H}_{\mathsf{proof}}$ strings $w_1, w_2, \ldots, w_{\hat{H}_{\mathsf{proof}}} \in \{0,1\}^n$. Let π be the $\hat{H}_{\mathsf{proof}} \times W_{\mathsf{proof}}$ proof matrix where for each $i \in [\hat{H}_{\mathsf{proof}}]$, the i-th row of π is equal to $C(w_i)$. Let p_{acc} be the acceptance probability of the PCPP verifier VPCPP for L^{enc} given the input $\mathsf{Encode}(x)$ and the proof π, i.e.,

$$p_{\mathsf{acc}} := \Pr_{\mathsf{seed} \leftarrow \{0,1\}^r}[\mathsf{VPCPP}^{\mathsf{Encode}(x) \circ \pi}(\mathsf{seed}) \text{ accepts}].$$

We need to distinguish between the case that $p_{\mathsf{acc}} \ge c_{\mathsf{pcp}}$ and the case that $p_{\mathsf{acc}} \le s_{\mathsf{pcp}}$. We set $\varepsilon := (c_{\mathsf{pcp}} - s_{\mathsf{pcp}})/4$ so that this can be done by estimating p_{acc} with an additive error at most ε, which will be done by applying the $\mathsf{Approx}_\varepsilon$-$\mathscr{C}'$-Satisfying-Pairs algorithm in the assumption. (Recall that c_{pcp} and s_{pcp} are absolute constants that only depend on δ_{code}, which means that ε is also an absolute constant.)

In what follows, we reduce the problem of estimating p_{acc} to $2^{r_{\mathsf{shared}}}$ instances of $\mathsf{Approx}_{\varepsilon}$-$\mathscr{C}'$-Satisfying-Pairs, where each instance consists of $2^{r_{\mathsf{col}}} = \ell^{1-\eta} \cdot \mathrm{polylog}(\ell)$ circuits and $2^{r_{\mathsf{row}}} = \ell^{c+1-\eta} \cdot \mathrm{polylog}(\ell)$ inputs. Then we will use the algorithm for $\mathsf{Approx}_{\varepsilon}$-$\mathscr{C}'$-Satisfying-Pairs to estimate p_{acc}, where the data structure in the preprocessing phase will be treated as an advice of M^{PCPP}.

For the simplicity of presentation, we define the notation:

$$\begin{aligned}
(\mathsf{itype}[1], \ldots, \mathsf{itype}[q]) &\leftarrow V_{\mathsf{type}}(\mathsf{seed.shared}),\\
(\mathsf{irow}[1], \ldots, \mathsf{irow}[q]) &\leftarrow V_{\mathsf{row}}(\mathsf{seed.shared}, \mathsf{seed.row}),\\
(\mathsf{icol}[1], \ldots, \mathsf{icol}[q]) &\leftarrow V_{\mathsf{col}}(\mathsf{seed.shared}, \mathsf{seed.col}), \qquad \text{and}\\
(pc_1, \ldots, pc_p) &\leftarrow V_{\mathsf{pc}}(\mathsf{seed.shared}),
\end{aligned}$$

where $p = p(\mathsf{seed.shared})$, $q = q(\mathsf{seed.shared})$, $p + q \le 2$, and $pc_i : \{0,1\}^{r_{\mathsf{row}}+r_{\mathsf{col}}} \to \{0,1\}$ is an XOR of (some of) its input bits (i.e. a GF(2)-linear function) for every $i \in [p]$.

Reduction to Satisfying Pairs. The input strings in the $\mathsf{Approx}_{\varepsilon}$-$\mathscr{C}'$-Satisfying-Pairs instance will be of the form $(a_1, \ldots, a_q, pc_1^{\mathsf{row}}, \ldots, pc_p^{\mathsf{row}})$. For each $j \in [q]$, the meaning of a_j is as follows:

- if $\mathsf{itype}[j] = \mathsf{input}$, then a_j is interpreted as a row of the input matrix, and we use $(a_j)_{col}$ to denote the col-th bit of a_j;
- if $\mathsf{itype}[j] = \mathsf{proof}$, then a_j is interpreted as a "seed" such that $C(a_j)$ is a row of the proof matrix, and we use $(a_j)_{col}$ to denote the col-th bit of $\underline{C(a_j)}$. (NOT the col-th bit of a_j!)

For each $j \in [p]$, pc_j^{row} is a bit representing the contribution of $\mathsf{seed.row}$ in the j-th parity-check bit, i.e. $pc_j^{\mathsf{row}} := pc_j(\mathsf{seed.row}, 0^{|\mathsf{seed.col}|})$.

We first enumerate $\mathsf{seed.shared} \in \{0,1\}^{r_{\mathsf{shared}}}$. For each $\mathsf{seed.shared}$, we create an instance $\mathcal{I} := \mathcal{I}_{\mathsf{seed.shared}}$ of $\mathsf{Approx}_{\varepsilon}$-$\mathscr{C}'$-Satisfying-Pairs as follows. Let $\tilde{x}_j$ be the j-th row of $\mathsf{Encode}(x)$ (viewed as an $H_{\mathsf{input}} \times W_{\mathsf{input}}$ matrix). For each $\mathsf{seed.row} \in \{0,1\}^{r_{\mathsf{row}}}$, we add the following input to $\mathcal{I}$:

$$\mathsf{Input}_{\mathsf{seed.shared},\mathsf{seed.row}} = (a_1, \ldots, a_q, pc_1^{\mathsf{row}}, \ldots, pc_p^{\mathsf{row}}),$$

where for every $j \in [q]$,

$$a_j := \begin{cases} \tilde{x}_{\mathsf{irow}[j]} & \text{if } \mathsf{itype}[j] = \mathsf{input},\\ w_{\mathsf{irow}[j]} & \text{if } \mathsf{itype}[j] = \mathsf{proof}, \end{cases}$$

and pc_j^{row} is the contribution of $\mathsf{seed.row}$ to the j-th parity-check bit as defined above. Note that since $\tilde{n}_{\mathsf{hard}} = H_{\mathsf{input}} \cdot W_{\mathsf{input}}$, $\tilde{x}_{\mathsf{irow}[j]} \in \{0,1\}^{W_{\mathsf{input}}}$ when $\mathsf{itype}[j] = \mathsf{input}$, i.e., $\tilde{x}_{\mathsf{irow}[j]}$ will not contain $\perp$ (see Definition 2.5.4). The length of a_j is at most $\max\{W_{\mathsf{input}}, n\} \le n \cdot \mathrm{polylog}(\ell)$, thus the total length of $\mathsf{Input}_{\mathsf{seed.shared},\mathsf{seed.row}}$ is also bounded by $n \cdot \mathrm{polylog}(\ell)$.

Then, for every $\mathsf{seed.col} \in \{0,1\}^{r_{\mathsf{col}}}$, we define a circuit $C_{\mathsf{seed.shared},\mathsf{seed.col}}$ as follows. On input

$$(a_1, \ldots, a_q, pc_1^{\mathsf{row}}, \ldots, pc_p^{\mathsf{row}}),$$

it outputs

$$\mathsf{VDec}\Big((a_1)_{\mathsf{icol}[1]}, \ldots, (a_q)_{\mathsf{icol}[q]}, pc_1^{\mathsf{row}} \oplus pc_1^{\mathsf{col}}, \ldots, pc_p^{\mathsf{row}} \oplus pc_p^{\mathsf{col}}\Big).$$

Here, $\mathsf{VDec} \leftarrow V_{\mathsf{dec}}(\mathsf{seed.shared})$ is the decision predicate of VPCPP and pc_i^{col} represents the contribution of $\mathsf{seed.col}$ to the i-th parity-check bit, i.e., $pc_i^{\mathsf{col}} := pc_i(0^{|\mathsf{seed.row}|}, \mathsf{seed.col})$. Note that by definition, $pc_i(\mathsf{seed.row}, \mathsf{seed.col}) = pc_i^{\mathsf{row}} \oplus pc_i^{\mathsf{col}}$. Also note that $C_{\mathsf{seed.shared},\mathsf{seed.col}}$ is indeed an $\mathsf{OR}_2 \circ \mathscr{C}$ circuit, since VDec is always the OR of its two input bits or their negation.

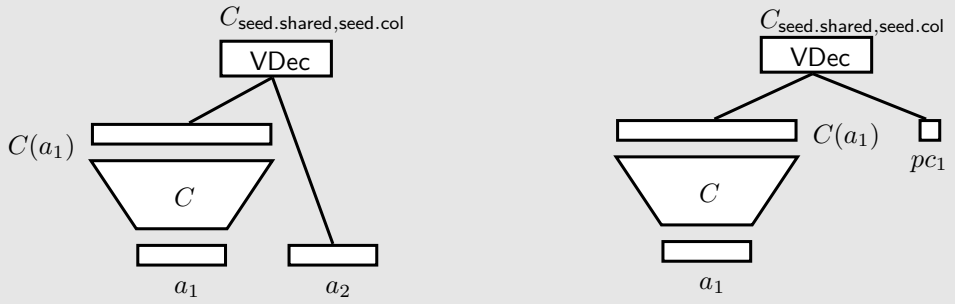


Figure 3.1: Examples of the circuit $C_{\mathsf{seed.shared},\mathsf{seed.col}}$. In the left example, there are two queries and no parity-check bits, the first query has type proof, and the second query has type input. In the right example, there are one query with type proof and one parity-check bit.

Now, our instance $\mathcal{I}$ contains $M := 2^{r_{\mathsf{row}}}$ inputs and $N := 2^{r_{\mathsf{col}}}$ circuits. By definition,

$$\mathsf{VPCPP}^{\mathsf{Encode}(x)\circ\pi}(\mathsf{seed}) = C_{\mathsf{seed.shared},\mathsf{seed.col}}(\mathsf{Input}_{\mathsf{seed.shared},\mathsf{seed.row}}).$$

Since $M = \ell^{c+1-\eta} \cdot \mathrm{polylog}(\ell)$ and $N = \ell^{1-\eta} \cdot \mathrm{polylog}(\ell)$, there is a non-trivial algorithm for $\mathsf{Approx}_\varepsilon$-$\mathscr{C}'$-Satisfying-Pairs with N circuits of size s and M inputs of length $n \cdot \mathrm{polylog}(\ell)$. In particular, we can estimate $p_{\mathsf{acc}}(\mathsf{seed.shared})$ using this algorithm on $\mathcal{I}_{\mathsf{seed.shared}}$ up to an additive error ε, where

$$p_{\mathsf{acc}}(\mathsf{seed.shared}) := \Pr_{\mathsf{seed.row},\mathsf{seed.col}}\left[\mathsf{VPCPP}^{\mathsf{Encode}(x)\circ\pi}(\mathsf{seed})\right].$$

In other words, we can obtain a $p'_{\mathsf{acc}}(\mathsf{seed.shared}) \in p_{\mathsf{acc}}(\mathsf{seed.shared}) \pm \varepsilon$. The overall acceptance probability of VPCPP on the input $\mathsf{Encode}(x)$ and proof π is

$$\begin{aligned}
p_{\mathsf{acc}} &:= \Pr_{\mathsf{seed} \leftarrow \{0,1\}^r}\left[\mathsf{VPCPP}^{\mathsf{Encode}(x)\circ\pi}(\mathsf{seed})\right] \\
&= \mathop{\mathbb{E}}_{\mathsf{seed.shared}}\left[\Pr_{\mathsf{seed.row},\mathsf{seed.col}}\left[\mathsf{VPCPP}^{\mathsf{Enc}(x)\circ\pi}(\mathsf{seed})\right]\right] \\
&= \mathop{\mathbb{E}}_{\mathsf{seed.shared}}\left[p_{\mathsf{acc}}(\mathsf{seed.shared})\right] \\
&\in \mathop{\mathbb{E}}_{\mathsf{seed.shared}}\left[p'_{\mathsf{acc}}(\mathsf{seed.shared})\right] \pm \varepsilon.
\end{aligned}$$

Hence we can estimate p_{acc} up to an additive error ε by taking average over all $p'_{\mathsf{acc}}(\mathsf{seed.shared})$ obtained by the $\mathsf{Approx}_\varepsilon$-$\mathscr{C}'$-Satisfying-Pairs algorithm over $\mathcal{I}_{\mathsf{seed.shared}}$.

To summarise, our algorithm M^{PCPP} works as follows. It first computes $\mathsf{Encode}(x)$ in $O(n)$ time. Then, it enumerates $\mathsf{seed.shared}$, produces the instance $\mathcal{I}_{\mathsf{seed.shared}}$, and feeds it to the algorithm for $\mathsf{Approx}_\varepsilon$-$\mathscr{C}'$-Satisfying-Pairs to obtain $p'_{\mathsf{acc}}(\mathsf{seed.shared})$. Let p'_{acc} be the average

of $p'_{\mathsf{acc}}(\mathsf{seed.shared})$ over all $\mathsf{seed.shared} \in \{0,1\}^{r_{\mathsf{shared}}}$. It accepts if and only if $p'_{\mathsf{acc}} \geq c_{\mathsf{pcp}} - \varepsilon$.

Correctness of M^{PCPP}. For every $x \in \{0,1\}^{n_{\mathsf{hard}}}$, we know by the discussion above that:

- If $x \notin L^{\mathsf{hard}}$, we know that $\mathsf{Encode}(x)$ is δ_{code} far from being in L^{enc}. By the soundness of VPCPP, $p_{\mathsf{acc}} \leq s_{\mathsf{pcp}}$, which further means that $p'_{\mathsf{acc}} \leq p_{\mathsf{acc}} + \varepsilon < c_{\mathsf{pcp}} - \varepsilon$, hence M^{PCPP} will reject x.
- If $x \in L^{\mathsf{hard}}$ has an *easy witness*, we can see by the definition of easiness that there is a proof π of $\mathsf{Encode}(x) \in L^{\mathsf{enc}}$ such that for every row $\pi_i \in \{0,1\}^{W_{\mathsf{proof}}}$ of π, there is a string $w_i \in \{0,1\}^n$ such that $\pi_i = C(w_i)$. These w_i can be found by non-deterministic guessing at the beginning of M^{PCPP}. In such case, we know by the completeness of VPCPP that $p_{\mathsf{acc}} \geq c_{\mathsf{pcp}}$, which further means that $p'_{\mathsf{acc}} \geq p_{\mathsf{acc}} - \varepsilon \geq c_{\mathsf{pcp}} - \varepsilon$. Therefore M^{PCPP} will accept x.

Complexity of M^{PCPP}. Each instance $\mathcal{I}$ of $\mathsf{Approx}_{\varepsilon}$-$\mathscr{C}'$-Satisfying-Pairs contains $M := 2^{r_{\mathsf{row}}}$ inputs and $N := 2^{r_{\mathsf{col}}}$ circuits. Since each instance can be solved in $NM/\log^{c_0}(NM)$ time, the total time are

$$\begin{aligned} & 2^{r_{\mathsf{shared}}} \cdot NM/\log^{c_0}(NM) \\ \leq\ & 2^{r_{\mathsf{shared}}} \cdot 2^{r_{\mathsf{row}}} \cdot 2^{r_{\mathsf{col}}}/r^{c_0} \\ \leq\ & 2^r/r^{c_0}. \end{aligned}$$

Recall that $r = \log T + O(\log\log T + m\log m)$, where $O(\cdot)$ hides some absolute constant, we can see that $2^r/r^{c_0} = T\log^{O(1)} T/\log^{c_0} T$. By setting c_0 to be an sufficiently large absolute constant depending on c_{hard}, we can make $2^r/r^{c_0} \leq T/\log^{c_{\mathsf{hard}}} T$. Also, we can compute $\mathsf{Encode}(x)$ in $O(n_{\mathsf{hard}})$ time, and this is not the bottleneck. Therefore, the total running time of M^{PCPP} is at most $T/\log^{c_{\mathsf{hard}}} T$.

It then suffices to determine the advice and non-determinism complexity of M^{PCPP}. For every $\mathsf{seed.shared}$, the machine M^{PCPP} needs the data structure $\mathsf{DS}_{\mathsf{seed.shared}}$ as advice to support the algorithm for Satisfying Pairs. Since $|\mathsf{DS}_{\mathsf{seed.shared}}| \leq N^c = 2^{cr_{\mathsf{col}}}$ by the assumption, the advice complexity of M^{PCPP} is

$$2^{cr_{\mathsf{col}}+r_{\mathsf{shared}}} \leq \ell^{c-c\eta+2\eta} \leq \ell^{c+1} \leq n_{\mathsf{hard}}/10.$$

Also, the number of nondeterministic bits that M^{PCPP} guesses is at most $\hat{H}_{\mathsf{proof}} \cdot n \leq n_{\mathsf{hard}}/10$. Therefore, we can see that

$$M^{\mathsf{PCPP}} \in \mathsf{NTIMEGUESS}_{\mathsf{RAM}}[T/\log^{c_{\mathsf{hard}}}(T), n_{\mathsf{hard}}/10]_{/(n_{\mathsf{hard}}/10)}.$$

The final algorithm. Given a multi-output circuit $C : \{0,1\}^n \to \{0,1\}^\ell$, our algorithm for finding a non-output of C works as follows. First, we construct the hard language L^{hard} and the algorithm M^{PCPP}. Since M^{PCPP} is a nondeterministic algorithm that runs in $T/\log^{c_{\mathsf{hard}}}(T)$ time, uses at most $n_{\mathsf{hard}}/10$ bits of nondeterminism and at most $n_{\mathsf{hard}}/10$ bits of advice, it follows that there is an input $x_{\mathsf{hard}} \in \{0,1\}^{n_{\mathsf{hard}}}$ such that $M^{\mathsf{PCPP}}(x_{\mathsf{hard}}) \neq L^{\mathsf{hard}}(x_{\mathsf{hard}})$. Moreover, let α be

the advice string fed to M^{PCPP}, i.e., the data structures $\mathsf{DS}_{\mathsf{seed.shared}}$ for each $\mathsf{seed.shared}$. (Note that we can obtain α since the avoidance algorithm has an NP oracle.) We can find such an input x_{hard} by running $\mathcal{R}(1^{n_{\mathsf{hard}}}, M^{\mathsf{PCPP}}, \alpha)$, where $\mathcal{R}$ is the refuter guaranteed by Theorem 3.3.1. Thus, we can find x_{hard} in $\mathrm{poly}(T)$ time with an NP oracle.

If $x_{\mathsf{hard}} \notin L^{\mathsf{hard}}$, then M^{PCPP} also rejects x_{hard}, which means $M^{\mathsf{PCPP}}(x_{\mathsf{hard}}) = L^{\mathsf{hard}}(x_{\mathsf{hard}})$. Thus, it has to be the case that $x_{\mathsf{hard}} \in L^{\mathsf{hard}}$ but M^{PCPP} rejects x_{hard}. Therefore, x_{hard} does not have an easy witness. We can then use the NP oracle to find the lexicographically first proof matrix π such that

$$\Pr_{\mathsf{seed} \leftarrow \{0,1\}^r}[\mathsf{VPCPP}^{\mathsf{Encode}(x)\circ\pi}(\mathsf{seed}) \text{ accepts}] \geq c_{\mathsf{pcp}}.$$

Treating π as a matrix of dimension $\hat{H}_{\mathsf{proof}} \times W_{\mathsf{proof}}$, there has to be a row that is not in the range of C. We can pick such a row by using the NP oracle. □

Remark 3.4.3. In Theorem 3.4.2, we assumed a non-trivial Satisfying-Pairs algorithm for the circuit class $\mathsf{OR}_2 \circ \mathscr{C}$. By Theorem 3.3.9, a non-trivial Satisfying-Pairs algorithm for $\mathsf{AND}_2 \circ \mathscr{C}$ or $\mathsf{XOR}_2 \circ \mathscr{C}$ also suffices. This property might be useful for some circuit classes with a better closure property under top XOR_2 gates (or AND_2 gates).

By replacing the 2-query PCPP (with imperfect completeness) with the 3-query PCPP (with perfect completeness) in Theorem 2.5.10, we can show that non-trivial algorithms for Gap_ε-$\mathscr{C}'$-Satisfying-Pairs where $\mathscr{C}' = \mathsf{OR}_3 \circ \mathscr{C}$ also imply $\mathsf{FP}^{\mathsf{NP}}$ algorithms for $\mathscr{C}$-Avoid.

Corollary 3.4.4. *There are constants $\varepsilon > 0$ and c_0 such that the following holds. Let $0 < \eta < 1/2$ be a constant, $\ell(n) > n^{1+4\eta}$ be a good function. Let $s = s(n)$ be a size parameter, $\mathscr{C}[s]$ be a typical circuit class where s is a size parameter, and $\mathscr{C}'[3s] := \mathsf{OR}_3 \circ \mathscr{C}[s]$ (i.e. a $\mathscr{C}'$ circuit of size $3s$ refers to an OR_3 of at most two $\mathscr{C}$ circuits of size s).*

Assumption: *Suppose that for some constant $c \geq 1$, there is an $(NM/\log^{c_0}(NM))$-time algorithm for Gap_ε-$\mathscr{C}'$-Satisfying-Pairs with $N := \ell^{1-\eta} \cdot \mathrm{polylog}(\ell)$ circuits of size $s(n)$ and $M := \ell^{c+1-\eta} \cdot \mathrm{polylog}(\ell)$ inputs of length $n \cdot \mathrm{polylog}(\ell)$, allowing a P^{NP} preprocessing of an N^c-size data structure.*

Conclusion: *Then there is an $\mathsf{FP}^{\mathsf{NP}}$ algorithm for $\mathscr{C}[s]$-Avoid with stretch $\ell(n)$.*

Proof Sketch. Compared to the proof of Theorem 3.4.2, the only difference is that the Gap_ε-$\mathscr{C}'$-Satisfying-Pairs algorithm can only distinguish between the case that $p_{\mathsf{acc}}(\mathsf{seed.shared}) = 1$ and that $p_{\mathsf{acc}}(\mathsf{seed.shared}) \leq 1 - \varepsilon$, and our algorithm M^{PCPP} rejects immediately if there is a $\mathsf{seed.shared}$ such that $p_{\mathsf{acc}}(\mathsf{seed.shared}) \neq 1$.

If the PCPP has perfect completeness, we can still distinguish between the case that $x \notin L^{\mathsf{hard}}$ and the case that $x \in L^{\mathsf{hard}}$ has an easy witness. Indeed, if $x \notin L^{\mathsf{hard}}$, then $p_{\mathsf{acc}} \leq 1 - \varepsilon$, hence by an averaging argument there exists some $\mathsf{seed.shared}$ such that $p_{\mathsf{acc}}(\mathsf{seed.shared}) < 1 - \varepsilon$; on the other hand, if $x \in L^{\mathsf{hard}}$ has an easy witness, then (on some nondeterministic guess of M^{PCPP}) we have that $p_{\mathsf{acc}}(\mathsf{seed.shared}) = 1$ for every $\mathsf{seed.shared}$. □

3.5 Remote Point

Theorem 3.5.1. *There is a universal constant $c_u \geq 1$ such that the following holds. Let $N := N(n)$ be a parameter such that $2^{\log^{c_u} n} < N < 2^{n^{0.99}}$, $\varepsilon := \varepsilon(n) > n^{-c_u}$ be the error parameter, and $\ell := N^{c_u \log(1/\varepsilon)}$. Let $\mathscr{C}[s]$ be a typical circuit class, where $s := s(n) \leq N$ is a size parameter, and denote $\mathscr{C}'[c_u s] := \mathsf{AND}_{c_u} \circ \mathscr{C}[s]$ (i.e. a $\mathscr{C}'$ circuit of size $c_u s$ refers to the AND of at most c_u $\mathscr{C}$ circuits of size s).*

Assumption: *Let $P := (\log N)^{\log(1/\varepsilon)}$. Suppose there is a deterministic algorithm running in time $T^{\mathrm{alg}} := N^2/P^{c_u}$ that, given as input a list of N $\mathscr{C}'[c_u s]$ circuits $\{C_i\}$ and a list of N inputs $\{x_j\}$ with input length $n \cdot \mathrm{polylog}(\ell)$, estimates $\Pr_{i,j \leftarrow [N]}[C_i(x_j)]$ with additive error $\eta := \varepsilon^{c_u}$. For some constant c, this algorithm is allowed to have a P^{NP} preprocessing phase on circuits that outputs a "data structure" of length $\ell_{\mathrm{DS}} := N^c$.*

Conclusion: *Then there is an $\mathsf{FP}^{\mathsf{NP}}$ algorithm that takes as input a circuit $C : \{0,1\}^n \to \{0,1\}^\ell$, where each output bit of C can be computed in $\mathscr{C}[s]$, and prints a string y that is $(1/2 - \varepsilon)$-far from $\mathrm{Range}(C)$.*

The rest of this section is devoted to proving Theorem 3.5.1. Since the proof is quite technical and consists of a few components, we give an overview below:

Overview of Section 3.5

- In Section 3.5.1, we define a circuit class called $\mathsf{Prod}_d \circ \mathsf{Sum} \circ \mathscr{C}$, and show that a Satisfying-Pairs algorithm for $\mathsf{AND}_d \circ \mathscr{C}$ implies a Satisfying-Pairs algorithm for this class. This will be a convenient tool for our subsequent arguments.
- To solve the remote point problem, we need to define a nondeterministic machine called M^{PCPP} trying to contradict the nondeterministic time hierarchy (Theorem 3.3.1). In Section 3.5.2, we set the framework for this machine: it uses the PCPP theorem in Theorem 2.5.11, guesses a "compressed" version of the PCPP proof, and verifies the validity of this PCPP proof without decompressing it.
- The first problem we encounter is the "non-Booleanness" of the PCPP proof. As we use Theorem 3.3.5, the decompressed proof consists of real numbers instead of Boolean values, and we need to check whether the decompressed proof is "close to Boolean" (in a carefully defined technical sense). This is done in Section 3.5.3 via the Satisfying-Pairs algorithm.
- In Section 3.5.4, we use the faster algorithm for Satisfying-Pairs to verify the PCPP proof. This step is straightforward but tedious.
- After we obtain a non-trivial algorithm for verifying the PCPP proof, we conclude the machine M^{PCPP} in Section 3.5.5. Then we use this machine to build an $\mathsf{FP}^{\mathsf{NP}}$ algorithm for the remote point problem in Section 3.5.6.

3.5.1 Satisfying-Pairs for $\mathsf{Prod}_d \circ \mathsf{Sum} \circ \mathscr{C}$ Circuits

It turns out that as an intermediate step, we need a Satisfying-Pairs algorithm for the following class of multi-output circuits that output real numbers. Let $d \geq 1$ be a constant, denote by $\mathsf{Prod}_d \circ \mathsf{Sum} \circ \mathscr{C}$ the class of multi-output circuits which take two inputs $x \in \{0,1\}^n$ and α, and have the following components:

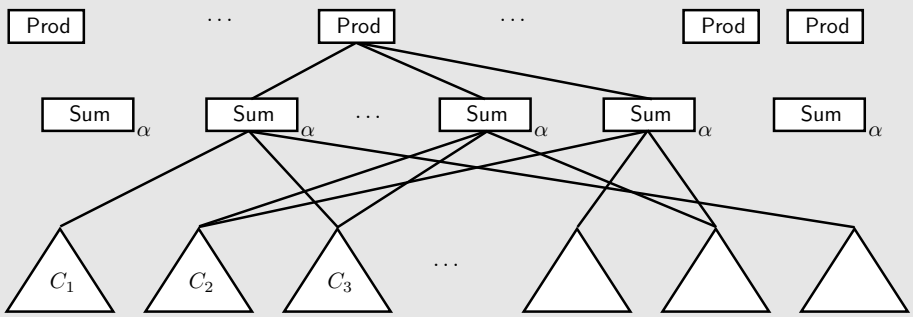


Figure 3.2: Example of a $\mathsf{Prod}_d \circ \mathsf{Sum} \circ \mathscr{C}$ circuit.

- Let $\ell_{\mathscr{C}}$ denote the number of bottom $\mathscr{C}$ circuits. For each $i \in [\ell_{\mathscr{C}}]$, the i-th bottom circuit is a $\mathscr{C}$ circuit computing a function $C_i : \{0,1\}^n \to \{0,1\}$.
- Let ℓ_{Sum} denote the number of middle "linear sum" gates. For each $i \in [\ell_{\mathsf{Sum}}]$, the i-th gate outputs a real number

$$\mathsf{Sum}_i(x, \alpha) := \sum_{k \in [A]} \mathsf{coeff}_k(\alpha) \cdot C_{\mathsf{idx}_k(\alpha, i)}(x).$$

  (See Definition 3.3.3 for the definition of linear sum circuits, in particular the *coefficient sum* and *locality* of a linear sum circuit. Note that this definition is different from the definitions in [Wil18b, CW19b] and in Chapter 4.)
- Let ℓ_{Prod} denote the number of output gates. Each output gate is a product (i.e., multiplication) gate of fan-in d and is connected to the $q_1(i), q_2(i), \ldots, q_d(i)$-th linear sum gate. It outputs the real number

$$C_i^{\mathsf{Prod}}(x, \alpha) := \prod_{t=1}^{d} \mathsf{Sum}_{q_t(i)}(x, \alpha).$$

Remark 3.5.2. The important complexity measures of a $\mathsf{Prod}_d \circ \mathsf{Sum} \circ \mathscr{C}$ circuit are:
- the number of gates in each level ($\ell_{\mathscr{C}}$, ℓ_{Sum}, ℓ_{Prod});
- the fan-in of the top Prod gates (d);
- the fan-in (A), coefficient sum (U), and locality (l) of the linear sum layer.

We show that a Satisfying Pairs algorithm for $\mathsf{AND}_d \circ \mathscr{C}$ circuits implies a "Satisfying Pairs algorithm" for $\mathsf{Prod}_d \circ \mathsf{Sum} \circ \mathscr{C}$ circuits that given a $\mathsf{Prod}_d \circ \mathsf{Sum} \circ \mathscr{C}$ circuit and a list of input strings, estimates the expected output value (as a real number) of a random output Prod gate in the circuit on a random input string in the list.

Theorem 3.5.3. *Let $\mathscr{C}$ be a typical circuit class and $\eta \in (0,1)$ be a parameter. Suppose there is a deterministic algorithm running in time $T^{\mathrm{alg}} = T^{\mathrm{alg}}(N, M)$ that, given as inputs a list of $\hat{N} \leq N$ $\mathsf{AND}_d \circ \mathscr{C}$ circuits $\{C_i\}$ and a list of $\hat{M} \leq M$ inputs $\{x_j\}$, estimates the following quantity with additive error η:*

$$\Pr_{i \leftarrow [\hat{N}], j \leftarrow [\hat{M}]}[C_i(x_j)].$$

Then, there is a deterministic algorithm running in time $A^d(2^{dl}+M'/M)\cdot\lceil \ell_{\mathsf{Prod}}/N\rceil\cdot O(T^{\mathrm{alg}})$ *that, given as input a* $\mathsf{Prod}_d\circ\mathsf{Sum}\circ\mathscr{C}$ *circuit* C^{Prod} *with parameters specified in Remark 3.5.2, and a list of* M' *inputs* $\{(x_j,\alpha_j)\}$*, estimates the following quantity with additive error* $\eta\cdot U^d$*:*

$$\mathop{\mathbb{E}}_{i\leftarrow[\ell_{\mathsf{Prod}}],j\leftarrow[M']}\left[C_i^{\mathsf{Prod}}(x_j,\alpha_j)\right].$$

Moreover, if the algorithm in the hypothesis requires a P^{NP} *preprocessing phase on the circuits that outputs a "data structure" of length* ℓ_{DS}*, then the algorithm in the conclusion requires a* P^{NP} *preprocessing phase on* C^{Prod} *that outputs a "data structure" of length* $O(A^d2^{dl})\cdot\lceil\ell_{\mathsf{Prod}}/N\rceil\cdot\ell_{\mathrm{DS}}$*.*

Proof. For any fixed i and j, we know that

$$\begin{aligned}
C_i^{\mathsf{Prod}}(x_j,\alpha_j)&=\prod_{t=1}^{d}\mathsf{Sum}_{q_t(i)}(x_j,\alpha_j)\\
&=\prod_{t=1}^{d}\sum_{k\in[A]}\mathsf{coeff}_k(\alpha_j)\cdot C_{\mathsf{idx}_k(\alpha_j,q_t(i))}(x_j)\\
&=\sum_{k_1\in[A]}\sum_{k_2\in[A]}\cdots\sum_{k_d\in[A]}\prod_{t=1}^{d}\Big(\mathsf{coeff}_{k_t}(\alpha_j)\cdot C_{\mathsf{idx}_{k_t}(\alpha_j,q_t(i))}(x_j)\Big).
\end{aligned}\tag{3.6}$$

As we can enumerate $k_1,k_2,\ldots,k_d\in[A]$ in A^d time, it suffices to estimate

$$\mathop{\mathbb{E}}_{i\leftarrow[\ell_{\mathsf{Prod}}],j\leftarrow[M']}\left[\prod_{t=1}^{d}\Big(\mathsf{coeff}_{k_t}(\alpha_j)\cdot C_{\mathsf{idx}_{k_t}(\alpha_j,q_t(i))}(x_j)\Big)\right].\tag{3.7}$$

Fix $k_1,k_2,\ldots,k_d\in[A]$. Since C^{Prod} is of locality l, we can see that (3.7) only depends on dl bits of α_j. We partition $j\in[M']$ into 2^{dl} groups as follows: For each $\alpha\in\{0,1\}^{dl}$, let $\mathcal{J}_\alpha$ be the set of $j\in[M']$ such that the dl bits of α_j (that (3.7) for this j depends on) equals to α. We will estimate (3.7) by enumerating $\alpha\in\{0,1\}^{dl}$, estimating it for $j\leftarrow\mathcal{J}_\alpha$ (instead of $j\leftarrow[M']$), and then taking the (weighted) average over all possible α.

Now we fix any $\alpha\in\{0,1\}^{dl}$. We can rephrase the following two expressions as they no longer depend on α_j:

$$\begin{aligned}
\mathsf{coeff}_{k_t}(\alpha_j)&=:\mathsf{coeff}'_t;\\
C_{\mathsf{idx}_{k_t}(\alpha_j,q_t(i))}(x_j)&=:C_{\mathsf{idx}'_t(i)}(x_j).
\end{aligned}$$

It then suffices to estimate

$$\mathop{\mathbb{E}}_{i\leftarrow[\ell_{\mathsf{Prod}}],j\leftarrow\mathcal{J}_\alpha}\left[\prod_{t=1}^{d}\mathsf{coeff}'_t\cdot C_{\mathsf{idx}'_t(i)}(x_j)\right]=\left(\prod_{t=1}^{d}\mathsf{coeff}'_t\right)\cdot\mathop{\mathbb{E}}_{i\leftarrow[\ell_{\mathsf{Prod}}],j\leftarrow\mathcal{J}_\alpha}\left[\bigwedge_{t=1}^{d}C_{\mathsf{idx}'_t(i)}(x_j)\right].\tag{3.8}$$

Each expression of the form $\mathbb{E}_{i,j}\left[\bigwedge_{t=1}^{d}C_{\mathsf{idx}'_t(i)}(x_j)\right]$ can be reduced to the Satisfying-Pairs problem for $\mathsf{AND}_d\circ\mathscr{C}$ circuits. More precisely, we split $\mathcal{J}_\alpha$ into blocks of size M, split $[\ell_{\mathsf{Prod}}]$ into blocks of size N, and use the assumed algorithm (which works for N $\mathsf{AND}_d\circ\mathscr{C}$ circuits and M inputs) to estimate (3.8) within an additive error of $\left(\eta\cdot\prod_{t=1}^{d}\left|\mathsf{coeff}'_t\right|\right)$ in $T^{\mathrm{alg}}\cdot\lceil|\mathcal{J}_\alpha|/M\rceil\cdot$

$\lceil \ell_{\mathsf{Prod}}/N \rceil$ time.[14] We substitute this estimation in (3.7) and then in (3.6) to obtain our final algorithm.

Running time. Consider the subroutine for estimating (3.7). This subroutine itself is reduced to subroutines for each $\mathcal{J}_\alpha$, which takes $T^{\mathrm{alg}} \cdot \lceil |\mathcal{J}_\alpha|/M \rceil \cdot \lceil \ell_{\mathsf{Prod}}/N \rceil$ time. The time complexity of this subroutine is

$$O(T^{\mathrm{alg}}) \cdot \lceil \ell_{\mathsf{Prod}}/N \rceil \cdot \sum_\alpha \lceil |\mathcal{J}_\alpha|/M \rceil \le O(T^{\mathrm{alg}}) \cdot (2^{dl} + M'/M) \cdot \lceil \ell_{\mathsf{Prod}}/N \rceil.$$

We invoked this subroutine A^d times by enumerating $k_1, k_2, \ldots, k_d \in [A]$ to estimate (3.6), so the total time complexity of our algorithm is $A^d(2^{dl} + M'/M)\lceil \ell_{\mathsf{Prod}}/N \rceil O(T^{\mathrm{alg}})$.

Additive error. Our estimation of (3.8) is within an additive error of $\left(\eta \cdot \prod_{t=1}^d |\mathsf{coeff}'_t|\right)$. Thus, our estimation of (3.7) is within an additive error of

$$\eta \cdot \sum_{\alpha \in \{0,1\}^{dl}} \frac{|\mathcal{J}_\alpha|}{M'} \cdot \prod_{i=1}^d |\mathsf{coeff}'_t| = \eta \cdot \mathop{\mathbb{E}}_j \left[\prod_{i=1}^d |\mathsf{coeff}_{k_t}(\alpha_j)| \right].$$

It follows that our estimation of (3.6) is within an additive error of

$$\begin{aligned}
&\eta \cdot \sum_{k_1 \in [A]} \sum_{k_2 \in [A]} \cdots \sum_{k_d \in [A]} \mathop{\mathbb{E}}_j \left[\prod_{i=1}^d |\mathsf{coeff}_{k_t}(\alpha_j)| \right] \\
=\, &\eta \cdot \mathop{\mathbb{E}}_j \left[\left(\sum_{k \in [A]} |\mathsf{coeff}_k(\alpha_j)| \right)^d \right] \\
\le\, &\eta \cdot U^d.
\end{aligned}$$

Preprocessing. Finally, suppose the assumed Satisfying-Pairs algorithm for $\mathsf{AND}_d \circ \mathscr{C}$ circuits require a P^{NP} preprocessing phase on the circuits. The P^{NP} preprocessing phase for our algorithm enumerates $k_1, k_2, \ldots, k_d \in [A]$ and $\alpha \in \{0,1\}^{dl}$. For each $(k_1, k_2, \ldots, k_d, \alpha)$, it splits the ℓ_{Prod} circuits $\{C_{\mathsf{idx}'_t(i)}\}$ into blocks of size N, invokes the preprocessing algorithm on each block, and computes a "data structure" for this block. The final "data structure" outputted by the preprocessing phase of our algorithm consists of the concatenation of all these "data structures" computed, which has total length $O(A^d 2^{dl}) \cdot \lceil \ell_{\mathsf{Prod}}/N \rceil \cdot \ell_{\mathrm{DS}}$. □

3.5.2 Set Up

Suppose that we are given a $\mathscr{C}$ circuit $C : \{0,1\}^n \to \{0,1\}^\ell$ as input. Let q, c_m, c_{tm} be constants that will be determined later. Define

$$\begin{aligned}
\delta &:= (10^9 q)^{-10q^2}, \\
m &:= c_m \log(1/\varepsilon)/\delta,
\end{aligned}$$

[14] Note that at most one of the block may contain less than M inputs. However, the assumed algorithm works for input number $\le M$ as well, and this will not have any blow-up on the error factor.

$$
\begin{aligned}
w_{\mathsf{proof}} &:= (300q/m)\log \ell, & W_{\mathsf{proof}} &:= 2^{w_{\mathsf{proof}}} = \ell^{O(\delta/\log(1/\varepsilon))},\\
h_{\mathsf{proof}} &:= (25q+1)\log \ell, & H_{\mathsf{proof}} &:= 2^{h_{\mathsf{proof}}} = \ell^{25q+1},\\
n_{\mathsf{hard}} &:= 20H_{\mathsf{proof}} \cdot n^{c_u} \cdot (1/\varepsilon)^{c_u \log\log \ell},\\
T &:= H_{\mathsf{proof}} \cdot W_{\mathsf{proof}}/\log^{c_{\mathsf{tm}}}(\ell).
\end{aligned}
$$

Let L^{hard} be the hard language constructed in Theorem 3.3.1, i.e.,

$$
L^{\mathsf{hard}} \in \mathsf{NTIME}_{\mathsf{TM}}[T] \setminus \text{i.o.-}\mathsf{NTIMEGUESS}_{\mathsf{RAM}}[T/\log^{c_{\mathsf{hard}}}(T), n_{\mathsf{hard}}/10]/_{(n_{\mathsf{hard}}/10)},
$$

where n_{hard} refers to the input length and c_{hard} is an absolute constant.

We now show that $T \geq n_{\mathsf{hard}} \cdot \mathrm{polylog}(n_{\mathsf{hard}})$, which means that the technical condition of Theorem 3.3.1 is satisfied. In fact, $\mathrm{polylog}(n_{\mathsf{hard}}) \leq \mathrm{polylog}(\ell)$, hence it suffices to show that $W_{\mathsf{proof}} \geq (n\cdot(1/\varepsilon)^{\log\log \ell})^{\omega(1)}$, i.e., $\ell \geq (n\cdot(1/\varepsilon)^{\log\log \ell})^{\omega(\log(1/\varepsilon))}$. This is true since $\ell = N^{c_u \log(1/\varepsilon)}$ and $n, (1/\varepsilon)^{\log\log \ell} < 2^{O(\log^{1/2} N)}$.

Like in the proof of Theorem 3.4.2, we describe a nondeterministic RAM M^{PCPP} that runs in $T/\log^{c_{\mathsf{hard}}}(T)$ time, uses $n_{\mathsf{hard}}/10$ advice bits, guesses $n_{\mathsf{hard}}/10$ nondeterministic bits, and attempts to solve L^{hard}. We will show that for every input $x \in \{0,1\}^{n_{\mathsf{hard}}}$, if $x \notin L^{\mathsf{hard}}$ then $M^{\mathsf{PCPP}}(x)$ rejects; while if $x \in L^{\mathsf{hard}}$ and *has an easy witness*, then $M^{\mathsf{PCPP}}(x)$ accepts. However, to solve $\mathscr{C}$-Remote-Point, we need a slightly different definition for "easy witness".

Let VPCPP be the verifier for the smooth and rectangular PCPP (Theorem 2.5.11) for the language

$$
L^{\mathsf{enc}} := \{\mathsf{Encode}(x) : x \in L^{\mathsf{hard}}\},
$$

where we fix an error-correcting code $(\mathsf{Encode}, \mathsf{Decode})$ as in Theorem 2.4.1. Let δ_{code} be the (relative) distance of the error-correcting code. Suppose a string of length n_{hard} is encoded (via Encode) into a string of length $\tilde{n}_{\mathsf{hard}} := \Theta(n_{\mathsf{hard}})$. We set the following parameters:

$$
\begin{aligned}
h_{\mathsf{input}} &:= \left(1 - \frac{\Theta(m^2 \log\log T)}{\log T}\right) h_{\mathsf{proof}}, & H_{\mathsf{input}} &:= 2^{h_{\mathsf{input}}} = H_{\mathsf{proof}}/\mathrm{poly}(2^{m^2}, \log \ell),\\
w_{\mathsf{input}} &:= \lceil \log \tilde{n}_{\mathsf{hard}} \rceil - h_{\mathsf{input}}, & W_{\mathsf{input}} &:= 2^{w_{\mathsf{input}}} = \mathrm{poly}(n, 2^{m^2}, \log \ell).
\end{aligned}
$$

Again, we assume without loss of generality that $\tilde{n}_{\mathsf{hard}} = H_{\mathsf{input}} \cdot W_{\mathsf{input}}$.

We invoke Theorem 2.5.11 for L^{enc} to obtain a verifier VPCPP with proof size $\hat{H}_{\mathsf{proof}} \times W_{\mathsf{proof}}$ and input size $H_{\mathsf{input}} \times W_{\mathsf{input}}$, where $\hat{H}_{\mathsf{proof}} = 2^{\hat{h}_{\mathsf{proof}}}$ for some $\hat{h}_{\mathsf{proof}} = \log T + \Theta(m \log\log T) - w_{\mathsf{proof}}$. We can check the technical requirements of Theorem 2.5.11 as follows:

$$
\begin{aligned}
m &= \Theta(\log n/\delta) \leq (\log T)^{0.1},\\
w_{\mathsf{proof}} &= (300q/m)\log \ell \geq (5/m)\log T,\\
\hat{h}_{\mathsf{proof}} &= h_{\mathsf{proof}} + \Theta(\log(1/\varepsilon)\log\log \ell) \geq (25q+1)\log \ell \geq (5/m)\log T,\\
\frac{w_{\mathsf{input}}}{w_{\mathsf{proof}}} &= \frac{O(m^2 + \log n + \log\log \ell)}{(\log \ell)/m} < o(1).
\end{aligned}
$$

By Theorem 2.5.11, VPCPP has the following parameters:

- soundness error $= 1/2$,
- proximity parameter $= \delta_{\mathsf{code}}$,
- query complexity $= q := O(1)$,
- parity-check complexity $= q := O(1)$,
- total randomness $= r := w_{\mathsf{proof}} + \hat{h}_{\mathsf{proof}} + O(\log\log T + m\log m) = \log T + O(m\log\log T)$,
- row randomness $= r_{\mathsf{row}} := \hat{h}_{\mathsf{proof}} - (5/m)\log T = \Theta(\log \ell)$,
- column randomness $= r_{\mathsf{col}} := w_{\mathsf{proof}} - (5/m)\log T = \Theta((\log \ell)/m)$,
- shared randomness $= r_{\mathsf{shared}} := (10/m)\log T + O(\log\log T + m\log m) = \Theta((\log \ell)/m)$.

Here, all the $\Theta(\cdot)$ hides constants that may depend on q, c_m, c_{tm}. Moreover, as we choose the soundness error and the proximity parameters to be absolute constants, the query complexity q is also an absolute constant.

Note that if c_u is large enough, then we have $2^{\Omega(r_{\mathsf{col}})} \le N \le 2^{r_{\mathsf{col}}}$. (This is because $r_{\mathsf{col}} = qc_u\delta/c_m \cdot \Theta(\log N)$.) Therefore, we can solve the Approx_η-Satisfying-Pairs problem for $2^{r_{\mathsf{col}}}$ inputs and $2^{r_{\mathsf{col}}}$ $\mathsf{AND}_{c_u} \circ \mathscr{C}$ circuits, by partitioning the inputs and circuits into groups of size N. The time complexity is still at most $2^{2r_{\mathsf{col}}}/P^{c_u}$, where $P := (r_{\mathsf{col}})^{\log(1/\varepsilon)}$. Without loss of generality, we may assume $N = 2^{r_{\mathsf{col}}}$ in what follows. It still holds that our Satisfying-Pairs algorithm has a P^{NP} preprocessing phase on circuits that outputs a "data structure" of length $\ell_{\mathrm{DS}} = N^c$.

We also fix the hardness amplification procedure $\mathsf{Amp} : \{0,1\}^{W_{\mathsf{proof}}} \to \{0,1\}^{\ell'}$ described in Theorem 3.3.5 that amplifies hardness δ to hardness $(1/2 - \varepsilon)$. Here, $\ell' := W_{\mathsf{proof}}^{O(\log(1/\varepsilon)/\delta)} = \ell^{O(300q/c_m)}$. We set the parameter c_m such that $\ell' \le \ell$. By Lemma 3.3.7, we may assume that $\ell' = \ell$ without loss of generality. Let $(\mathsf{idx}, \mathsf{coeff})$ be the family of linear sum circuit described in Theorem 3.3.5, then $(\mathsf{idx}, \mathsf{coeff})$ has the following parameters:

$$\begin{aligned}
\text{advice complexity} &= a := O(\log^2 W_{\mathsf{proof}}/(\varepsilon\delta)^2) = O(\log^2 \ell/\varepsilon^2),\\
\text{fan-in} &= A := O(\log W_{\mathsf{proof}}/(\varepsilon\delta)^2) = O(\log \ell/\varepsilon^2),\\
\text{coefficient sum} &= U := O(1/\varepsilon),\\
\text{locality} &= l := \log \ell.
\end{aligned}$$

We say an input x *has an easy witness* if there is a proof matrix π such that:

(completeness) for every $\mathsf{seed} \in \{0,1\}^r$, $\mathsf{VPCPP}^{\mathsf{Encode}(x)\circ\pi}(\mathsf{seed})$ accepts;

(*approximate* easiness) for every row π_i of π, there exists an input $w_i \in \{0,1\}^n$ and an advice $\alpha_i \in \{0,1\}^a$ such that the decoding of $C(w_i)$ with advice α_i is δ-close to π_i with respect to ℓ_1-norm. (Recall that $\mathsf{dec}_\alpha(x)$ denotes the decoding of x under advice α.) In particular:

1. for every $j \in [W_{\mathsf{proof}}]$, $(\mathsf{dec}_{\alpha_i}(C(w_i)))_j \in [0,1]$;
2. $\|\mathsf{dec}_{\alpha_i}(C(w_i)) - \pi_i\|_1 \le \delta$.

Recall that $P = (r_{\mathsf{col}})^{\log(1/\varepsilon)}$. By our hypothesis, there is an algorithm that takes as input a list of $N = 2^{r_{\mathsf{col}}}$ $\mathsf{AND}_{4q} \circ \mathscr{C}$ circuits $\{C_i\}$ and a list of N inputs $\{x_j\}$, runs in deterministic

$T^{\mathrm{alg}} := 2^{2r_{\mathsf{col}}}/P^{c_u}$ time, and estimates $\mathbb{E}_{i,j}[C_i(x_j)]$ within an additive error of $\eta := \varepsilon^{c_u} \le U^{-10q}$. This algorithm is allowed to have a P^{NP} preprocessing phase on the circuits $\{C_i\}$ which outputs a "data structure" of length $\ell_{\mathrm{DS}} = N^c$.

3.5.3 Guessing and Verifying the PCPP

On input $x \in \{0,1\}^{n_{\mathsf{hard}}}$, we guess $\hat{H}_{\mathsf{proof}}$ strings $w_1, w_2, \ldots, w_{\hat{H}_{\mathsf{proof}}} \in \{0,1\}^n$ as well as $\hat{H}_{\mathsf{proof}}$ advice strings $\alpha_1, \alpha_2, \ldots, \alpha_{\hat{H}_{\mathsf{proof}}} \in \{0,1\}^a$. Let $\pi_i^{\mathsf{Real}} := \mathsf{dec}_{\alpha_i}(C(w_i))$, and π_i^{Bool} be the Boolean string that is closest to π_i^{Real}. We will think of the matrix π^{Bool} as the PCPP proof, although our algorithm M^{PCPP} will operate on π^{Real}.

Therefore, before we proceed, we need to verify that π^{Real} and π^{Bool} are "close", so that it does no harm to operate on π^{Real} even if the correct PCPP proof should be π^{Bool}. This verification phase also occurs in previous works proving lower bounds against linear combinations of circuits [Wil18b, CW19b, CR22, CLW20]. Like in previous work, we only provide an "approximate" verification algorithm: if the input has an easy witness, then the PCPP proof π^{Real} corresponding to this easy witness is accepted; on the other hand, we reject every π^{Real} that is "too far" from Boolean.

In what follows, denote

$$\begin{aligned}(\mathsf{itype}[1], \mathsf{itype}[2], \ldots, \mathsf{itype}[q]) &\leftarrow V_{\mathsf{type}}(\mathsf{seed.shared}),\\ (\mathsf{irow}[1], \mathsf{irow}[2], \ldots, \mathsf{irow}[q]) &\leftarrow V_{\mathsf{row}}(\mathsf{seed.shared}, \mathsf{seed.row}), \qquad \text{and}\\ (\mathsf{icol}[1], \mathsf{icol}[2], \ldots, \mathsf{icol}[q]) &\leftarrow V_{\mathsf{col}}(\mathsf{seed.shared}, \mathsf{seed.col}).\end{aligned}$$

For each $\mathsf{seed.shared} \in \{0,1\}^{r_{\mathsf{shared}}}$ and each $\iota \in [q]$ such that $\mathsf{itype}[\iota] = \mathsf{proof}$, we define the following functions:

$$\begin{aligned}f^{\mathsf{Bool}}_{\mathsf{seed.shared},\iota}(\mathsf{seed.row}, \mathsf{seed.col}) &= \pi^{\mathsf{Bool}}_{\mathsf{irow}[\iota],\mathsf{icol}[\iota]} \quad \text{and}\\ f^{\mathsf{Real}}_{\mathsf{seed.shared},\iota}(\mathsf{seed.row}, \mathsf{seed.col}) &= \pi^{\mathsf{Real}}_{\mathsf{irow}[\iota],\mathsf{icol}[\iota]}.\end{aligned}$$

We will talk about the ℓ_d-norms of the above functions. For example, let $d \in \mathbb{N}$ be a constant, then

$$\|f^{\mathsf{Bool}}_{\mathsf{seed.shared},\iota} - f^{\mathsf{Real}}_{\mathsf{seed.shared},\iota}\|_d = \mathop{\mathbb{E}}_{\substack{\mathsf{seed.row}\leftarrow\{0,1\}^{r_{\mathsf{row}}}\\ \mathsf{seed.col}\leftarrow\{0,1\}^{r_{\mathsf{col}}}}} \left[\left|\pi^{\mathsf{Bool}}_{\mathsf{irow}[\iota],\mathsf{icol}[\iota]} - \pi^{\mathsf{Real}}_{\mathsf{irow}[\iota],\mathsf{icol}[\iota]}\right|^d\right]^{1/d}.$$

Lemma 3.5.4. *Let $\mathscr{C}$ be a typical circuit class and $d \ge 2$ be an even number. Suppose there is an algorithm that takes as inputs a list of $2^{r_{\mathsf{col}}}$ $\mathsf{AND}_{2d} \circ \mathscr{C}$ circuits $\{C_i\}$ and a list of $2^{r_{\mathsf{col}}}$ inputs $\{x_j\}$, runs in deterministic T^{alg} time, and estimates the following quantity with additive error η:*

$$\Pr_{i,j\leftarrow[2^{r_{\mathsf{col}}}]}[C_i(x_j)].$$

Then there is an algorithm that takes $(w_1, w_2, \ldots, w_{\hat{H}_{\mathsf{proof}}})$, $(\alpha_1, \alpha_2, \ldots, \alpha_{\hat{H}_{\mathsf{proof}}})$, and the circuit C as inputs, runs in deterministic $O((3A)^{2d}T^{\mathrm{alg}}) \cdot \left(2^{2dl+r_{\mathsf{shared}}} + T\log^{O(m)} T/2^{2r_{\mathsf{col}}}\right)$ time, and satisfies the following:

(Completeness) *If for every $i \in [\hat{H}_{\mathsf{proof}}]$, it holds that (1) for every $j \in [W_{\mathsf{proof}}]$, $\pi^{\mathsf{Real}}_{i,j} \in [0,1]$; (2) $\|\pi^{\mathsf{Real}}_i - \pi^{\mathsf{Bool}}_i\|_1 \le \delta$, then the algorithm accepts.*

(Soundness) *If the algorithm accepts, then it holds that*

1. *for every $\mathsf{seed.shared} \in \{0,1\}^{r_{\mathsf{shared}}}$ and $\iota \in [q]$, $\|f^{\mathsf{Real}}_{\mathsf{seed.shared},\iota}\|^d_d \le 1 + 2\eta \cdot U^d$;*
2. *$\mathbb{E}_{i \leftarrow [\hat{H}_{\mathsf{proof}}], j \leftarrow [W_{\mathsf{proof}}]} \left[|\pi^{\mathsf{Real}}_{i,j} - \pi^{\mathsf{Bool}}_{i,j}|^d \right] \le 4^d \cdot \delta + 2^{d+1}\eta(2U+1)^{2d}$.*

Moreover, if the algorithm in the hypothesis requires a P^{NP} preprocessing phase on the circuits that outputs a "data structure" of length ℓ_{DS}, then the algorithm in the conclusion requires a P^{NP} preprocessing phase on C that outputs a "data structure" of length $(3A)^{2d} 2^{2dl + r_{\mathsf{shared}}} \cdot \ell_{\mathrm{DS}}$.

Proof. Fix $\mathsf{seed.shared}$ and ι, we first estimate

$$\|f^{\mathsf{Real}}_{\mathsf{seed.shared},\iota}\|^d_d := \mathop{\mathbb{E}}_{\mathsf{seed.row},\mathsf{seed.col}} \left[|\pi^{\mathsf{Real}}_{\mathsf{irow}[\iota],\mathsf{icol}[\iota]}|^d \right].$$

Recall that

$$\pi^{\mathsf{Real}}_{i,j} = \sum_{k \in [A]} \mathsf{coeff}_k(\alpha_i) \cdot C_{\mathsf{idx}_k(\alpha_i, j)}(w_i).$$

Therefore, we can build a $\mathsf{Prod}_d \circ \mathsf{Sum} \circ \mathscr{C}$ circuit $C_{\mathsf{norm}} := C_{\mathsf{norm}}(\mathsf{seed.shared}, \iota)$ as follows.

Circuit C_{norm}

(Inputs) The input consists of (w, α) with the intended meaning that $w = w_{\mathsf{irow}[\iota]}$ and $\alpha = \alpha_{\mathsf{irow}[\iota]}$.

(Bottom circuits) The bottom circuit is exactly C (taking input w). Thus, there are ℓ output gates of $\mathscr{C}$ circuits with the i-th one being precisely the i-th output gate of C.

(Intermediate linear sum gates) There are $2^{r_{\mathsf{col}}}$ intermediate linear sum gates. For each $\mathsf{seed.col}$,

$$\mathsf{Sum}_{\mathsf{seed.col}}(w, \alpha) = \sum_{k \in [A]} \mathsf{coeff}_k(\alpha) \cdot C_{\mathsf{idx}_k(\alpha, \mathsf{icol}[\iota])}(w).$$

(Output product gates) There are $2^{r_{\mathsf{col}}}$ product gates. For each $\mathsf{seed.col}$, the $\mathsf{seed.col}$-th output gate is simply

$$(C_{\mathsf{norm}})_{\mathsf{seed.col}}(w, \alpha) = \left(\mathsf{Sum}_{\mathsf{seed.col}}(w, \alpha)\right)^d.$$

Recall that this circuit C_{norm} has parameters as follows:

- the number of gates in each layer: $\ell_{\mathscr{C}} = \ell$, $\ell_{\mathsf{Sum}} = 2^{r_{\mathsf{col}}}$, $\ell_{\mathsf{Prod}} = 2^{r_{\mathsf{col}}}$;
- the fan-in of the top Prod gates d;
- the fan-in A, coefficient sum U, and locality l of the linear sum layer.

We invoke Theorem 3.5.3 on the circuit C_{norm} and $M' := 2^{r_{\mathsf{row}}}$ inputs $\{(w_{\mathsf{irow}[\iota]}, \alpha_{\mathsf{irow}[\iota]})\}_{\mathsf{seed.row}}$. We obtain an estimation $\mathsf{EST}_{\mathsf{norm}} = \mathsf{EST}_{\mathsf{norm}}(\mathsf{seed.shared}, \iota)$ where

$$\left| \mathsf{EST}_{\mathsf{norm}} - \|f^{\mathsf{Real}}_{\mathsf{seed.shared},\iota}\|^d_d \right| \le \eta \cdot U^d.$$

If $\mathsf{EST}_{\mathsf{norm}} > 1 + \eta \cdot U^d$, then we reject the input. Otherwise, we proceed to verify that π^{Real} and π^{Bool} are close. Consider the polynomial $P(z) := z^d(1-z)^d$, which intuitively measures how close z is to Boolean. We will estimate

$$\mathop{\mathbb{E}}_{i \leftarrow [\hat{H}_{\mathsf{proof}}], j \leftarrow [W_{\mathsf{proof}}]} \left[P(\pi^{\mathsf{Real}}_{i,j}) \right]. \tag{3.9}$$

Similarly, we estimate (3.9) by building a $\mathsf{Prod}_{2d} \circ \mathsf{Sum} \circ \mathscr{C}$ circuit C_{diff}.

Circuit C_{diff}

(Inputs and bottom circuits) The inputs and bottom circuits of C_{diff} are exactly the same as C_{norm}.

(Intermediate linear sum gates) There are $2W_{\mathsf{proof}}$ intermediate linear sum gates. Let $j \in [W_{\mathsf{proof}}]$, then the $2j$-th linear sum gate computes $(\pi^{\mathsf{Real}})_j$, and the $(2j+1)$-th one computes $1 - (\pi^{\mathsf{Real}})_j$. That is,

$$\mathsf{Sum}_{2j}(w, \alpha) = \sum_{k \in [A]} \mathsf{coeff}_k(\alpha) \cdot C_{\mathsf{idx}_k(\alpha, j)}(w); \quad \mathsf{Sum}_{2j+1}(w, \alpha) = 1 - \mathsf{Sum}_{2j}(w, \alpha).$$

Implementation of the linear sum layer: Since we did not allow $\mathsf{coeff}_k(\alpha)$ to depend on i (the output index in $[2W_{\mathsf{proof}}]$), we need to be careful when implementing the linear sum layer. The fan-in of this layer will be $2A+1$ (instead of A). We identify $[2A+1]$ with the disjoint union of $[A] \times \{0,1\}$ and $\{\star\}$ (where $\star$ denotes the constant term 1 in $\mathsf{Sum}_{2j+1}(w, \alpha)$). Let idx' and coeff' be the idx and coeff functions of the intermediate linear sum gates of C_{diff}:

(Function $\mathsf{idx}'_k(\alpha, i)$) We write $i = 2j + b$ where $j \in [W_{\mathsf{proof}}]$ and $b \in \{0,1\}$. If $k = (k', b') \in [A] \times \{0,1\}$, then $\mathsf{idx}'_k(\alpha, i)$ returns $\mathsf{idx}_{k'}(\alpha, j)$ if $b = b'$ and returns ZERO if $b \neq b'$. If $k = \star$ then $\mathsf{idx}'_k(\alpha, i)$ returns ZERO if $b = 0$ and returns ONE if $b = 1$.

(Function $\mathsf{coeff}'_k(\alpha)$) If $k = (k', b') \in [A] \times \{0,1\}$, then $\mathsf{coeff}'_k(\alpha) = (-1)^{b'} \cdot \mathsf{coeff}_{k'}(\alpha)$. If $k = \star$ then $\mathsf{coeff}'_k(\alpha) = 1$.

The locality of $(\mathsf{idx}', \mathsf{coeff}')$ is still l. The coefficient sum becomes $2U + 1$.

(Output product gates) There are W_{proof} product gates. For each $j \in [W_{\mathsf{proof}}]$, the j-th output gate is

$$C_{\mathsf{diff}}(w, \alpha) = (\mathsf{Sum}_{2j}(w, \alpha) \cdot \mathsf{Sum}_{2j+1}(w, \alpha))^d.$$

The parameters of the circuit C_{diff} are as follows:

- the number of gates in each layer: $\ell_{\mathscr{C}} = \ell$, $\ell_{\mathsf{Sum}} = 2W_{\mathsf{proof}}$, $\ell_{\mathsf{Prod}} = W_{\mathsf{proof}}$;
- the fan-in of the top Prod gates $2d$;
- the fan-in $2A+1$, coefficient sum $2U+1$, and locality l of the linear sum layer.

We invoke Theorem 3.5.3 on the circuit C_{diff} and the $M' := \hat{H}_{\mathsf{proof}}$ inputs $\{(w_i, \alpha_i)\}_{i \in [\hat{H}_{\mathsf{proof}}]}$, and obtain an estimation $\mathsf{EST}_{\mathsf{diff}}$ where

$$|\mathsf{EST}_{\mathsf{diff}} - (3.9)| \leq \eta \cdot (2U+1)^{2d}.$$

We then accept if and only if $\mathsf{EST}_{\mathsf{diff}} \leq 2^d \cdot \delta + \eta(2U+1)^{2d}$.

Complexity. Our algorithm calls the algorithm in Theorem 3.5.3 as a subroutine on the circuits C_{norm} and C_{diff}. It takes $A^d(2^{dl} + 2^{r_{\mathsf{row}}}/2^{r_{\mathsf{col}}}) \cdot O(T^{\mathrm{alg}})$ time to process each C_{norm}. Similarly, it takes $(2A+1)^{2d}(2^{2dl} + \hat{H}_{\mathsf{proof}}/2^{r_{\mathsf{col}}}) \cdot (W_{\mathsf{proof}}/2^{r_{\mathsf{col}}}) \cdot O(T^{\mathrm{alg}})$ time to process C_{diff}. It follows that our algorithm runs in deterministic time

$$\begin{aligned} &2^{r_{\mathsf{shared}}} \cdot A^d(2^{dl} + 2^{r_{\mathsf{row}}}/2^{r_{\mathsf{col}}}) \cdot O(T^{\mathrm{alg}}) + (2A+1)^{2d}(2^{2dl} + \hat{H}_{\mathsf{proof}}/2^{r_{\mathsf{col}}}) \cdot (W_{\mathsf{proof}}/2^{r_{\mathsf{col}}}) \cdot O(T^{\mathrm{alg}}) \\ &= O((3A)^{2d} T^{\mathrm{alg}}) \cdot \left(2^{dl + r_{\mathsf{shared}}} + 2^{r - 2r_{\mathsf{col}}} + 2^{2dl} \cdot W_{\mathsf{proof}}/2^{r_{\mathsf{col}}} + \hat{H}_{\mathsf{proof}} \cdot W_{\mathsf{proof}}/2^{2r_{\mathsf{col}}}\right) \end{aligned}$$

$$= O((3A)^{2d} T^{\mathrm{alg}}) \cdot \left(2^{2dl + r_{\mathsf{shared}}} + T \log^{O(m)} T / 2^{2r_{\mathsf{col}}}\right).$$

(Recall that since $\log(W_{\mathsf{proof}}/2^{r_{\mathsf{col}}}) = (5/m)\log T$, $r_{\mathsf{shared}} \geq (10/m)\log T$, we have $W_{\mathsf{proof}}/2^{r_{\mathsf{col}}} \leq 2^{r_{\mathsf{shared}}}$. Also, from $r = \log T + O(m \log\log T)$ we know that $2^{r-2r_{\mathsf{col}}} \leq T \log^{O(m)} T / 2^{2r_{\mathsf{col}}}$.)

Now it suffices to prove the completeness and soundness requirements. Before that, we need the following fact regarding the polynomial P. Let $z \in \mathbb{R}$, $d_{\mathsf{bin}}(z)$ be the distance between z to the closest Boolean value; namely $d_{\mathsf{bin}}(z) := \min\{|z|, |1-z|\}$.

Fact 3.5.5. *For every $z \in \mathbb{R}$, $d_{\mathsf{bin}}(z)^d \cdot 2^{-d} \leq P(z) \leq d_{\mathsf{bin}}(z)^d \cdot (1 + d_{\mathsf{bin}}(z))^d$.* ◇

Completeness. Suppose that for every $i \in [\hat{H}_{\mathsf{proof}}]$ and $j \in [W_{\mathsf{proof}}]$, $\pi^{\mathsf{Real}}_{i,j} \in [0,1]$. Then for every $\iota \in [q]$ and $\mathsf{seed.shared} \in \{0,1\}^{r_{\mathsf{shared}}}$, $\|f^{\mathsf{Real}}_{\mathsf{seed.shared},\iota}\|_d^d \leq 1$, and thus $\mathsf{EST}_{\mathsf{norm}} \leq 1 + \eta \cdot U^d$.

Suppose in addition that for every $i \in [\hat{H}_{\mathsf{proof}}]$, $\|\pi^{\mathsf{Real}}_i - \pi^{\mathsf{Bool}}_i\|_1 \leq \delta$. Then:

$$\begin{aligned}(3.9) &\leq \mathop{\mathbb{E}}_{i,j}\left[d_{\mathsf{bin}}(\pi^{\mathsf{Real}}_{i,j})^d \cdot (1 + d_{\mathsf{bin}}(\pi^{\mathsf{Real}}_{i,j}))^d\right] \\ &\leq 2^d \cdot \mathop{\mathbb{E}}_{i,j}\left[d_{\mathsf{bin}}(\pi^{\mathsf{Real}}_{i,j})^d\right] \\ &\leq 2^d \cdot \delta.\end{aligned}$$

Therefore, $\mathsf{EST}_{\mathsf{diff}} \leq 2^d \cdot \delta + \eta(2U+1)^{2d}$ and our algorithm accepts.

Soundness. Suppose our algorithm accepts.

1. For every $\mathsf{seed.shared}$ and ι, we have $\mathsf{EST}_{\mathsf{norm}} \leq 1 + \eta \cdot U^d$ and thus $\|f^{\mathsf{Real}}_{\mathsf{seed.shared},\iota}\|_d^d \leq 1 + 2\eta \cdot U^d$.
2. We have $(3.9) \leq 2^d \cdot \delta + 2\eta(2U+1)^{2d}$ and

$$\begin{aligned}\mathop{\mathbb{E}}_{i \leftarrow [\hat{H}_{\mathsf{proof}}], j \leftarrow [W_{\mathsf{proof}}]}\left[|\pi^{\mathsf{Real}}_{i,j} - \pi^{\mathsf{Bool}}_{i,j}|^d\right] &= \mathop{\mathbb{E}}_{i,j}\left[d_{\mathsf{bin}}(\pi^{\mathsf{Real}}_{i,j})^d\right] \\ &\leq 2^d \cdot (3.9) \qquad \text{(Fact 3.5.5)} \\ &\leq 4^d \cdot \delta + 2^{d+1}\eta(2U+1)^{2d}.\end{aligned}$$

Preprocessing. Suppose the algorithm in the hypothesis requires a P^{NP} preprocessing phase on the circuits that outputs a "data structure" of length ℓ_{DS}. By Theorem 3.5.3, the P^{NP} preprocessing phase on each $C_{\mathsf{norm}}(\mathsf{seed.shared}, \iota)$ outputs a "data structure" of length $O(A^d 2^{dl}) \cdot \ell_{\mathrm{DS}}$ and the P^{NP} preprocessing phase on C_{diff} outputs a "data structure" of length $O((2A+1)^{2d} 2^{2dl}) \cdot (W_{\mathsf{proof}}/2^{r_{\mathsf{col}}}) \cdot \ell_{\mathrm{DS}}$. Since C_{norm} and C_{diff} only depend on C (not w_i or α_i), the preprocessing phase of our algorithm also only depends on C. The total length of these "data structures" is

$$O(A^d 2^{dl} 2^{r_{\mathsf{shared}}} + (2A+1)^{2d} 2^{2dl} (W_{\mathsf{proof}}/2^{r_{\mathsf{col}}}))\ell_{\mathrm{DS}} \leq (3A)^{2d} 2^{2dl + r_{\mathsf{shared}}} \cdot \ell_{\mathrm{DS}}.$$ □

We substitute $d := 2q$ in the above lemma. If x has an easy witness, then there is some $(w_1, w_2, \ldots, w_{\hat{H}_{\mathsf{proof}}})$ and $(\alpha_1, \alpha_2, \ldots, \alpha_{\hat{H}_{\mathsf{proof}}})$ that passes the test; on the other hand, if the test is passed, then both soundness properties in Lemma 3.5.4 hold:

1. for every $\mathsf{seed.shared}$ and ι, $\|f^{\mathsf{Real}}_{\mathsf{seed.shared},\iota}\|_{2q}^{2q} \le 1 + 2\eta \cdot U^{2q}$;
2. $\mathbb{E}_{i\leftarrow[H_{\mathsf{proof}}],j\leftarrow[W_{\mathsf{proof}}]}\left[|\pi^{\mathsf{Real}}_{i,j} - \pi^{\mathsf{Bool}}_{i,j}|^{2q}\right] \le 16^q \cdot \delta + 128^q \eta U^{4q}$.

Moreover, the output length of the P^{NP} preprocessing phase is at most $(3A)^{4q}2^{4ql+r_{\mathsf{shared}}}\ell_{\mathrm{DS}}$.

3.5.4 Estimating the Acceptance Probability

After checking that the PCPP proof is "close to Boolean", the next step is to use it to speed up L^{hard}. We estimate

$$p_{\mathsf{acc}} := \Pr_{\mathsf{seed}\leftarrow\{0,1\}^r}\left[\mathsf{VPCPP}^{\mathsf{Encode}(x)\circ\pi^{\mathsf{Bool}}}(\mathsf{seed}) \text{ accepts}\right].$$

(Indeed, it suffices to distinguish between the case that $p_{\mathsf{acc}} \ge 5/6$ and the case that $p_{\mathsf{acc}} < 1/2$ as we will explain later.)

We enumerate $\mathsf{seed.shared}$. After fixing $\mathsf{seed.shared}$, each $\mathsf{itype}[\iota]$ is completely fixed, each $\mathsf{irow}[\iota]$ only depends on $\mathsf{seed.row}$, and each $\mathsf{icol}[\iota]$ only depends on $\mathsf{seed.col}$. We now need to estimate

$$p_{\mathsf{acc}}(\mathsf{seed.shared}) := \Pr_{\substack{\mathsf{seed.row}\leftarrow\{0,1\}^{r_{\mathsf{row}}}\\ \mathsf{seed.col}\leftarrow\{0,1\}^{r_{\mathsf{col}}}}}\left[\mathsf{VPCPP}^{\mathsf{Encode}(x)\circ\pi^{\mathsf{Bool}}}(\mathsf{seed}) \text{ accepts}\right].$$

If we also fix $\mathsf{seed.row}$, then we know the q rows of the input matrix $\mathsf{Encode}(x)$ and the proof matrix π that could influence the PCPP verifier, denoted as $\mathsf{row}^{\mathsf{Bool}}_1, \mathsf{row}^{\mathsf{Bool}}_2, \ldots, \mathsf{row}^{\mathsf{Bool}}_q$. In particular, letting $\tilde{x}_i$ denote the i-th row of the matrix $\mathsf{Encode}(x)$, then for each $\iota \in [q]$:

$$\mathsf{row}^{\mathsf{Bool}}_\iota = \begin{cases} \tilde{x}_{\mathsf{irow}[\iota]} & \text{if } \mathsf{itype}[\iota] = \mathsf{input},\\ \pi^{\mathsf{Bool}}_{\mathsf{irow}[\iota]} & \text{if } \mathsf{itype}[\iota] = \mathsf{proof}. \end{cases}$$

We also let $pc_1, pc_2, \ldots, pc_q \leftarrow V_{\mathsf{pc}}(\mathsf{seed.shared})$ be the parity-check functions of the PCPP verifier, where each $pc_\iota : \{0,1\}^{r_{\mathsf{row}}+r_{\mathsf{col}}} \to \{0,1\}$. In particular, let pc^{row}_ι (resp. pc^{col}_ι) denote the contribution of $\mathsf{seed.row}$ (resp. $\mathsf{seed.col}$) to pc_ι, i.e.,

$$pc^{\mathsf{row}}_\iota(\mathsf{seed.row}) := pc_\iota(\mathsf{seed.row}, 0^{r_{\mathsf{col}}}), \quad \text{and} \quad pc^{\mathsf{col}}_\iota(\mathsf{seed.col}) := pc_\iota(0^{r_{\mathsf{row}}}, \mathsf{seed.col}).$$

Then $pc_\iota(\mathsf{seed.row}, \mathsf{seed.col}) = pc^{\mathsf{row}}_\iota(\mathsf{seed.row}) \oplus pc^{\mathsf{col}}_\iota(\mathsf{seed.col})$. For simplicity, we omit $\mathsf{seed.row}$ and $\mathsf{seed.col}$ when they are clear from the context.

Let VDec be the decision predicate of the PCPP verifier; note that as $\mathsf{seed.shared}$ is fixed, $\mathsf{VDec} \leftarrow V_{\mathsf{dec}}(\mathsf{seed.shared})$ is also fixed. The input of VDec includes the answers to the q queries and the parity-check bits $pc_1, \ldots, pc_q$. On $\mathsf{seed.row}$ and $\mathsf{seed.col}$, the PCPP verifier outputs

$$\mathsf{VDec}\Big((\mathsf{row}^{\mathsf{Bool}}_1)_{\mathsf{icol}[1]}, (\mathsf{row}^{\mathsf{Bool}}_2)_{\mathsf{icol}[2]}, \ldots, (\mathsf{row}^{\mathsf{Bool}}_q)_{\mathsf{icol}[q]}, pc_1, pc_2, \ldots, pc_q\Big).$$

As every Boolean function over $2q$ bits can be written as a degree-$2q$ polynomial over the

reals, we write

$$\mathsf{VDec}(a_1, a_2, \ldots, a_q, pc_1, pc_2, \ldots, pc_q) = \sum_{S\subseteq[q], S'\subseteq[q]} \theta_{S,S'} \left(\prod_{\iota\in S} a_\iota\right) \cdot \left(\prod_{\iota\in S'} pc_\iota\right),$$

where $\theta_{S,S'} \in [-2^{2q}, 2^{2q}]$. Now, define

$$p_{\mathsf{acc}}(\mathsf{seed.shared}, S, S') := \mathop{\mathbb{E}}_{\substack{\mathsf{seed.row}\leftarrow\{0,1\}^{r_{\mathsf{row}}} \\ \mathsf{seed.col}\leftarrow\{0,1\}^{r_{\mathsf{col}}}}} \left[\prod_{\iota\in S} (\mathsf{row}_\iota^{\mathsf{Bool}})_{\mathsf{icol}[\iota]} \cdot \prod_{\iota\in S'} pc_\iota\right].$$

We have

$$p_{\mathsf{acc}}(\mathsf{seed.shared}) = \sum_{S\subseteq[q], S'\subseteq[q]} \theta_{S,S'} p_{\mathsf{acc}}(\mathsf{seed.shared}, S, S'),$$

thus it suffices to estimate each $p_{\mathsf{acc}}(\mathsf{seed.shared}, S, S')$.

Fix S and S'. Since we only have access to a real proof matrix π^{Real} instead of a Boolean proof matrix, we use the following number as an estimation of $p_{\mathsf{acc}}(\mathsf{seed.shared}, S, S')$, with the only difference being π_i^{Bool} being replaced by π_i^{Real}:

$$p_{\mathsf{acc}}^{\mathsf{Real}}(\mathsf{seed.shared}, S, S') = \mathop{\mathbb{E}}_{\substack{\mathsf{seed.row}\leftarrow\{0,1\}^{r_{\mathsf{row}}} \\ \mathsf{seed.col}\leftarrow\{0,1\}^{r_{\mathsf{col}}}}} \left[\prod_{\iota\in S} (\mathsf{row}_\iota^{\mathsf{Real}})_{\mathsf{icol}[\iota]} \cdot \prod_{\iota\in S'} pc_\iota\right],$$

where

$$\mathsf{row}_\iota^{\mathsf{Real}} = \begin{cases} \tilde{x}_{\mathsf{irow}[\iota]} & \text{if } \mathsf{itype}[\iota] = \mathsf{input}, \\ \pi^{\mathsf{Real}}_{\mathsf{irow}[\iota]} & \text{if } \mathsf{itype}[\iota] = \mathsf{proof}. \end{cases}$$

The following claim bounds the accuracy of the estimation given the ℓ_d-distance between the functions $f^{\mathsf{Bool}}_{\mathsf{seed.shared},\iota}$ and $f^{\mathsf{Real}}_{\mathsf{seed.shared},\iota}$. The proof is deferred to Section 3.5.7.

Claim 3.5.6. *For every $S, S' \subseteq [q]$,*

$$|p_{\mathsf{acc}}(\mathsf{seed.shared}, S, S') - p_{\mathsf{acc}}^{\mathsf{Real}}(\mathsf{seed.shared}, S, S')| \le (1 + \delta_{\mathsf{seed.shared}})^{2q-1} \cdot \delta_{\mathsf{seed.shared}}.$$

Here,

$$\begin{aligned} \delta_{\mathsf{seed.shared}} &:= \sum_{\iota:\mathsf{itype}[\iota]=\mathsf{proof}} \|f^{\mathsf{Bool}}_{\mathsf{seed.shared},\iota} - f^{\mathsf{Real}}_{\mathsf{seed.shared},\iota}\|_{2q} \\ (\text{recall}) &= \sum_{\iota:\mathsf{itype}[\iota]=\mathsf{proof}} \mathop{\mathbb{E}}_{\substack{\mathsf{seed.row}\leftarrow\{0,1\}^{r_{\mathsf{row}}} \\ \mathsf{seed.col}\leftarrow\{0,1\}^{r_{\mathsf{col}}}}} \left[\left|\pi^{\mathsf{Bool}}_{\mathsf{irow}[\iota],\mathsf{icol}[\iota]} - \pi^{\mathsf{Real}}_{\mathsf{irow}[\iota],\mathsf{icol}[\iota]}\right|^{2q}\right]^{1/(2q)}. \end{aligned}$$

Now we fix S, S' and estimate $p_{\mathsf{acc}}^{\mathsf{Real}}(\mathsf{seed.shared}, S, S')$. Let $d_S := |S|, d_{S'} := |S'|$, it is without loss of generality to assume that $S = \{1, 2, \ldots, d_S\}$ and $S' = \{1, 2, \ldots, d_{S'}\}$. We construct a $\mathsf{Prod}_{d_S+d_{S'}} \circ \mathsf{Sum} \circ \mathscr{C}$ circuit $C^{\mathsf{Prod}} := C^{\mathsf{Prod}}_{\mathsf{seed.shared},S,S'}$, as well as a list of inputs $(z_{\mathsf{seed.row}}, \alpha_{\mathsf{seed.row}})$, such that

$$C^{\mathsf{Prod}}_{\mathsf{seed.col}}(z_{\mathsf{seed.row}}, \alpha_{\mathsf{seed.row}}) = \prod_{\iota\in S} (\mathsf{row}_\iota^{\mathsf{Real}})_{\mathsf{icol}[\iota]} \cdot \prod_{\iota\in S'} pc_\iota. \tag{3.10}$$

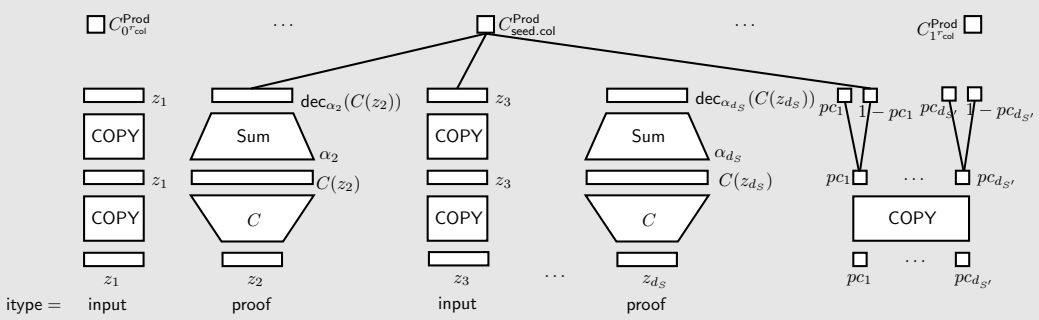


Figure 3.3: Construction of the circuit C^{Prod}. Note that for convenience, we only drew the "relevant" parts of this circuit, e.g., $C(z_i)$ when $\mathsf{itype}[i] = \mathsf{proof}$ and the copying circuit for z_i when $\mathsf{itype}[i] = \mathsf{input}$.

Construction of C^{Prod} and inputs. For each $\mathsf{seed.row}$ and each $\iota \in S$, define

$$(z_{\mathsf{seed.row}})_\iota := \begin{cases} \tilde{x}_{\mathsf{irow}[\iota]} & \text{if } \mathsf{itype}[\iota] = \mathsf{input}, \\ w_{\mathsf{irow}[\iota]} & \text{if } \mathsf{itype}[\iota] = \mathsf{proof}. \end{cases}$$

Then we concatenate each $(z_{\mathsf{seed.row}})_\iota$ and the (row-)parity-check bits to obtain

$$z_{\mathsf{seed.row}} := \Big((z_{\mathsf{seed.row}})_1, (z_{\mathsf{seed.row}})_2, \ldots, (z_{\mathsf{seed.row}})_{d_S}, pc_1^{\mathsf{row}}, pc_2^{\mathsf{row}}, \ldots, pc_{d_{S'}}^{\mathsf{row}}\Big).$$

It is easy to check that given $\mathsf{seed.row}$ (and $\mathsf{seed.shared}$), we can compute $z_{\mathsf{seed.row}}$ easily. We also define

$$\alpha_{\mathsf{seed.row}} := \big(\alpha_{\mathsf{irow}[1]}, \alpha_{\mathsf{irow}[2]}, \ldots, \alpha_{\mathsf{irow}[d_S]}\big).$$

Next, we define the circuit C^{Prod} that takes two inputs (z, α) and outputs $2^{|\mathsf{seed.col}|}$ real numbers. Fix $\mathsf{seed.col}$, we want that

$$\begin{aligned} &C^{\mathsf{Prod}}_{\mathsf{seed.col}}(z_{\mathsf{seed.row}}, \alpha_{\mathsf{seed.row}}) \\ &= \prod_{\iota \in S} (\mathsf{row}_\iota^{\mathsf{Real}})_{\mathsf{icol}[\iota]} \cdot \prod_{\iota \in S'} pc_\iota \\ &= \prod_{\iota \in S^{\mathsf{proof}}} (\mathsf{dec}_{\alpha_{\mathsf{irow}[\iota]}}(C(w_{\mathsf{irow}[\iota]})))_{\mathsf{icol}[\iota]} \cdot \prod_{\iota \in S^{\mathsf{input}}} \tilde{x}_{\mathsf{irow}[\iota],\mathsf{icol}[\iota]} \cdot \prod_{\iota \in S'} (pc_\iota^{\mathsf{row}} \oplus pc_\iota^{\mathsf{col}}), \end{aligned}$$

where

$$(\mathsf{dec}_{\alpha_{\mathsf{irow}[\iota]}}(C(w_{\mathsf{irow}[\iota]})))_{\mathsf{icol}[\iota]} = \sum_{k \in [A]} \mathsf{coeff}_k(\alpha_{\mathsf{irow}[\iota]}) \cdot C_{\mathsf{idx}_k(\alpha_{\mathsf{irow}[\iota]},\mathsf{icol}[\iota])}(w_{\mathsf{irow}[\iota]})$$

denotes the $\mathsf{icol}[\iota]$-th bit of the string obtained by decoding $C(w_{\mathsf{irow}[\iota]})$ with the advice $\alpha_{\mathsf{irow}[\iota]}$ using the decoder dec, $S^{\mathsf{proof}} := \{i \in S : \mathsf{itype}[i] = \mathsf{proof}\}$, $S^{\mathsf{input}} := \{i \in S : \mathsf{itype}[i] = \mathsf{input}\}$.

This motivates the definition of the circuit C^{Prod} (see Figure 3.3 for an illustration and Figure 3.4 for the detailed definition). The parameters of the circuit C^{Prod} are as follows.

- The number of gates: $\ell_{\mathscr{C}} = d_S(\ell + W_{\mathsf{proof}}) + d_{S'}$, $\ell_{\mathsf{Sum}} = W_{\mathsf{proof}} \cdot d_S + 2d_{S'}$, $\ell_{\mathsf{Prod}} = 2^{r_{\mathsf{col}}}$.
- The fan-in of the top Prod gates $d_S + d_{S'} \le 2q$.
- The fan-in $A' := A \cdot d_S + 2$, coefficient sum $d_S \cdot U + 2$, and locality l of the linear sum layer.

Circuit C^{Prod}

(Inputs) The input z will have the form $z = (z_1, z_2, \ldots, z_{d_S}, pc_1, pc_2, \ldots, pc_{d_{S'}})$ and the input α will have the form $\alpha = (\alpha_1, \alpha_2, \ldots, \alpha_{d_S})$. The intended meanings are $z_i = (z_{\mathsf{seed.row}})_i$, $pc_i = pc_i^{\mathsf{row}}$, and $\alpha_i = \alpha_{\mathsf{irow}[i]}$.

(Bottom circuits) We make d_S copies to C, where the i-th copy is applied to the input z_i. (The i-th copy is useful only when $\mathsf{itype}[i] = \mathsf{proof}$, but we make all d_S copies for convenience.) We also add $W_{\mathsf{proof}} \cdot d_S + d_{S'}$ gates to copy the input.

Thus, there are $\ell_{\mathscr{C}} := d_S \cdot \ell + d_S \cdot W_{\mathsf{proof}} + d_{S'}$ output gates; we identify $[\ell_{\mathscr{C}}]$ with the disjoint union of $\{1\} \times [d_S] \times [\ell]$, $\{2\} \times [d_S] \times [W_{\mathsf{proof}}]$, and $\{3\} \times [d_{S'}]$.

- For each $j \in [d_S]$ and $i \in [\ell]$, the $(1, j, i)$-th gate is $C_{(1,j,i)}(z) := C(z_j)_i$.
- For each $j \in [d_S]$ and $i \in [W_{\mathsf{proof}}]$, the $(2, j, i)$-th gate is $C_{(2,j,i)}(z) := (z_j)_i$.
- For each $j \in [d_{S'}]$, the $(3, j)$-th gate is $C_{3,j}(z) := pc_j$.

(Intermediate linear sum gates) There are $\ell_{\mathsf{Sum}} := W_{\mathsf{proof}} \cdot d_S + 2d_{S'}$ linear sum gates and we identify $[\ell_{\mathsf{Sum}}]$ with the disjoint union of $[W_{\mathsf{proof}}] \times [d_S]$ and $[d_{S'}] \times \{0, 1\}$.

Let $i \in [W_{\mathsf{proof}}]$ and $j \in [d_S]$. If $\mathsf{itype}[j] = \mathsf{proof}$, then the (i, j)-th intermediate gate is

$$\mathsf{Sum}_{(i,j)}(z, \alpha) = \sum_{k \in [A]} \mathsf{coeff}_k(\alpha_j) \cdot C_{\mathsf{idx}_k(\alpha_j, i) \cdot d_S + j}(z).$$

It is easy to verify that

$$\mathsf{Sum}_{(i,j)}(z_{\mathsf{seed.row}}, \alpha_{\mathsf{seed.row}}) = (\mathsf{dec}_{\alpha_{\mathsf{irow}[j]}}(C(w_{\mathsf{irow}[j]})))_i.$$

On the other hand, if $\mathsf{itype}[j] = \mathsf{input}$, then the (i, j)-th intermediate gate is $\mathsf{Sum}_{(i,j)}(z, \alpha) := (z_j)_i$. (If $i > W_{\mathsf{input}}$ then we simply set $\mathsf{Sum}_{(i,j)}(z, \alpha) = 0$ and this intermediate gate would not be used.) Finally, for each $i \in [d_{S'}]$, we have two intermediate gates

$$\mathsf{Sum}_{(i,0)}(z, \alpha) = pc_i, \quad \mathsf{Sum}_{(i,1)}(z, \alpha) = 1 - pc_i.$$

Implementation of the linear sum layer: The linear sum has fan-in $A' := A \cdot d_S + 2$ and we identify $[A']$ with the disjoint union of $[A] \times [d_S]$ and $\{+, -\}$. Let idx' and coeff' be the idx and coeff functions of the linear sum layer of C^{Prod}, then

(Function $\mathsf{idx}'_k(\alpha, i)$) Suppose $i = (i', j) \in [W_{\mathsf{proof}}] \times [d_S]$. If $\mathsf{itype}[j] = \mathsf{proof}$ and $k = (k', j')$ where $j = j'$, then we return $\mathsf{idx}'_k(\alpha, i) = (1, j, \mathsf{idx}_{k'}(\alpha_j, i'))$; if $\mathsf{itype}[j] = \mathsf{input}$ and $k = +$, then $\mathsf{idx}'_k(\alpha, i) = (2, j, i')$. Otherwise $\mathsf{idx}'_k(\alpha, i) = \mathsf{ZERO}$.

On the other hand, suppose $i = (j, b) \in [d_{S'}] \times \{0, 1\}$. If ($b = 0$ and $k = +$) or ($b = 1$ and $k = -$) then $\mathsf{idx}'_k(\alpha, i) = (3, j)$. If $b = 1$ and $k = +$ then $\mathsf{idx}'_k(\alpha, i) = \mathsf{ONE}$. Otherwise $\mathsf{idx}'_k(\alpha, i) = \mathsf{ZERO}$.

(Function $\mathsf{coeff}'_k(\alpha)$) If $k = +$ then $\mathsf{coeff}'_k(\alpha) = 1$; if $k = -$ then $\mathsf{coeff}'_k(\alpha) = -1$; otherwise, if $k = (k', j')$ then $\mathsf{coeff}'_k(\alpha) = \mathsf{coeff}_{k'}(\alpha_{j'})$.

The locality of $(\mathsf{idx}', \mathsf{coeff}')$ is still l. The coefficient sum becomes $d_S \cdot U + 2$.

(Output product gates) There are $2^{r_{\mathsf{col}}}$ product gates. For each $\mathsf{seed.col}$, the $\mathsf{seed.col}$-th output gate is

$$C^{\mathsf{Prod}}_{\mathsf{seed.col}}(z, \alpha) = \prod_{i \in S} \mathsf{Sum}_{\mathsf{icol}[i] \cdot d_S + i}(z, \alpha) \cdot \prod_{i \in S'} \mathsf{Sum}_{W_{\mathsf{proof}} \cdot d_S + 2i + pc_i^{\mathsf{col}}}(z, \alpha).$$

Figure 3.4: Detailed definition of C^{Prod}.

Given the above construction, it is easy to check that (3.10) holds for every $\mathsf{seed.row}$ and $\mathsf{seed.col}$. We can see that

$$p^{\mathsf{Real}}_{\mathsf{acc}}(\mathsf{seed.shared}, S, S') = \mathop{\mathbb{E}}_{\substack{\mathsf{seed.row}\leftarrow\{0,1\}^{r_{\mathsf{row}}}\\ \mathsf{seed.col}\leftarrow\{0,1\}^{r_{\mathsf{col}}}}}\left[C^{\mathsf{Prod}}_{\mathsf{seed.col}}(z_{\mathsf{seed.row}}, \alpha_{\mathsf{seed.row}})\right].$$

Since $d_S \le q$ and $d_{S'} \le q$, by Theorem 3.5.3, we can estimate $p^{\mathsf{Real}}_{\mathsf{acc}}(\mathsf{seed.shared}, S, S')$ with additive error $\eta \cdot (qU+2)^{2q}$ in deterministic $O((qA)^{2q}(2^{2ql} + 2^{r_{\mathsf{row}}}/N) \cdot T^{\mathrm{alg}})$ time.

Analysis. First, the verification step takes

$$\begin{aligned}
&O((3A)^{4q}T^{\mathrm{alg}}) \cdot \left(2^{4ql+r_{\mathsf{shared}}} + T\log^{O(m)} T/2^{2r_{\mathsf{col}}}\right)\\
\le\ &O((3A)^{4q}) \cdot (T\log^{O(m)} T)/(r_{\mathsf{col}})^{c_u \log(1/\varepsilon)}\\
\le\ &T(\log T)^{O(m)-c_u\log(1/\varepsilon)/2}
\end{aligned}$$

time, which is at most $T/\log^{c_{\mathsf{hard}}} T$ if c_u is a large enough constant.

Our algorithm estimates $p^{\mathsf{Real}}_{\mathsf{acc}}(\mathsf{seed.shared}, S, S')$. By Claim 3.5.6, the same algorithm estimates $p_{\mathsf{acc}}(\mathsf{seed.shared}, S, S')$ within an additive error of $\eta(qU+2)^{2q} + \delta'_{\mathsf{seed.shared}}$, where

$$\delta'_{\mathsf{seed.shared}} := (1+\delta_{\mathsf{seed.shared}})^{2q-1} \cdot \delta_{\mathsf{seed.shared}}.$$

Running this algorithm for every possible (S, S'), we obtain an algorithm that runs in deterministic $(O(qA))^{2q}(2^{2ql} + 2^{r_{\mathsf{row}}}/M) \cdot T^{\mathrm{alg}}$ time and estimates $p_{\mathsf{acc}}(\mathsf{seed.shared})$ within an additive error of

$$\begin{aligned}
\le \bar{\delta}_{\mathsf{seed.shared}} &:= 4^q \cdot \sum_{S,S'}(\eta \cdot (qU+2)^{2q} + \delta'_{\mathsf{seed.shared}})\\
&\le \eta \cdot (4qU+8)^{2q} + 16^q \cdot \delta'_{\mathsf{seed.shared}}.
\end{aligned}$$

Finally, running this algorithm for every $\mathsf{seed.shared} \in \{0,1\}^{r_{\mathsf{shared}}}$, we obtain an algorithm that runs in deterministic

$$\begin{aligned}
&O(qA)^{2q}(2^{2ql} + 2^{r_{\mathsf{row}}}/N) \cdot 2^{r_{\mathsf{shared}}} \cdot O(T^{\mathrm{alg}})\\
\le\ &O(\log^{2q}\ell/\varepsilon^{4q})2^{r_{\mathsf{row}}}/2^{r_{\mathsf{col}}} \cdot 2^{r_{\mathsf{shared}}} \cdot 2^{2r_{\mathsf{col}}}/(r_{\mathsf{col}})^{c_u\log(1/\varepsilon)}\\
\le\ &2^r/\log^{0.5c_u\log(1/\varepsilon)}\ell < T/\log^{c_{\mathsf{hard}}}(T)
\end{aligned}$$

time that estimates p_{acc} within an additive error of at most

$$\mathop{\mathbb{E}}_{\mathsf{seed.shared}}\left[\bar{\delta}_{\mathsf{seed.shared}}\right] \le \eta \cdot (4qU+8)^{2q} + 16^q \mathop{\mathbb{E}}_{\mathsf{seed.shared}}[\delta'_{\mathsf{seed.shared}}].$$

Next, we upper bound the quantity $\mathbb{E}_{\mathsf{seed.shared}}[\delta'_{\mathsf{seed.shared}}]$. We abstract this task in the following lemma and defer the proof to Section 3.5.8.

Lemma 3.5.7. *Let $f : [N] \times [q] \to \mathbb{R}_{\ge 0}$ be a function and $d \ge 1$ be a constant. Suppose that*

1. *for every $s \in [N]$ and $i \in [q]$, $f(s,i) \le \alpha$ (where $\alpha \ge 1$);*

2. $\mathbb{E}_{s,i}[f(s,i)^d] \le \delta$.

Let $f(s) := \sum_{i\in[q]} f(s,i)$. *Then*

$$\mathop{\mathbb{E}}_{s}[(1+f(s))^{d-1}\cdot f(s)] \le q\delta^{1/d}(2q\alpha)^{d-1}.$$

To see how this lemma corresponds to our scenario: Let $d = 2q$ and $[N] = \{0,1\}^{r_{\mathsf{shared}}}$. For $\mathsf{seed.shared}$ and ι, if $\mathsf{itype}[\iota] = \mathsf{proof}$, then define $f(\mathsf{seed.shared},\iota) = \|f^{\mathsf{Bool}}_{\mathsf{seed.shared},\iota} - f^{\mathsf{Real}}_{\mathsf{seed.shared},\iota}\|_{2q}$; otherwise define $f(\mathsf{seed.shared},\iota) = 0$. Since the verification algorithm did not reject π^{Real}, we have

1. For every $\mathsf{seed.shared}$ and ι,

$$f(\mathsf{seed.shared},\iota) \le \|f^{\mathsf{Bool}}_{\mathsf{seed.shared},\iota}\|_{2q} + \|f^{\mathsf{Real}}_{\mathsf{seed.shared},\iota}\|_{2q} \le 2 + 2\eta\cdot U^{2q}.$$

2. Since the PCPP is smooth, the distribution of $(\mathsf{irow}[\iota],\mathsf{icol}[\iota])$ for random (seed,ι) (conditioned on $\mathsf{itype}[\iota] = \mathsf{proof}$) is the same as the uniform distribution over $[H_{\mathsf{proof}}]\times[W_{\mathsf{proof}}]$. Therefore

$$\begin{aligned}\mathop{\mathbb{E}}_{\mathsf{seed.shared},\iota}[f(\mathsf{seed.shared},\iota)^{2q}] &= \mathop{\mathbb{E}}_{\mathsf{seed},\iota:\mathsf{itype}[\iota]=\mathsf{proof}}[|\pi^{\mathsf{Bool}}_{\mathsf{irow}[\iota],\mathsf{icol}[\iota]} - \pi^{\mathsf{Real}}_{\mathsf{irow}[\iota],\mathsf{icol}[\iota]}|^{2q}]\\ &= \mathop{\mathbb{E}}_{i,j}[|\pi^{\mathsf{Bool}}_{i,j} - \pi^{\mathsf{Real}}_{i,j}|^{2q}]\\ &\le 16^q\cdot\delta + 128^q\eta U^{4q}.\end{aligned}$$

It follows from Lemma 3.5.7 that

$$\begin{aligned}\mathop{\mathbb{E}}_{\mathsf{seed.shared}}[\delta'_{\mathsf{seed.shared}}] &\le q(16^q\cdot\delta + 128^q\eta U^{4q})^{1/2q}(2q(2+2\eta\cdot U^{2q}))^{2q-1}\\ &\le q(17^q\cdot\delta)^{1/2q}\cdot(100q)^{2q} < 100^{-q}.\end{aligned}$$

Therefore, the algorithm estimates p_{acc} within an additive error of at most

$$\eta(4qU+8)^{2q} + 16^q\cdot 100^{-q} < 1/6,$$

thus successfully distinguishes between the case that $p_{\mathsf{acc}} > 5/6$ and that $p_{\mathsf{acc}} < 1/2$.

Finally, since C^{Prod} only depends on C (but not the input x to M^{PCPP}), the preprocessing phase of our algorithm only needs to know C. It outputs a "data structure" of length

$$(3A)^{4q}2^{4ql+r_{\mathsf{shared}}}\ell_{\mathrm{DS}} + 2^{r_{\mathsf{shared}}+4q}((Ad_S+2)^{2q}2^{2ql})\lceil\ell_{\mathsf{Prod}}/N\rceil\ell_{\mathrm{DS}} \le (6A)^{4q}\ell^{4q}2^{r_{\mathsf{shared}}}\ell_{\mathrm{DS}},$$

which is at most $\ell^{10q}\cdot\ell_{\mathrm{DS}}$ since $A = O(\log\ell/\varepsilon^2) = \ell^{o(1)}$ and $2^{r_{\mathsf{shared}}} = 2^{O(\log\ell/m)} = \ell^{o(1)}$.

3.5.5 Wrap Up: Description of M^{PCPP}

The machine M^{PCPP} hardwires C and the data structure of length $\ell^{10q}\cdot\ell_{\mathrm{DS}}$. This data structure can be computed in P^{NP}, hence the description of M^{PCPP} can be computed in P^{NP}.

On input x, we consider the smooth and rectangular PCPP for the language

$$L^{\mathsf{enc}} = \{\mathsf{Encode}(x) : x \in L^{\mathsf{hard}}\}.$$

(Recall that M^{PCPP} aims to reject every $x \notin L$ and accepts every $x \in L$ with easy witness.) We guess $(w_1, \ldots, w_{\hat{H}_{\mathsf{proof}}})$ and $(\alpha_1, \ldots, \alpha_{\hat{H}_{\mathsf{proof}}})$, which implicitly defines the PCPP proof matrices π^{Bool} and π^{Real}. Then we verify π^{Real} using Lemma 3.5.4 and reject immediately if π^{Real} did not pass the test. If π^{Real} passes the test (which means that it is "close" to a Boolean proof π^{Bool}), we use the algorithm described above to estimate p_{acc}. We accept x if and only if our estimation is above $2/3$.

The correctness of M^{PCPP} is easy to see:

Claim 3.5.8. *For every input x, if $x \notin L$ then M^{PCPP} rejects x; while if $x \in L$ and x has an easy witness then M^{PCPP} accepts x.*

Proof. If $x \notin L$, then it always holds that $p_{\mathsf{acc}} < 1/2$, so M^{PCPP} rejects. If $x \in L$ and x has an easy witness, then there exists a proof π, $(w_1, \ldots, w_{\hat{H}_{\mathsf{proof}}})$ and $(\alpha_1, \ldots, \alpha_{\hat{H}_{\mathsf{proof}}})$ such that

1. for every $j \in [W_{\mathsf{proof}}]$, $\pi^{\mathsf{Real}}_{i,j} \in [0, 1]$;
2. for every $i \in [W_{\mathsf{proof}}], j \in [\hat{H}_{\mathsf{proof}}]$, $\|\pi^{\mathsf{Real}}_i - \pi_i\|_1 \le \delta$.

Note that $\|\pi^{\mathsf{Real}}_i - \pi^{\mathsf{Bool}}_i\|_1 \le \|\pi^{\mathsf{Real}}_i - \pi_i\|_1 \le \delta$ since π^{Bool}_i is the closest Boolean string to π^{Real}_i, and thus $\|\pi_i - \pi^{\mathsf{Bool}}_i\|_1 \le 2\delta$. Since the probability that VPCPP accepts π is 1, by Lemma 2.5.8, $p_{\mathsf{acc}} \ge 1 - q \cdot 2\delta > 5/6$, so M^{PCPP} accepts. □

The machine M^{PCPP} guesses $\hat{H}_{\mathsf{proof}}(n + a) < n_{\mathsf{hard}}/10$ bits of nondeterminism, and uses $\tilde{O}(s\ell) + \ell^{10q}\ell_{\mathrm{DS}} < n_{\mathsf{hard}}/10$ bits of advice. Thus

$$L(M^{\mathsf{PCPP}}) \in \mathsf{NTIMEGUESS}_{\mathsf{RAM}}[T/\log^{c_{\mathsf{hard}}}(T), n_{\mathsf{hard}}/10]/_{(n_{\mathsf{hard}}/10)}.$$

3.5.6 The $\mathsf{FP}^{\mathsf{NP}}$ Algorithm for Remote-Point

Let $C : \{0,1\}^n \to \{0,1\}^\ell$ be the input circuit. We first construct the hard language L^{hard} and the algorithm M^{PCPP}. Since M^{PCPP} is a nondeterministic RAM algorithm that runs in $T/\log^{c_{\mathsf{hard}}}(T)$ time, uses at most $n_{\mathsf{hard}}/10$ nondeterministic bits and at most $n_{\mathsf{hard}}/10$ advice bits, it follows that there is an input $x_{\mathsf{hard}} \in \{0,1\}^{n_{\mathsf{hard}}}$ such that $M^{\mathsf{PCPP}}(x_{\mathsf{hard}}) \neq L^{\mathsf{hard}}(x_{\mathsf{hard}})$. Moreover, the advice string fed to M^{PCPP} is exactly our Remote-Point instance C. We can find such an input x_{hard} by running $\mathcal{R}(1^{n_{\mathsf{hard}}}, M^{\mathsf{PCPP}}, C)$, where $\mathcal{R}$ is the refuter guaranteed by Theorem 3.3.1. Thus, we can find x_{hard} in deterministic $\mathrm{poly}(T)$ time with an NP oracle.

It follows from Claim 3.5.8 that $x_{\mathsf{hard}} \in L^{\mathsf{hard}}$ but x_{hard} does not have an easy witness. Thus, we can use the NP oracle to find the lexicographically first PCPP proof matrix π such that

$$\Pr_{\mathsf{seed} \leftarrow \{0,1\}^r}[\mathsf{VPCPP}^{\mathsf{Encode}(x) \circ \pi}(\mathsf{seed}) \text{ accepts}] = 1.$$

Then, there must exist a row π_i that is $(1/2 - \varepsilon)$-far from $\mathrm{Range}(C)$. To see this, suppose that for every i, the i-th row π_i is $(1/2 - \varepsilon)$-close to $\mathrm{Range}(C)$. Then there exists some $w_i \in \{0,1\}^n$ such

that $\delta(\mathsf{Amp}(\pi_i), C(w_i)) \le 1/2-\varepsilon$. By Theorem 3.3.5, there is an advice α_i such that $\mathsf{dec}_{\alpha_i}(C(w_i))$ satisfies (1) for every $j \in [W_{\mathsf{proof}}]$, $(\mathsf{dec}_{\alpha_i}(C(w_i)))_j \in [0,1]$; (2) $\|\mathsf{dec}_{\alpha_i}(C(w_i)) - \pi_i\|_1 \le \delta$. It follows that π is an easy witness for x_{hard}, a contradiction.

Finally, we use the NP oracle to find the first row π_i, such that $\mathsf{Amp}(\pi_i)$ is $(1/2-\varepsilon)$-far from Range(C). The overall procedure takes deterministic $\mathrm{poly}(T) \le \mathrm{poly}(\ell)$ time with an NP oracle.

3.5.7 Proof of Claim 3.5.6

We need the following technical lemma (see [CW19b, Lemma 28]):

Lemma 3.5.9. *Let $d \ge 2$ be an integer, $f_1, f_2, \ldots, f_d, g_1, g_2, \ldots, g_d : [N] \to \mathbb{R}$ be functions. For all $i \in [d]$, suppose that $\|f_i\|_d \le 1$, and define $\varepsilon := \sum_{i=1}^d \|f_i - g_i\|_d$. Then*

$$\left| \mathop{\mathbb{E}}_{x \leftarrow [N]} \left[\prod_{i=1}^d f_i(x) - \prod_{i=1}^d g_i(x) \right] \right| \le (1+\varepsilon)^{d-1} \cdot \varepsilon.$$

The above lemma is a consequence of the following generalisation of Hölder's inequality:

Fact 3.5.10. *Let $f_1, f_2, \ldots, f_d : [N] \to \mathbb{R}$ be functions, $f : [N] \to \mathbb{R}$ be their product, i.e., $f(x) = \prod_{i=1}^d f_i(x)$. Then $\|f\|_1 \le \prod_{i=1}^d \|f_i\|_d$.*

Proof of Lemma 3.5.9. Let $\varepsilon_i := \|f_i - g_i\|_d$, then $\varepsilon = \sum_{i=1}^d \varepsilon_i$. Define

$$\mathsf{Hyb}_i := \mathop{\mathbb{E}}_{x \leftarrow [N]} \left[\prod_{j=1}^{i} f_j(x) \cdot \prod_{j=i+1}^{d} g_j(x) \right].$$

Then, for every $1 \le i \le d$,

$$\begin{aligned}
|\mathsf{Hyb}_i - \mathsf{Hyb}_{i-1}| &\le \mathop{\mathbb{E}}_{x \leftarrow [N]} \left[\left| \prod_{j=1}^{i-1} f_j(x) \cdot \prod_{j=i+1}^{d} g_j(x) \cdot (f_i(x) - g_i(x)) \right| \right] \\
&\le \prod_{j=1}^{i-1} \|f_j\|_d \cdot \prod_{j=i+1}^{d} \|g_j\|_d \cdot \|f_i - g_i\|_d && \text{(Fact 3.5.10)} \\
&\le \prod_{j=2}^{d} (1+\varepsilon_j) \cdot \varepsilon_i \\
&\le (1+\varepsilon)^{d-1} \cdot \varepsilon_i.
\end{aligned}$$

It follows that

$$\left| \mathop{\mathbb{E}}_{x \leftarrow [N]} \left[\prod_{i=1}^d f_i(x) - \prod_{i=1}^d g_i(x) \right] \right| = |\mathsf{Hyb}_d - \mathsf{Hyb}_0| \le \sum_{i=1}^d |\mathsf{Hyb}_i - \mathsf{Hyb}_{i-1}| \le (1+\varepsilon)^{d-1} \cdot \varepsilon. \qquad \square$$

Recall that for $S, S' \subseteq [q]$, we define

$$p_{\mathsf{acc}}(\mathsf{seed.shared}, S, S') := \mathop{\mathbb{E}}_{\substack{\mathsf{seed.row} \leftarrow \{0,1\}^{r_{\mathsf{row}}} \\ \mathsf{seed.col} \leftarrow \{0,1\}^{r_{\mathsf{col}}}}} \left[\prod_{\iota \in S} (\mathsf{row}_\iota^{\mathsf{Bool}})_{\mathsf{icol}[\iota]} \cdot \prod_{\iota \in S'} pc_\iota \right],$$

$$p^{\mathsf{Real}}_{\mathsf{acc}}(\mathsf{seed.shared}, S, S') := \mathop{\mathbb{E}}_{\substack{\mathsf{seed.row}\leftarrow\{0,1\}^{r_{\mathsf{row}}}\\ \mathsf{seed.col}\leftarrow\{0,1\}^{r_{\mathsf{col}}}}} \left[\prod_{\iota\in S}(\mathsf{row}^{\mathsf{Real}}_{\iota})_{\mathsf{icol}[\iota]} \cdot \prod_{\iota\in S'} pc_{\iota}\right],$$

$$\delta_{\mathsf{seed.shared}} := \sum_{\iota:\mathsf{itype}[\iota]=\mathsf{proof}} \|f^{\mathsf{Bool}}_{\mathsf{seed.shared},\iota} - f^{\mathsf{Real}}_{\mathsf{seed.shared},\iota}\|_{2q}.$$

Claim 3.5.6. *For every $S, S' \subseteq [q]$,*

$$|p_{\mathsf{acc}}(\mathsf{seed.shared}, S, S') - p^{\mathsf{Real}}_{\mathsf{acc}}(\mathsf{seed.shared}, S, S')| \le (1+\delta_{\mathsf{seed.shared}})^{2q-1} \cdot \delta_{\mathsf{seed.shared}}.$$

Proof. Define the following $2(|S|+|S'|)$ functions f^{Bool}_i, g^{Bool}_j, f^{Real}_i, g^{Real}_j, where $i \in S$ and $j \in S'$. Each function takes $(\mathsf{seed.row}, \mathsf{seed.col})$ as inputs, and:

$$f^{\mathsf{Bool}}_i := (\mathsf{row}^{\mathsf{Bool}}_i)_{\mathsf{icol}[i]}; \quad f^{\mathsf{Real}}_i := (\mathsf{row}^{\mathsf{Real}}_i)_{\mathsf{icol}[i]}; \quad \text{and} \quad g^{\mathsf{Bool}}_j = g^{\mathsf{Real}}_j := pc_j.$$

(Note: for convenience, we omit the input $(\mathsf{seed.row}, \mathsf{seed.col})$.) It follows that $\|f^{\mathsf{Bool}}_i\|_{2q} \le 1$ and $\|g^{\mathsf{Bool}}_j\|_{2q} \le 1$; for every $j \in S'$, $\|g^{\mathsf{Bool}}_i - g^{\mathsf{Real}}_i\|_{2q} = 0$; and for every $i \in S$,

$$\|f^{\mathsf{Bool}}_i - f^{\mathsf{Real}}_i\|_{2q} = \begin{cases} 0 & \text{if } \mathsf{itype}[i] = \mathsf{input}; \\ \|f^{\mathsf{Bool}}_{\mathsf{seed.shared},i} - f^{\mathsf{Real}}_{\mathsf{seed.shared},i}\|_{2q} & \text{if } \mathsf{itype}[i] = \mathsf{proof}. \end{cases}$$

Therefore, by Lemma 3.5.9,

$$\begin{aligned} &|p_{\mathsf{acc}}(\mathsf{seed.shared}, S, S') - p^{\mathsf{Real}}_{\mathsf{acc}}(\mathsf{seed.shared}, S, S')| \\ =&\left|\mathop{\mathbb{E}}_{\substack{\mathsf{seed.row}\leftarrow\{0,1\}^{r_{\mathsf{row}}}\\ \mathsf{seed.col}\leftarrow\{0,1\}^{r_{\mathsf{col}}}}} \left[\prod_{i\in S} f^{\mathsf{Bool}}_i \cdot \prod_{j\in S'} g^{\mathsf{Bool}}_j - \prod_{i\in S} f^{\mathsf{Real}}_i \cdot \prod_{j\in S'} g^{\mathsf{Real}}_j\right]\right| \\ \le&(1+\delta_{\mathsf{seed.shared}})^{2q-1} \cdot \delta_{\mathsf{seed.shared}}. \end{aligned}$$

□

3.5.8 Proof of Lemma 3.5.7

Lemma 3.5.7. *Let $f : [N] \times [q] \to \mathbb{R}_{\ge 0}$ be a function and $d \ge 1$ be a constant. Suppose that*

1. *for every $s \in [N]$ and $i \in [q]$, $f(s,i) \le \alpha$ (where $\alpha \ge 1$);*
2. *$\mathbb{E}_{s,i}[f(s,i)^d] \le \delta$.*

Let $f(s) := \sum_{i\in[q]} f(s,i)$. Then

$$\mathop{\mathbb{E}}_{s}[(1+f(s))^{d-1} \cdot f(s)] \le q\delta^{1/d}(2q\alpha)^{d-1}.$$

Proof. By Jensen's inequality,

$$\mathop{\mathbb{E}}_{s}[f(s)] = q \mathop{\mathbb{E}}_{s,i}[f(s,i)] \le q\delta^{1/d}.$$

It follows that for every $k \ge 1$,

$$\mathop{\mathbb{E}}_{s}[f(s)^k] \le \mathop{\mathbb{E}}_{s}[f(s)] \cdot \max_{s}\{f(s)\}^{k-1} \le q\delta^{1/d} \cdot (q\alpha)^{k-1}.$$

Finally, we have

$$\mathop{\mathbb{E}}_{s}[(1+f(s))^{d-1}\cdot f(s)] = \sum_{i=0}^{d-1}\binom{d-1}{i}\cdot \mathop{\mathbb{E}}_{s}[f(s)^{i+1}] \le q\delta^{1/d}(2q\alpha)^{d-1}. \qquad \square$$

3.6 Hard Partial Truth Tables

Instead of allowing a P^{NP} preprocessing on the circuits, the algorithm for Satisfying-Pairs used to solve Partial-Hard allows a P^{NP} preprocessing on *inputs*, formally defined as follows.

Definition 3.6.1 (Algorithms for Satisfying-Pairs with P^{NP} Preprocessing on Inputs)**.** Let P be one of $\mathscr{C}$-Satisfying-Pairs, $\#\mathscr{C}$-Satisfying-Pairs, Approx_δ-$\mathscr{C}$-Satisfying-Pairs, and Gap_δ-$\mathscr{C}$-Satisfying-Pairs. A t-time algorithm for P with P^{NP} preprocessing of an ℓ-size data structure on inputs is a pair of algorithms (A_1, A_2) that solves P in two phases:

1. Given the inputs $x_1, x_2, \ldots, x_M \in \{0,1\}^n$, the polynomial-time algorithm A_1 with oracle access to a SAT oracle computes a string $\mathsf{DS} \in \{0,1\}^\ell$.
2. Given the circuits $C_1, C_2, \ldots, C_N : \{0,1\}^n \to \{0,1\}$ of size s and the string DS, the algorithm A_2 solves P on the instance $(C_1, \ldots, C_N, x_1, \ldots, x_M)$ in time t.

Theorem 3.6.2. *There are constants $\varepsilon > 0$ and c_0 such that the following holds. Let $0 < \eta < 1/2$ be a constant, $\mathscr{C}[s]$ be a typical and complete circuit class where $s = s(n) > n$ is a size parameter, and $\mathscr{C}'[2s] := \mathsf{OR}_2 \circ \mathscr{C}[s]$. Let $\ell(n)$ be a good function such that $s(n)^{1+\Omega(1)} \le \ell(n) \le 2^n$.*

Assumption: *Suppose that for some constant $c \ge 1$, there is an $(NM/\log^{c_0}(NM))$-time algorithm for $\mathsf{Approx}_\varepsilon$-$\mathscr{C}'$-Satisfying-Pairs with $N := \ell^{c+1-\eta}\cdot\mathrm{polylog}(\ell)$ circuits of size $\mathrm{poly}(s(n))$ and $M := \ell^{1-\eta}\cdot\mathrm{polylog}(\ell)$ inputs of length $2n$, allowing a P^{NP} preprocessing of an M^c-size data structure on inputs.*

Conclusion: *There is an $\mathsf{FP}^{\mathsf{NP}}$ algorithm for $\mathscr{C}[s]$-Partial-Hard with $\ell(n)$ input strings. More precisely, given a list of inputs $z_1, z_2, \ldots, z_\ell \in \{0,1\}^n$, we can compute a list of bits $b_1, b_2, \ldots, b_\ell$ such that for every $\mathscr{C}$ circuit $C : \{0,1\}^n \to \{0,1\}$ of size s, there exists an $i \in [\ell]$ such that $C(z_i) \ne b_i$.*

Proof Sketch of Theorem 3.6.2. The proof is very similar to the proof of Theorem 3.4.2; in fact, it is (nearly) equivalent to first reducing Partial-Hard to Avoid and then invoking Theorem 3.4.2. Therefore, we only highlight the differences.

It is without loss of generality to assume ℓ is a power of 2 and $c \ge 2$. We set the following parameters:

$$\begin{aligned}
m &:= 5(c+2)/\eta = O(1), \\
w_{\mathsf{proof}} &:= \log \ell, & W_{\mathsf{proof}} &:= 2^{w_{\mathsf{proof}}} = \ell, \\
h_{\mathsf{proof}} &:= (c+1)\log \ell, & H_{\mathsf{proof}} &:= 2^{h_{\mathsf{proof}}} = \ell^{c+1}, \\
n_{\mathsf{hard}} &:= 100\hat{H}_{\mathsf{proof}}\cdot \mathrm{polylog}(\ell)\cdot s\log s, \\
T &:= H_{\mathsf{proof}}\cdot W_{\mathsf{proof}}/\log^{c_{\mathsf{tm}}}(\ell), \\
h_{\mathsf{input}} &:= \left(1 - \tfrac{\Theta(\log\log T)}{\log T}\right) h_{\mathsf{proof}}, & H_{\mathsf{input}} &:= 2^{h_{\mathsf{input}}} = H_{\mathsf{proof}}/\mathrm{polylog}(\ell), \\
w_{\mathsf{input}} &:= \lceil \log \tilde{n}_{\mathsf{hard}} \rceil - h_{\mathsf{input}}, & W_{\mathsf{input}} &:= 2^{w_{\mathsf{input}}} = s\log s\cdot \mathrm{polylog}(\ell).
\end{aligned}$$

Here $\tilde{n}_{\mathsf{hard}} = O(n_{\mathsf{hard}})$ is the codeword length of a length-n_{hard} string encoded via Encode where $(\mathsf{Encode}, \mathsf{Decode})$ is a fixed error-correcting code in Theorem 2.4.1; and c_{tm} is a sufficiently large constant.

We can check the technical condition $n_{\mathsf{hard}}^{1+\Omega(1)} \leq T \leq 2^{\mathrm{poly}(n_{\mathsf{hard}})}$, so it is valid to invoke Theorem 3.3.1. Also, $(5/m)\log T \leq w_{\mathsf{proof}}$, so it is valid to invoke the 2-query rectangular PCPP in Theorem 2.5.10. There are other checks for technical conditions that we omit here. The proof matrix is of size $\hat{H}_{\mathsf{proof}} \times W_{\mathsf{proof}}$, where $\hat{H}_{\mathsf{proof}} = 2^{\hat{h}_{\mathsf{proof}}}$ and $\hat{h}_{\mathsf{proof}} = \log T + \Theta(m \log\log T) - w_{\mathsf{proof}} = (c+1)\log \ell + O(\log\log \ell)$.

The first difference is the definition of "easy witness". We say x *has an easy witness* if there is a proof matrix π (of size $\hat{H}_{\mathsf{proof}} \times W_{\mathsf{proof}}$) for the statement "$\mathsf{Encode}(x) \in L^{\mathsf{enc}}$" such that:

(completeness) for every $\mathsf{seed} \in \{0,1\}^r$, $\mathsf{VPCPP}^{\mathsf{Encode}(x)\circ\pi}(\mathsf{seed})$ accepts w.p. at least c_{pcp};

(easiness) for every row π_j of π, there exists a *size-s $\mathscr{C}$ circuit* $C_j : \{0,1\}^n \to \{0,1\}$ such that for every i, $\pi_{j,i} = C_j(z_i)$.

Then, our machine M^{PCPP} guesses $\hat{H}_{\mathsf{proof}}$ size-s $\mathscr{C}$ circuits $C_1, C_2, \ldots, C_{\hat{H}_{\mathsf{proof}}} : \{0,1\}^n \to \{0,1\}$. Let π be the $\hat{H}_{\mathsf{proof}} \times W_{\mathsf{proof}}$ proof matrix where for each $j \in [\hat{H}_{\mathsf{proof}}], i \in [W_{\mathsf{proof}}]$, $\pi_{j,i} = C_j(z_i)$. We need to estimate

$$p_{\mathsf{acc}} := \Pr_{\mathsf{seed} \leftarrow \{0,1\}^r}[\mathsf{VPCPP}^{\mathsf{Encode}(x)\circ\pi}(\mathsf{seed}) \text{ accepts}].$$

We reduce the problem of estimating p_{acc} to $2^{r_{\mathsf{shared}}}$ instances of $\mathsf{Approx}_{\varepsilon}$-$\mathscr{C}'$-Satisfying-Pairs, where $\varepsilon := (c_{\mathsf{pcp}} - s_{\mathsf{pcp}})/4$. However, now, each instance consists of $N := 2^{r_{\mathsf{row}}} = 2^{\hat{h}_{\mathsf{proof}} - (5/m)\log T}$ circuits and $M := 2^{r_{\mathsf{col}}} = 2^{w_{\mathsf{proof}} - (5/m)\log T}$ inputs.[15]

We enumerate $\mathsf{seed.shared}$. For each $\mathsf{seed.shared}$, we create an $\mathsf{Approx}_{\varepsilon}$-$\mathscr{C}'$-Satisfying-Pairs instance $\mathcal{I}_{\mathsf{seed.shared}}$ corresponding to $\mathsf{seed.shared}$, which contains an input $\mathsf{Input}_{\mathsf{seed.shared},\mathsf{seed.col}}$ for every $\mathsf{seed.col}$ and a circuit $C_{\mathsf{seed.shared},\mathsf{seed.row}}$ for every $\mathsf{seed.row}$. We elaborate on how this instance is constructed, as this is different from Theorem 3.4.2.

Each $\mathsf{seed.col}$ corresponds to an input $\mathsf{Input}_{\mathsf{seed.shared},\mathsf{seed.col}}$ of the following form:

$$(a_1, \ldots, a_q, pc_1^{\mathsf{col}}, \ldots, pc_p^{\mathsf{col}}),$$

where $p + q \leq 2$, for each $i \in [q]$,

$$a_i := \begin{cases} \mathsf{icol}[i] & \text{if } \mathsf{itype}[i] = \mathsf{input}; \\ z_{\mathsf{icol}[i]} & \text{if } \mathsf{itype}[i] = \mathsf{proof}, \end{cases}$$

and pc_i^{col} represents the contribution of $\mathsf{seed.col}$ in the i-th parity-check bit.

The circuit $C_{\mathsf{seed.shared},\mathsf{seed.row}}$ corresponding to $\mathsf{seed.row}$ is as follows:

- It receives input $(a_1, \ldots, a_q, pc_1^{\mathsf{col}}, \ldots, pc_p^{\mathsf{col}})$.

[15] That is, the role of inputs and circuits are swapped as compared to Theorem 3.4.2.

- For each $j \in [q]$, let

$$\mathsf{ans}_j = \begin{cases} \mathsf{Encode}(x)_{\mathsf{irow}[j],a_j} & \text{if } \mathsf{itype}[j] = \mathsf{input} \\ C_{\mathsf{irow}[j]}(a_j) & \text{if } \mathsf{itype}[j] = \mathsf{proof} \end{cases}.$$

  Note that since $\mathscr{C}$ is complete, we can compute a $\mathscr{C}$ circuit of size $\mathrm{poly}(W_{\mathsf{input}}) = \mathrm{poly}(s)$ whose truth table is the $\mathsf{irow}[j]$-th row of $\mathsf{Encode}(x)$. That is, we can compute a $\mathscr{C}$ circuit of size $\mathrm{poly}(s)$ that on input a_j, outputs ans_j.

- For each $j \in [q]$, let pc_j^{row} be the contribution of $\mathsf{seed.row}$ in the j-th parity-check bit.

- It returns

$$\mathsf{VDec}(\mathsf{ans}_1, \ldots, \mathsf{ans}_q, pc_1^{\mathsf{col}} \oplus pc_1^{\mathsf{row}}, \ldots, pc_p^{\mathsf{col}} \oplus pc_p^{\mathsf{row}}).$$

Here, VDec is the decision predicate of VPCPP, and is an OR_2 of its input bits or their negations. Since $\mathscr{C}$ is typical, C is a $\mathsf{OR}_2 \circ \mathscr{C}$ circuit. And one can easily verify that for each $\mathsf{seed} = (\mathsf{seed.shared}, \mathsf{seed.row}, \mathsf{seed.col})$, $C_{\mathsf{seed.shared},\mathsf{seed.row}}(\mathsf{Input}_{\mathsf{seed.shared},\mathsf{seed.col}}) = 1$ if and only if $\mathsf{VPCPP}^{\mathsf{Enc}(x)\circ\pi}(\mathsf{seed})$ accepts. It follows that we can estimate p_{acc} by solving the instances $\mathcal{I}_{\mathsf{seed.shared}}$ for every $\mathsf{seed.shared}$.

To summarise, our algorithm M^{PCPP} works as follows. It first computes $\mathsf{Encode}(x)$ and guesses $C_1, C_2, \ldots, C_{\hat{H}_{\mathsf{proof}}}$. Then, it enumerates $\mathsf{seed.shared}$, produces the instances $\mathcal{I}_{\mathsf{seed.shared}}$, and feeds them to the algorithm for $\mathsf{Approx}_\varepsilon$-$\mathscr{C}$-SATISFYING-PAIRS to obtain an estimation $p'_{\mathsf{acc}}(\mathsf{seed.shared})$. Let p'_{acc} be the average of $p'_{\mathsf{acc}}(\mathsf{seed.shared})$ over all $\mathsf{seed.shared} \in \{0,1\}^{r_{\mathsf{shared}}}$.

We can still see that M^{PCPP} rejects every $x \notin L^{\mathsf{hard}}$ and accepts every x with an easy witness. The machine M^{PCPP} runs in $T/\log^{c_{\mathsf{hard}}} T$ time, guesses $\hat{H}_{\mathsf{proof}} \cdot 5s \log s < n_{\mathsf{hard}}/10$ nondeterministic bits (since a size-s circuit can be encoded with at most $5s \log s$ bits), and uses at most $\ell^{c+1} < n_{\mathsf{hard}}/10$ advice bits. By Theorem 3.3.1, M^{PCPP} cannot compute L^{hard}.

The hard partial truth tables algorithm. Given a list of inputs $z_1, z_2, \ldots, z_\ell \in \{0,1\}^n$, our algorithm for finding a hard partial truth table $((z_1, b_1), (z_2, b_2), \ldots, (z_\ell, b_\ell))$ works as follows. First, we construct the hard language L^{hard} and the algorithm M^{PCPP}. Let α be the advice string fed to M^{PCPP} and $\mathcal{R}$ be the refuter in Theorem 3.3.1, we can use $\mathcal{R}(1^{n_{\mathsf{hard}}}, M^{\mathsf{PCPP}}, \alpha)$ to find an input x_{hard} where M^{PCPP} fails on x_{hard}; in particular, $M^{\mathsf{PCPP}}(x_{\mathsf{hard}}) = 0$ but $x_{\mathsf{hard}} \in L^{\mathsf{hard}}$. This takes deterministic $\mathrm{poly}(\hat{H}_{\mathsf{proof}}) = \mathrm{poly}(\ell)$ time with an NP oracle.

Then we find the lexicographically first proof matrix π such that $\mathsf{VPCPP}^{\mathsf{Encode}(x_{\mathsf{hard}})\circ\pi}$ accepts w.p. at least c_{pcp}, using the NP oracle. There has to be some $j \in [\hat{H}_{\mathsf{proof}}]$ such that for every size-s $\mathscr{C}$ circuit C, there exists $i \in [W_{\mathsf{proof}}]$ such that $C(z_i) \neq \pi_{j,i}$; moreover, the first such j can be found in $\mathrm{poly}(\hat{H}_{\mathsf{proof}}) = \mathrm{poly}(\ell)$ time with an NP oracle. We can pick

$$((z_1, \pi_{j,1}), (z_2, \pi_{j,2}), \ldots, (z_{W_{\mathsf{proof}}}, \pi_{j,W_{\mathsf{proof}}}))$$

as the partial truth table that is hard for size-s $\mathscr{C}$ circuits. □

3.7 Average-Case Hard Partial Truth Tables

Theorem 3.7.1. *There is a universal constant $c_u \geq 1$ such that the following holds. Let $s = s(n) > n$ be a circuit size parameter, $N := N(n)$ be a parameter such that $2^{\log^{c_u} s} < N < 2^{s^{0.99}}$, $\varepsilon := \varepsilon(n) > s^{-c_u}$ be the error parameter, and $\ell := N^{c_u \log(1/\varepsilon)}$. Let $\mathscr{C}[s]$ be a typical and complete circuit class, and denote $\mathscr{C}'[c_u s] := \mathsf{AND}_{c_u} \circ \mathscr{C}[s]$ (i.e. a $\mathscr{C}'$ circuit of size $c_u s$ refers to the AND of at most c_u $\mathscr{C}$ circuits of size s).*

Assumption: *Let $P := (\log N)^{\log(1/\varepsilon)}$. Suppose there is a deterministic algorithm running in time $T^{\mathrm{alg}} := N^2/P^{c_u}$ that, given as input a list of N $\mathscr{C}'[c_u s]$ circuits $\{C_i\}$ and a list of N inputs $\{x_j\}$ with input length $n \cdot \mathrm{polylog}(\ell)$, estimates $\Pr_{i,j \leftarrow [N]}[C_i(x_j) = 1]$ with additive error $\eta := \varepsilon^{c_u}$.*

Conclusion: *There is an $\mathsf{FP}^{\mathsf{NP}}$ algorithm for $\mathscr{C}[s]$-Partial-AvgHard with $\ell(n)$ input strings. More precisely, given a list of inputs $w_1, w_2, \ldots, w_\ell \in \{0,1\}^n$, we can compute a list of bits $b_1, b_2, \ldots, b_\ell$ such that for every $\mathscr{C}$ circuit $C : \{0,1\}^n \to \{0,1\}$ of size s,*

$$\Pr_{i \leftarrow [\ell]}[C(w_i) \neq b_i] \geq \frac{1}{2} - \varepsilon.$$

Proof Sketch of Theorem 3.7.1. The proof is similar to that of Theorem 3.5.1, so here we only highlight the difference. Roughly speaking, the main difference is that we swap the role of inputs and circuits.

For a circuit C and a list of inputs $w = (w_1, w_2, \ldots, w_\ell)$, with slight abuse of notation, we define $C(w) := C(w_1) \circ C(w_2) \circ \cdots \circ C(w_\ell)$.

Analysing $\mathsf{Prod} \circ \mathsf{Sum}$ circuits. Let $d \geq 1$ be a constant. We use $\mathsf{Prod} \circ \mathsf{Sum}$ to denote the class of multi-output circuits that take inputs $y \in \{0,1\}^{\ell_y}$ and α, and has the following components:

- Let ℓ_{Sum} denote the number of middle "linear sum" gates. For each $i \in [\ell_{\mathsf{Sum}}]$, the i-th gate outputs
$$\mathsf{Sum}_i(y, \alpha) := \sum_{k \in [A]} \mathsf{coeff}_k(\alpha) \cdot y_{\mathsf{idx}_k(\alpha, i)}.$$
- Let ℓ_{Prod} denote the number of output gates. The i-th output gate is a product gate of fan-in d, and is connected to the $q_1(i), q_2(i), \ldots, q_d(i)$-th linear sum circuits. Its output is
$$C_i^{\mathsf{Prod}}(y, \alpha) := \prod_{t=1}^{d} \mathsf{Sum}_{q_t(i)}(y, \alpha).$$

Remark 3.7.2. The important measures of a $\mathsf{Prod}_d \circ \mathsf{Sum}$ circuit are:
- the number of gates in each level (ℓ_{Sum}, ℓ_{Prod});
- the fan-in of the top Prod gates (d);
- the fan-in (A), coefficient sum (U), and locality (l) of the linear sum layer.

As an intermediate step, we need the following algorithm.

Lemma 3.7.3. *Let $\mathscr{C}$ be a typical circuit class, $M' \geq 1$ and $\eta \in (0,1)$ be parameters. Suppose there is a deterministic algorithm running in time $T^{\mathrm{alg}} = T^{\mathrm{alg}}(N, M)$ that, given as inputs a list of $\hat{M} \leq M$ $\mathsf{AND}_d \circ \mathscr{C}$ circuits $\{C_i\}$ and a list of $\hat{N} \leq N$ inputs $\{x_j\}$ of length $n \cdot \mathrm{polylog}(\ell)$, estimates the following quantity with additive error η:*

$$\Pr_{i \leftarrow [\hat{M}], j \leftarrow [\hat{N}]}[C_i(x_j)].$$

Then, for any constant $\ell_{\mathscr{C}} > 0$, there is a deterministic algorithm running in time $A^d \cdot (\ell_{\mathscr{C}}^d + \ell_{\mathsf{Prod}}/N) \cdot (2^{dl} + M'/M) \cdot O(T^{\mathrm{alg}})$ that, given as inputs a $\mathsf{Prod}_d \circ \mathsf{Sum}$ circuit C^{Prod} with parameters specified in Remark 3.7.2, a list of ℓ_x strings $\{x_j\}$ of length $n \cdot \mathrm{polylog}(\ell)$, a list of M' inputs $\{\alpha_j\}$, and a list of M' $\mathscr{C}$ circuits $\{C_i\}$ from $\{0,1\}^{n \cdot \mathrm{polylog}(\ell)}$ to $\{0,1\}^{\ell_{\mathscr{C}}}$, estimates the following quantity with additive error $\eta \cdot U^d$:

$$\mathop{\mathbb{E}}_{i \leftarrow [\ell_{\mathsf{Prod}}], j \leftarrow [M']} \left[C_i^{\mathsf{Prod}}(C_j(x), \alpha_j) \right].$$

Recall here that $C_j(x) = C_j(x_1) \circ C_j(x_2) \circ \cdots \circ C_j(x_{\ell_x})$.

The proof is similar to that of Theorem 3.5.3 and we only provide a sketch here.

Proof Sketch of Lemma 3.7.3. We identify $\mathsf{idx}_k(\alpha, i) \in [\ell_y]$ with $(\mathsf{idx}_k^x(\alpha, i), \mathsf{idx}_k^{\mathscr{C}}(\alpha, i)) \in [\ell_x] \times [\ell_{\mathscr{C}}]$ (note that $\ell_y = \ell_x \ell_{\mathscr{C}}$). Then,

$$\begin{aligned} C_i^{\mathsf{Prod}}(C_j(x), \alpha_j) &= \prod_{t=1}^{d} \mathsf{Sum}_{q_t(i)}(C_j(x), \alpha_j) \\ &= \prod_{t=1}^{d} \sum_{k \in [A]} \mathsf{coeff}_k(\alpha_j) \cdot (C_j)_{\mathsf{idx}_k^{\mathscr{C}}(\alpha_j, q_t(i))}(x_{\mathsf{idx}_k^x(\alpha_j, q_t(i))}) \\ &= \sum_{k_1 \in [A]} \sum_{k_2 \in [A]} \cdots \sum_{k_d \in [A]} \prod_{t=1}^{d} \left(\mathsf{coeff}_{k_t}(\alpha_j) \cdot (C_j)_{\mathsf{idx}_{k_t}^{\mathscr{C}}(\alpha_j, q_t(i))}(x_{\mathsf{idx}_{k_t}^x(\alpha_j, q_t(i))}) \right). \end{aligned} \tag{3.11}$$

As we can enumerate $k_1, k_2, \ldots, k_d \in [A]$ in A^d time, it suffices to estimate

$$\mathop{\mathbb{E}}_{i \leftarrow [\ell_{\mathsf{Prod}}], j \leftarrow [M']} \left[\prod_{t=1}^{d} \left(\mathsf{coeff}_{k_t}(\alpha_j) \cdot (C_j)_{\mathsf{idx}_{k_t}^{\mathscr{C}}(\alpha_j, q_t(i))}(x_{\mathsf{idx}_{k_t}^x(\alpha_j, q_t(i))}) \right) \right]. \tag{3.12}$$

Fix $k_1, k_2, \ldots, k_d \in [A]$. Since C^{Prod} is of locality l, we can see that (3.12) only depends on dl bits of α_j. We partition $j \in [M']$ into 2^{dl} groups as follows: For each $\alpha \in \{0,1\}^{dl}$, let $\mathcal{J}_\alpha$ be the set of $j \in [M']$ such that the dl bits of α_j (that (3.12) for this j depends on) equals to α. We will estimate (3.12) by enumerating $\alpha \in \{0,1\}^{dl}$, estimating it for $j \leftarrow \mathcal{J}_\alpha$ (instead of $j \leftarrow [M']$), and then taking the average over all possible α.

Now we fix any $\alpha \in \{0,1\}^{dl}$. We can rephrase the following items as they no longer depend on α_j:

$$\mathsf{coeff}_{k_t}(\alpha_j) =: \mathsf{coeff}'_t;$$

$$\mathsf{idx}^{x}_{k_t}(\alpha_j, q_t(i)) =: \mathsf{idx}'_{x,t}(i);$$
$$\mathsf{idx}^{\mathscr{C}}_{k_t}(\alpha_j, q_t(i)) =: \mathsf{idx}'_{\mathscr{C},t}(i).$$

It then suffices to estimate

$$\begin{aligned}&\mathop{\mathbb{E}}_{i\leftarrow[\ell_{\mathsf{Prod}}],j\leftarrow\mathcal{J}_\alpha}\left[\prod_{t=1}^{d}\mathsf{coeff}'_t\cdot(C_j)_{\mathsf{idx}'_{\mathscr{C},t}(i)}\left(x_{\mathsf{idx}'_{x,t}(i)}\right)\right]\\ &=\left(\prod_{t=1}^{d}\mathsf{coeff}'_t\right)\cdot\mathop{\mathbb{E}}_{i\leftarrow[\ell_{\mathsf{Prod}}],j\leftarrow\mathcal{J}_\alpha}\left[\bigwedge_{t=1}^{d}(C_j)_{\mathsf{idx}'_{\mathscr{C},t}(i)}\left(x_{\mathsf{idx}'_{x,t}(i)}\right)\right].\end{aligned}\tag{3.13}$$

Now for $\beta\in[\ell_{\mathscr{C}}]^d$, let $\mathcal{I}_\beta := \{i\in[\ell_{\mathsf{Prod}}] : \forall t\in[d], \mathsf{idx}'_{\mathscr{C}',t}(i)=\beta_t\}$. We enumerate over β, and now it suffices to estimate

$$\mathop{\mathbb{E}}_{i\leftarrow\mathcal{I}_\beta,j\leftarrow\mathcal{J}_\alpha}\left[\bigwedge_{t=1}^{d}(C_j)_{\beta_t}\left(x_{\mathsf{idx}'_{x,t}(i)}\right)\right].\tag{3.14}$$

Each expression of the form $\mathbb{E}_{i,j}\left[\bigwedge_{t=1}^{d}(C_j)_{\beta_t}\left(x_{\mathsf{idx}'_{x,t}(i)}\right)\right]$ reduces to the Satisfying-Pairs problem for $\mathsf{AND}_d\circ\mathscr{C}$ circuits. More precisely, we split $\mathcal{I}_\beta$ into blocks of size N and $\mathcal{J}_\alpha$ into blocks of size M, and use the assumed algorithm to estimate (3.14). By a similar argument as in Theorem 3.5.3, the additive error of our algorithm is bounded by $\eta\cdot U^d$.

Complexity. The subroutine for estimating (3.14) takes $O(T^{\mathrm{alg}})\cdot\lceil|\mathcal{J}_\alpha|/M\rceil\cdot\lceil|\mathcal{I}_\beta|/N\rceil$ time. Therefore, the subroutine for estimating (3.13) takes

$$\sum_{\beta\in[\ell_{\mathscr{C}}]^d}O(T^{\mathrm{alg}})\cdot\lceil|\mathcal{J}_\alpha|/M\rceil\cdot\lceil|\mathcal{I}_\beta|/N\rceil = O(T^{\mathrm{alg}})\cdot\lceil|\mathcal{J}_\alpha|/M\rceil\cdot(\ell_{\mathscr{C}}^d+\ell_{\mathsf{Prod}}/N)$$

time. It then follows that the subroutine for estimating (3.12) takes

$$\sum_{\alpha\in\{0,1\}^{dl}}O(T^{\mathrm{alg}})\cdot\lceil|\mathcal{J}_\alpha|/M\rceil\cdot(\ell_{\mathscr{C}}^d+\ell_{\mathsf{Prod}}/N) = O(T^{\mathrm{alg}})\cdot(2^{dl}+M'/M)\cdot(\ell_{\mathscr{C}}^d+\ell_{\mathsf{Prod}}/N)$$

time, and finally, estimating (3.11) takes

$$O(T^{\mathrm{alg}})\cdot(2^{dl}+M'/M)\cdot(\ell_{\mathscr{C}}^d+\ell_{\mathsf{Prod}}/N)\cdot A^d$$

time, which is the total time complexity of our algorithm. □

Set up. We set the parameters as follows.

$$\begin{aligned}\delta &:= (10^9 q)^{-10q^2},\\ m &:= c_m\log(1/\varepsilon)/\delta,\\ w_{\mathsf{proof}} &:= (300q/m)\log\ell, & W_{\mathsf{proof}} &:= 2^{w_{\mathsf{proof}}} = \ell^{O(\delta/\log(1/\varepsilon))},\\ h_{\mathsf{proof}} &:= (25q+1)\log\ell, & H_{\mathsf{proof}} &:= 2^{h_{\mathsf{proof}}} = \ell^{25q+1},\\ n_{\mathsf{hard}} &:= 20H_{\mathsf{proof}}\cdot\mathrm{poly}(s,\varepsilon^{-\log\log\ell}),\end{aligned}$$

$$
\begin{aligned}
T &:= H_{\mathsf{proof}} \cdot W_{\mathsf{proof}} / \log^{c_{\mathsf{tm}}}(\ell).\\
h_{\mathsf{input}} &:= \left(1 - \frac{\Theta(m^2 \log\log T)}{\log T}\right) h_{\mathsf{proof}}, & H_{\mathsf{input}} &:= 2^{h_{\mathsf{input}}} = H_{\mathsf{proof}}/\mathrm{poly}(2^{m^2}, \log \ell),\\
w_{\mathsf{input}} &:= \lceil \log \tilde{n}_{\mathsf{hard}} \rceil - h_{\mathsf{input}}, & W_{\mathsf{input}} &:= 2^{w_{\mathsf{input}}} = \mathrm{poly}(s, 2^{m^2}, \log \ell).\\
a &:= O(\log^2 W_{\mathsf{proof}}/(\varepsilon\delta)^2) = O(\log^2 \ell/\varepsilon^2),\\
A &:= O(\log W_{\mathsf{proof}}/(\varepsilon\delta)^2) = O(\log \ell/\varepsilon^2),\\
U &:= O(1/\varepsilon),\\
l &:= \log \ell.
\end{aligned}
$$

Here c_m and c_{tm} are sufficiently large constants, q is the query complexity of the smooth and rectangular PCPP in Theorem 2.5.11, and $\tilde{n}_{\mathsf{hard}} = \Theta(n_{\mathsf{hard}})$ is the length of $\mathsf{Enc}(x)$ when the length of x is n_{hard}. Let $\hat{H}_{\mathsf{proof}}$ be the number of rows of the PCPP proof in Theorem 2.5.11, and let $\hat{h}_{\mathsf{proof}} = \log \hat{H}_{\mathsf{proof}}$, then $\hat{h}_{\mathsf{proof}} = \log T + \Theta(m \log\log T) - w_{\mathsf{proof}}$. Also, let $r, r_{\mathsf{shared}}, r_{\mathsf{col}}, r_{\mathsf{row}}$ be the total, shared, column, row randomness in Theorem 2.5.11, respectively.

We use a different definition of "easy witness" as follows. We say x has an easy witness if there is a proof matrix π such that:

(completeness) for every $\mathsf{seed} \in \{0,1\}^r$, $\mathsf{VPCPP}^{\mathsf{Encode}(x)\circ\pi}(\mathsf{seed})$ accepts;

(*approximate* easiness) for every row π_i of π, there exists a size-s $\mathscr{C}$ circuit $C_i : \{0,1\}^n \to \{0,1\}$ and an advice $\alpha_i \in \{0,1\}^a$ such that the decoding of the string $C_i(w)$ with advice α_i is δ-close to π_i with respect to ℓ_1-norm. (Recall that $w = (w_1, w_2, \ldots, w_\ell)$ is our input and $C(w)$ denotes the concatenation of $C(w_1)$, $C(w_2)$, $\ldots$, $C(w_\ell)$.) In particular:

1. for every $j \in [W_{\mathsf{proof}}]$, $(\mathsf{dec}_{\alpha_i}(C_i(w)))_j \in [0,1]$;
2. $\|\mathsf{dec}_{\alpha_i}(C_i(w)) - \pi_i\|_1 \le \delta$.

Our machine guesses $\hat{H}_{\mathsf{proof}}$ size-s $\mathscr{C}$ circuits $C_1, C_2, \ldots, C_{\hat{H}_{\mathsf{proof}}} : \{0,1\}^n \to \{0,1\}$ as well as $\hat{H}_{\mathsf{proof}}$ advice strings $\alpha_1, \alpha_2, \ldots, \alpha_{\hat{H}_{\mathsf{proof}}}$. Let $\pi_i^{\mathsf{Real}} := \mathsf{dec}_{\alpha_i}(C_i(w))$, and π_i^{Bool} be the Boolean string that is closest to π_i^{Real}. For $\iota \in [q]$, we define

$$
\begin{aligned}
f^{\mathsf{Bool}}_{\mathsf{seed.shared},\iota}(\mathsf{seed.row}, \mathsf{seed.col}) &= \pi^{\mathsf{Bool}}_{\mathsf{irow}[\iota],\mathsf{icol}[\iota]} \quad \text{and}\\
f^{\mathsf{Real}}_{\mathsf{seed.shared},\iota}(\mathsf{seed.row}, \mathsf{seed.col}) &= \pi^{\mathsf{Real}}_{\mathsf{irow}[\iota],\mathsf{icol}[\iota]}.
\end{aligned}
$$

Verifying closeness of π^{Bool} and π^{Real}. The next lemma shows that we can verify whether a Boolean proof π^{Bool} and a real proof π^{Real} are close.

Lemma 3.7.4. *Let $\mathscr{C}$ be a typical circuit class and $d \ge 2$ be an even number. Suppose there is an algorithm that takes as inputs a list of $2^{r_{\mathsf{col}}}$ $\mathsf{AND}_{2d} \circ \mathscr{C}$ circuits $\{C_i\}$ and a list of $2^{r_{\mathsf{col}}}$ inputs $\{x_j\}$ of length $n \cdot \mathrm{polylog}(\ell)$, runs in deterministic T^{alg} time, and estimates the following quantity with additive error η:*

$$\Pr_{i,j \leftarrow [2^{r_{\mathsf{col}}}]}[C_i(x_j)].$$

Then there is an algorithm that takes the strings $w_1, w_2, \ldots, w_\ell$, circuits $(C_1, C_2, \ldots, C_{\hat{H}_{\mathsf{proof}}})$, and $(\alpha_1, \alpha_2, \ldots, \alpha_{\hat{H}_{\mathsf{proof}}})$ as inputs, runs in $O((3A)^{2d} T^{\mathrm{alg}}) \cdot (2^{2dl + r_{\mathsf{shared}}} + T \log^{O(m)} T / 2^{2r_{\mathsf{col}}}))$ time

deterministically, and satisfies the following:

(Completeness) *If for every $i \in [\hat{H}_{\mathsf{proof}}]$, it holds that (1) for every $j \in [W_{\mathsf{proof}}]$, $\pi^{\mathsf{Real}}_{i,j} \in [0,1]$; (2) $\|\pi^{\mathsf{Real}}_i - \pi^{\mathsf{Bool}}_i\|_1 \le \delta$, then the algorithm accepts.*

(Soundness) *If the algorithm accepts, then it holds that*

1. *for every $\mathsf{seed.shared} \in \{0,1\}^{r_{\mathsf{shared}}}$ and $\iota \in [q]$, $\|f^{\mathsf{Real}}_{\mathsf{seed.shared},\iota}\|_d^d \le 1 + 2\eta \cdot U^d$;*
2. $\mathbb{E}_{i \leftarrow [\hat{H}_{\mathsf{proof}}], j \leftarrow [W_{\mathsf{proof}}]} \left[|\pi^{\mathsf{Real}}_{i,j} - \pi^{\mathsf{Bool}}_{i,j}|^d \right] \le 4^d \cdot \delta + 2^{d+1}\eta(2U+1)^{2d}$.

Proof Sketch. We first estimate $\|f^{\mathsf{Real}}_{\mathsf{seed.shared},\iota}\|_d^d$ for fixed $\mathsf{seed.shared}$ and ι. Recall that

$$\pi^{\mathsf{Real}}_{i,j} = \sum_{k \in [A]} \mathsf{coeff}_k(\alpha_i) \cdot C_i(w_{\mathsf{idx}_k(\alpha_i, j)}).$$

We build a $\mathsf{Prod}_d \circ \mathsf{Sum}$ circuit $C_{\mathsf{norm}} := C_{\mathsf{norm}}(\mathsf{seed.shared}, \iota)$ as follows.

Circuit C_{norm}

(Inputs) The input consists of (y, α) with the intended meaning that $y = (y_1, y_2, \ldots, y_\ell)$ where $y_i = C_{\mathsf{irow}[\iota]}(w_i)$, and $\alpha = \alpha_{\mathsf{irow}[\iota]}$.

(Linear sum gates) There are $2^{r_{\mathsf{col}}}$ linear sum gates. For each $\mathsf{seed.col}$,

$$\mathsf{Sum}_{\mathsf{seed.col}}(y, \alpha) = \sum_{k \in [A]} \mathsf{coeff}_k(\alpha) \cdot y_{\mathsf{idx}_k(\alpha, \mathsf{icol}[\iota])}.$$

(Output product gates) There are $2^{r_{\mathsf{col}}}$ product gates. For each $\mathsf{seed.col}$, the $\mathsf{seed.col}$-th output gate is simply

$$(C_{\mathsf{norm}})_{\mathsf{seed.col}}(y, \alpha) = (\mathsf{Sum}_{\mathsf{seed.col}}(y, \alpha))^d.$$

Recall that this circuit C_{norm} has parameters as follows:

- the number of gates in each layer: $\ell_{\mathsf{Sum}} = 2^{r_{\mathsf{col}}}$, $\ell_{\mathsf{Prod}} = 2^{r_{\mathsf{col}}}$;
- the fan-in of the top Prod gates d;
- the fan-in A, coefficient sum U, and locality l of the linear sum layer.

We invoke Lemma 3.7.3 on the circuit C_{norm}, strings $w_1, w_2, \ldots, w_\ell$, $2^{r_{\mathsf{row}}}$ inputs $\{\alpha_{\mathsf{irow}[\iota]}\}_{\mathsf{seed.row}}$, and $2^{r_{\mathsf{row}}}$ size-s $\mathscr{C}$ circuits $\{C_{\mathsf{irow}[\iota]}\}_{\mathsf{seed.row}}$. Here $\ell_{\mathscr{C}} = 1$. We thus obtain an estimation $\mathsf{EST}_{\mathsf{norm}} = \mathsf{EST}_{\mathsf{norm}}(\mathsf{seed.shared}, \iota)$ where

$$\left| \mathsf{EST}_{\mathsf{norm}} - \|f^{\mathsf{Real}}_{\mathsf{seed.shared},\iota}\|_d^d \right| \le \eta \cdot U^d.$$

If $\mathsf{EST}_{\mathsf{norm}} > 1 + \eta \cdot U^d$, then we reject the input. Otherwise, we proceed to verify that π^{Real} and π^{Bool} are close. Consider the polynomial $P(z) := z^d(1-z)^d$. We will estimate

$$\mathop{\mathbb{E}}_{i \leftarrow [\hat{H}_{\mathsf{proof}}], j \leftarrow [W_{\mathsf{proof}}]} \left[P(\pi^{\mathsf{Real}}_{i,j}) \right]. \tag{3.15}$$

Similarly, we estimate (3.15) by building a $\mathsf{Prod}_{2d} \circ \mathsf{Sum}$ circuit C_{diff}.

Circuit C_{diff}

(Inputs) The inputs are exactly the same as C_{norm}.

(Linear sum gates) There are $2W_{\mathsf{proof}}$ linear sum gates. Let $j \in [W_{\mathsf{proof}}]$, then the $2j$-th linear sum gate computes $(\pi^{\mathsf{Real}})_j$, and the $(2j+1)$-th one computes $1-(\pi^{\mathsf{Real}})_j$. That is,

$$\mathsf{Sum}_{2j}(y,\alpha) = \sum_{k\in[A]} \mathsf{coeff}_k(\alpha)\cdot y_{\mathsf{idx}_k(\alpha,j)}; \quad \mathsf{Sum}_{2j+1}(y,\alpha) = 1-\mathsf{Sum}_{2j}(y,\alpha).$$

The implementation of the linear sum layer is the same as in Lemma 3.5.4, and we omit it here.

(Output product gates) There are W_{proof} product gates. For each $j \in [W_{\mathsf{proof}}]$, the j-th output gate is

$$C_{\mathsf{diff}}(y,\alpha) = (\mathsf{Sum}_{2j}(y,\alpha)\cdot \mathsf{Sum}_{2j+1}(y,\alpha))^d.$$

The parameters of the circuit C_{diff} are as follows:

- the number of gates in each layer: $\ell_{\mathsf{Sum}} = 2W_{\mathsf{proof}}$, $\ell_{\mathsf{Prod}} = W_{\mathsf{proof}}$;
- the fan-in of the top Prod gates $2d$;
- the fan-in $2A+1$, coefficient sum $2U+1$, and locality l of the linear sum layer.

We invoke Lemma 3.7.3 on the circuit C_{diff}, strings $w_1, w_2, \ldots, w_\ell$, a list of $\hat{H}_{\mathsf{proof}}$ inputs $\{\alpha_i\}$, and a list of $\hat{H}_{\mathsf{proof}}$ size-s $\mathscr{C}$ circuits $\{C_i\}$. Here $\ell_{\mathscr{C}} = 1$. We obtain an estimation $\mathsf{EST}_{\mathsf{diff}}$ where

$$|\mathsf{EST}_{\mathsf{diff}} - (3.15)| \le \eta\cdot(2U+1)^{2d}.$$

We accept if and only if $\mathsf{EST}_{\mathsf{diff}} \le 2^d\cdot\delta + \eta(2U+1)^{2d}$.

The correctness and complexity are analysed in the same way as in Lemma 3.5.4, so we omit it here. ◇

Estimating p_{acc}. Now we verified that π^{Real} is close to π^{Bool} using Lemma 3.7.4, with parameter $d = 2q$. After that, the next step is to use it to speed up L^{hard}. We estimate

$$p_{\mathsf{acc}} := \Pr_{\mathsf{seed}\leftarrow\{0,1\}^r}\left[\mathsf{VPCPP}^{\mathsf{Enc}(x)\circ\pi^{\mathsf{Bool}}}(\mathsf{seed}) \text{ accepts}\right].$$

Actually, it suffices to distinguish between the case that $p_{\mathsf{acc}} > 5/6$ and the case that $p_{\mathsf{acc}} < 1/2$.

We still enumerate $\mathsf{seed.shared}$, and we now need to estimate

$$p_{\mathsf{acc}}(\mathsf{seed.shared}) := \Pr_{\substack{\mathsf{seed.row}\leftarrow\{0,1\}^{r_{\mathsf{row}}}\\ \mathsf{seed.col}\leftarrow\{0,1\}^{r_{\mathsf{col}}}}}\left[\mathsf{VPCPP}^{\mathsf{Enc}(x)\circ\pi^{\mathsf{Bool}}}(\mathsf{seed}) \text{ accepts}\right].$$

Let $pc_1, pc_2, \ldots, pc_q \leftarrow V_{\mathsf{pc}}(\mathsf{seed.shared})$ be the parity-check bits of the PCPP verifier, and let pc_ι^{row} (resp. pc_ι^{col}) denote the contribution of $\mathsf{seed.row}$ (resp. $\mathsf{seed.col}$) to pc_ι, then $pc_\iota = pc_\iota^{\mathsf{row}} \oplus pc_\iota^{\mathsf{col}}$.

As in the proof of Theorem 3.5.1, here it suffices to estimate for every $S, S' \subseteq [q]$

$$p_{\mathsf{acc}}^{\mathsf{Real}}(\mathsf{seed.shared}, S, S') = \mathop{\mathbb{E}}_{\substack{\mathsf{seed.row}\leftarrow\{0,1\}^{r_{\mathsf{row}}}\\ \mathsf{seed.col}\leftarrow\{0,1\}^{r_{\mathsf{col}}}}}\left[\prod_{\iota\in S} a_\iota^{\mathsf{Real}}\cdot\prod_{\iota\in S'} pc_\iota\right],$$

where

$$a_\iota^{\mathsf{Real}} := \begin{cases} \tilde{x}_{\mathsf{irow}[\iota],\mathsf{icol}[\iota]} & \mathsf{itype}[\iota] = \mathsf{input},\\ \pi^{\mathsf{Real}}_{\mathsf{irow}[\iota],\mathsf{icol}[\iota]} & \mathsf{itype}[\iota] = \mathsf{proof}. \end{cases}$$

We want to invoke Lemma 3.7.3 to estimate this, so we want to construct a $2^{r_{\mathsf{col}}}$-output $\mathsf{Prod}\circ\mathsf{Sum}$ circuit C^{Prod}, $2^{r_{\mathsf{row}}}$ circuits $\{C_{\mathsf{seed.row}}\}$ and $2^{r_{\mathsf{row}}}$ strings $\{\alpha_{\mathsf{seed.row}}\}$ such that

$$\begin{aligned}
&C^{\mathsf{Prod}}_{\mathsf{seed.col}}(C_{\mathsf{seed.row}}(w), \alpha_{\mathsf{seed.row}}) \\
&= \prod_{\iota\in S} a^{\mathsf{Real}}_{\iota} \cdot \prod_{\iota\in S'} pc_{\iota} \\
&= \prod_{\iota\in S^{\mathsf{proof}}} \left(\sum_{k\in[A]} \mathsf{coeff}_k(\alpha_{\mathsf{irow}[\iota]}) \cdot C_{\mathsf{irow}[\iota]}\left(w_{\mathsf{idx}_k(\alpha_{\mathsf{irow}[\iota]},\mathsf{icol}[\iota])}\right) \right) \cdot \prod_{\iota\in S^{\mathsf{input}}} \tilde{x}_{\mathsf{irow}[\iota],\mathsf{icol}[\iota]} \cdot \prod_{\iota\in S'} (pc^{\mathsf{row}}_{\iota} \oplus pc^{\mathsf{col}}_{\iota})
\end{aligned} \tag{3.16}$$

where $S^{\mathsf{proof}} = \{\iota \in S : \mathsf{itype}[\iota] = \mathsf{proof}\}$ and $S^{\mathsf{input}} = \{\iota \in S : \mathsf{itype}[\iota] = \mathsf{input}\}$. This motivates the following definitions.

For $i \in [\ell]$, let $z_i \in \{0,1\}^{n+w_{\mathsf{input}}}$ be the string such that the first n bits of z_i is w_i, and the last w_{input} bit is (the binary expression of)

$$\begin{cases} i & \text{if } i \in [W_{\mathsf{input}}], \\ 1 & \text{if } i \notin [W_{\mathsf{input}}]. \end{cases}$$

Here we identify $[W_{\mathsf{input}}]$ with $\{0,1\}^{w_{\mathsf{input}}}$.

For any string v, define $\hat{C}_v : \{0,1\}^{\lceil \log |v| \rceil} \to \{0,1\}$ as the circuit that on input $i \le |v|$, outputs v_i. Since $\mathscr{C}$ is complete, $\hat{C}_v$ is an efficiently computable $\mathscr{C}$ circuit of size $\mathrm{poly}(|v|)$. Let $\mathsf{proj}_{\mathsf{input}} : \{0,1\}^{n+w_{\mathsf{input}}} \to \{0,1\}^{w_{\mathsf{input}}}$ be the circuit that outputs the last w_{input} bits of its input, and let $\mathsf{proj}_{\mathsf{proof}} : \{0,1\}^{n+w_{\mathsf{input}}} \to \{0,1\}^{n}$ be the circuit that outputs the first n bits of its input.

Now for fixed $\mathsf{seed.row}$, define

$$C^{a}_{\iota} := \begin{cases} \hat{C}_{\tilde{x}_{\mathsf{irow}[\iota]}} \circ \mathsf{proj}_{\mathsf{input}} & \mathsf{itype}[\iota] = \mathsf{input} \\ C_{\mathsf{irow}[\iota]} \circ \mathsf{proj}_{\mathsf{proof}} & \mathsf{itype}[\iota] = \mathsf{proof} \end{cases}$$

for $\iota \in S$, and C^{pc}_{ι} be the circuit that outputs $pc^{\mathsf{row}}_{\iota}$ for $\iota \in S'$, regardless of its input. Let $C_{\mathsf{seed.row}} := (C^a_1, C^a_2, \ldots, C^a_{d_S}, C^{pc}_1, C^{pc}_2, \ldots, C^{pc}_{d_{S'}})$ where $d_S := |S|$ and $d_{S'} := |S'|$, that is, $C_{\mathsf{seed.row}}$ is a circuit with $d_S + d_{S'}$ outputs and each of its outputs is a circuit C^a_ι or C^{pc}_ι.

Now we define the $\mathsf{Prod}\circ\mathsf{Sum}$ circuit C^{Prod}.

Circuit C^{Prod}

(Inputs) The input y has the form $y = y_{\mathsf{seed.row}} = (y_1, y_2, \ldots, y_\ell)$ and the input $\hat{\alpha}$ has the form $\hat{\alpha} = \hat{\alpha}_{\mathsf{seed.row}} = (\hat{\alpha}_1, \hat{\alpha}_2, \ldots, \hat{\alpha}_{d_S})$. The intended meanings are $y_i = C_{\mathsf{seed.row}}(z_i)$, and $\hat{\alpha}_i = \alpha_{\mathsf{irow}[i]}$.

For convenience, we will use the following labels to refer to bits of y, assuming the intended meaning above:

- For $j \in S^{\mathsf{proof}}$, $i \in [\ell]$, let $(y_i)_j := C^a_j(z_i) = C_{\mathsf{irow}[j]}(w_i)$;
- For $j \in S^{\mathsf{input}}$, $i \in [W_{\mathsf{input}}]$, let $(y_i)_j := C^a_j(z_i) = \hat{C}_{\tilde{x}_{\mathsf{irow}[j]}}(i) = \tilde{x}_{\mathsf{irow}[j],i}$;
- For $j \in S'$, let $(y_i)_{j+d_S} := C^{pc}_j(z_i) = pc^{\mathsf{row}}_j$.

(Linear sum gates) There are $\ell_{\mathsf{Sum}} := W_{\mathsf{proof}} \cdot d_S + 2d_{S'}$ linear sum gates and we identify $[\ell_{\mathsf{Sum}}]$ with the disjoint union of $[W_{\mathsf{proof}}] \times S$ and $S' \times \{0,1\}$.

Let $i \in [W_{\mathsf{proof}}]$ and $j \in S$. If $\mathsf{itype}[j] = \mathsf{proof}$, then the (i,j)-th linear sum gate is

$$\mathsf{Sum}_{(i,j)}(y,\alpha) = \sum_{k\in[A]} \mathsf{coeff}_k(\alpha_j) \cdot (y_{\mathsf{idx}_k(\alpha_j,i)})_j.$$

It is easy to verify that

$$\mathsf{Sum}_{(i,j)}(y,\alpha) = (\mathsf{dec}_{\alpha_{\mathsf{irow}[j]}}(C_{\mathsf{irow}[j]}(w)))_i.$$

On the other hand, if $\mathsf{itype}[j] = \mathsf{input}$, then the (i,j)-th linear sum gate is $\mathsf{Sum}_{(i,j)}(y,\alpha) := (y_i)_j$. (If $i > W_{\mathsf{input}}$ then we simply set $\mathsf{Sum}_{(i,j)}(y,\alpha) = 0$ and this gate would not be used.)

Finally, for each $j \in S'$, we have two intermediate gates

$$\mathsf{Sum}_{(j,0)}(y,\alpha) = (y_1)_{j+d_S}, \quad \mathsf{Sum}_{(j,1)}(y,\alpha) = 1 - (y_1)_{j+d_S}.$$

Implementation of the linear sum layer: The linear sum has fan-in $A' := A \cdot d_S + 2$ and we identify $[A']$ with the disjoint union of $[A] \times S$ and $\{+,-\}$. Also, the length of y is $\ell_y := \ell \cdot (d_S + d_{S'})$, and we identify $[\ell_y]$ with $[\ell] \times (S \cup S')$. Let idx' and coeff' be the idx and coeff functions of the linear sum layer of C^{Prod}, then

(Function $\mathsf{idx}'_k(\alpha,i)$) Suppose $i = (i',j) \in [W_{\mathsf{proof}}] \times S$. If $\mathsf{itype}[j] = \mathsf{proof}$ and $k = (k',j')$ where $j = j'$, then we return $\mathsf{idx}'_k(\alpha,i) = (\mathsf{idx}_{k'}(\alpha_j,i'),j)$; if $\mathsf{itype}[j] = \mathsf{input}$ and $i' \in [W_{\mathsf{input}}]$ and $k = +$, then $\mathsf{idx}'_k(\alpha,i) = (i',j)$. Otherwise $\mathsf{idx}'_k(\alpha,i) = \mathsf{ZERO}$.

On the other hand, suppose $i = (j,b) \in S' \times \{0,1\}$. If ($b = 0$ and $k = +$) or ($b = 1$ and $k = -$) then $\mathsf{idx}'_k(\alpha,i) = (1,j)$. If $b = 1$ and $k = +$ then $\mathsf{idx}'_k(\alpha,i) = \mathsf{ONE}$. Otherwise $\mathsf{idx}'_k(\alpha,i) = \mathsf{ZERO}$.

(Function $\mathsf{coeff}'_k(\alpha)$) If $k = +$ then $\mathsf{coeff}'_k(\alpha) = 1$; if $k = -$ then $\mathsf{coeff}'_k(\alpha) = -1$; otherwise, if $k = (k',j')$ then $\mathsf{coeff}'_k(\alpha) = \mathsf{coeff}_{k'}(\alpha_{j'})$.

The locality of $(\mathsf{idx}', \mathsf{coeff}')$ is still l. The coefficient sum becomes $d_S \cdot U + 2$.

(Output product gates) There are $2^{r_{\mathsf{col}}}$ product gates. For each $\mathsf{seed.col}$, the $\mathsf{seed.col}$-th output gate is

$$C^{\mathsf{Prod}}_{\mathsf{seed.col}}(y,\alpha) = \prod_{j\in S} \mathsf{Sum}_{(\mathsf{icol}[j],j)}(y,\alpha) \cdot \prod_{j\in S'} \mathsf{Sum}_{(j,pc^{\mathsf{col}}_j)}(y,\alpha).$$

To summarise, the parameters of the circuit C^{Prod} are as follows.

- The number of gates in each layer: $\ell_{\mathsf{Sum}} = W_{\mathsf{proof}} \cdot d_S + 2d_{S'}$, $\ell_{\mathsf{Prod}} = 2^{r_{\mathsf{col}}}$.
- The length of input y: $\ell_y = \ell(d_S + d_{S'})$;
- The fan-in of the top Prod gates: $d_S + d_{S'} \le 2q$.
- The fan-in $A' := A \cdot d_S + 2$, coefficient sum $d_S \cdot U + 2$, and locality l of the linear sum layer.

Given the above construction, it is easy to see that (3.16) holds for every $\mathsf{seed.row}$ and $\mathsf{seed.col}$. We can thus see that

$$p^{\mathsf{Real}}_{\mathsf{acc}}(\mathsf{seed.shared}, S, S') = \mathop{\mathbb{E}}_{\substack{\mathsf{seed.row}\leftarrow\{0,1\}^{r_{\mathsf{row}}} \\ \mathsf{seed.col}\leftarrow\{0,1\}^{r_{\mathsf{col}}}}} \left[C^{\mathsf{Prod}}_{\mathsf{seed.col}}(C_{\mathsf{seed.row}}(w), \alpha_{\mathsf{seed.row}}) \right].$$

Since $d_S \le q, d_{S'} \le q$ and $\ell_{\mathscr{C}} \le d_S + d_{S'} \le 2q$, we can estimate $p^{\mathsf{Real}}_{\mathsf{acc}}(\mathsf{seed.shared}, S, S')$ using

Lemma 3.7.3 within an additive error of $\eta \cdot (qU+2)^{2q}$ in deterministic time

$$(A \cdot d_S + 2)^{2q} \cdot ((2q)^{2q} + 2^{r_{\mathsf{col}}}/N) \cdot (2^{2ql} + 2^{r_{\mathsf{row}}}/N) \cdot O(T^{\mathrm{alg}})$$

Analysis. First, the verification step takes

$$\begin{aligned}
&O((3A)^{4q} T^{\mathrm{alg}}) \cdot \left(2^{4ql+r_{\mathsf{shared}}} + T \log^{O(m)} T / 2^{2r_{\mathsf{col}}}\right) \\
\le\, &O((3A)^{4q}) \cdot (T \log^{O(m)} T)/(r_{\mathsf{col}})^{c_u \log(1/\varepsilon)} \\
\le\, &T(\log T)^{O(m) - c_u \log(1/\varepsilon)/2}
\end{aligned}$$

time, which is at most $T/(4\log^{c_{\mathsf{hard}}} T)$ if c_u is a large enough constant.

Then, the algorithm of Lemma 3.7.3 on C^{Prod} runs for every $\mathsf{seed.shared}, S, S'$, and in total takes time

$$\begin{aligned}
&(A \cdot d_S + 2)^{2q} \cdot ((2q)^{2q} + 2^{r_{\mathsf{col}}}/N) \cdot (2^{2ql} + 2^{r_{\mathsf{row}}}/N) \cdot O(T^{\mathrm{alg}}) \cdot 2^{2q} \cdot 2^{r_{\mathsf{shared}}} \\
\le\, &O(\log^{2q} \ell/\varepsilon^{4q}) \cdot O(2^{r_{\mathsf{col}}}/N) \cdot O(2^{r_{\mathsf{row}}}/N) \cdot O(N^2/\log^{c_u \log(1/\varepsilon)} N) \cdot 2^{r_{\mathsf{shared}}} \\
\le\, &2^r/\log^{\Omega(c_u)} \ell < T/(4\log^{c_{\mathsf{hard}}} T),
\end{aligned}$$

when c_u is sufficiently large. Therefore, the whole algorithm runs in $T/\log^{c_{\mathsf{hard}}} T$ time.

Besides, by the same argument as in the proof of Theorem 3.5.1, which we omit here, the algorithm estimates p_{acc} within an additive error of at most

$$\eta(4qU+8)^{2q} + 16^q \cdot 100^{-q} < 1/6,$$

thus successfully distinguishes between the case that $p_{\mathsf{acc}} > 5/6$ and that $p_{\mathsf{acc}} < 1/2$.

Description of M^{PCPP}. We summarise the algorithm M^{PCPP}. On input x, we consider the smooth and rectangular PCPP for the language $L^{\mathsf{enc}} = \{\mathsf{Encode}(x) : x \in L^{\mathsf{hard}}\}$. (Recall that M^{PCPP} aims to reject every $x \notin L$ and accepts every $x \in L$ with easy witness.) We guess $(C_1, \ldots, C_{H_{\mathsf{proof}}})$ and $(\alpha_1, \ldots, \alpha_{H_{\mathsf{proof}}})$, which implicitly defines the PCPP proof matrices π^{Bool} and π^{Real}. Then we verify π^{Real} using Lemma 3.5.4 and reject immediately if π^{Real} did not pass the test. If π^{Real} passes the test (which means that it is "close" to a Boolean proof π^{Bool}), we use the algorithm described above to estimate p_{acc}. We accept x if and only if our estimation is above $2/3$.

The correctness of M^{PCPP} is easy to see (and is exactly the same as Claim 3.5.8):

Claim 3.7.5. *For every input x, if $x \notin L$ then M^{PCPP} rejects x; while if $x \in L$ and x has an easy witness then M^{PCPP} accepts x.*

The machine M^{PCPP} guesses $\hat{H}_{\mathsf{proof}}(5s\log s + a) < n_{\mathsf{hard}}/10$ bits of nondeterminism (the number of size-s $\mathscr{C}$ circuits is at most $2^{5s\log s}$), and uses $\ell^{4q} < n_{\mathsf{hard}}/10$ bits of advice. Thus it computes a language in $\mathsf{NTIMEGUESS}_{\mathsf{RAM}}[T/\log^{c_{\mathsf{hard}}}(T), n_{\mathsf{hard}}/10]_{/(n_{\mathsf{hard}}/10)}$.

The $\mathsf{FP}^{\mathsf{NP}}$ algorithm for average-case hard partial truth tables. Let $w_1, w_2, \ldots, w_\ell \in \{0,1\}^n$ be the input. We first construct the hard language L^{hard} and the algorithm M^{PCPP}.

Since M^{PCPP} is a nondeterministic RAM algorithm that runs in $T/\log^{c_{\mathsf{hard}}}(T)$ time, uses at most $n_{\mathsf{hard}}/10$ nondeterministic bits and at most $n_{\mathsf{hard}}/10$ advice bits, it follows that there is an input $x_{\mathsf{hard}} \in \{0,1\}^{n_{\mathsf{hard}}}$ such that $M^{\mathsf{PCPP}}(x_{\mathsf{hard}}) \neq L^{\mathsf{hard}}(x_{\mathsf{hard}})$. Moreover, let α be the advice string fed to M^{PCPP}, i.e., the circuit C. We can find such an input x_{hard} by running $\mathcal{R}(1^{n_{\mathsf{hard}}}, M^{\mathsf{PCPP}}, \alpha)$, where $\mathcal{R}$ is the refuter guaranteed by Theorem 3.3.1. Thus, we can find x_{hard} in deterministic $\mathrm{poly}(T)$ time with an NP oracle.

It follows from Claim 3.7.5 that $x_{\mathsf{hard}} \in L^{\mathsf{hard}}$ but x_{hard} does not have an easy witness. Thus, we can use the NP oracle to find the lexicographically first PCPP proof matrix π such that

$$\Pr_{\mathsf{seed} \leftarrow \{0,1\}^r}[\mathsf{VPCPP}^{\mathsf{Encode}(x) \circ \pi}(\mathsf{seed}) \text{ accepts}] = 1.$$

Then, there must exist a row π_i such that $\mathsf{Amp}(\pi_i)$ is $(1/2-\varepsilon)$-far from $C(w) = C(w_1) \circ \cdots \circ C(w_\ell)$ for any size-s $\mathscr{C}$ circuit C. To see this, suppose that for every i, there exists a size-s $\mathscr{C}$ circuit C_i such that $\mathsf{Amp}(\pi_i)$ is $(1/2-\varepsilon)$-close to $C_i(w)$. By Theorem 3.3.5, there is an advice α_i such that $\mathsf{dec}_{\alpha_i}(C_i(w))$ satisfies (1) for every $j \in [W_{\mathsf{proof}}]$, $(\mathsf{dec}_{\alpha_i}(C_i(w)))_j \in [0,1]$; (2) $\|\mathsf{dec}_{\alpha_i}(C_i(w)) - \pi_i\|_1 \leq \delta$. It follows that π is an easy witness for x_{hard}, a contradiction.

Finally, we use the NP oracle to find the first row π_i, such that $\mathsf{Amp}(\pi_i)$ is $(1/2-\varepsilon)$-far from $C(w)$ for any size-s $\mathscr{C}$ circuit C, and output $\mathsf{Amp}(\pi_i)$. The overall procedure takes deterministic $\mathrm{poly}(T) \leq \mathrm{poly}(\ell)$ time with an NP oracle. □

3.8 Unconditional Algorithms for Range Avoidance

In this section, we apply the frameworks above to obtain *unconditional* results for ACC^0-Remote-Point and ACC^0-Partial-AvgHard.

3.8.1 An Algorithm for #ACC^0-Satisfying-Pairs

We first present a non-trivial algorithm for #ACC^0-Satisfying-Pairs. This algorithm utilises a quasi-polynomial simulation of $\mathsf{SYM} \circ \mathsf{ACC}^0$ circuits by $\mathsf{SYM} \circ \mathsf{AND}$ circuits.

We need the following algorithm for the batch evaluation of low-degree polynomials via fast rectangular matrix multiplication. This algorithm has been extensively used in previous works on the polynomial method and circuit complexity (see, e.g., [Wil14, Wil18a]). We provide a proof for completeness.

Theorem 3.8.1. *Let $x_1, x_2, \ldots, x_N \in \{0,1\}^n$ be N input strings, and $p_1, p_2, \ldots, p_N : \{0,1\}^n \to \mathbb{N}$ be N integer polynomials of degree at most d. Suppose that $n^{20d} \leq N$. Then there is a deterministic algorithm running in $\tilde{O}(N^2)$ time that outputs the table of $p_j(x_i)$ for every $i, j \in [N]$.*

Theorem 3.8.2 ([Cop82]; see also [Wil18a, Appendix C])**.** *There is a (deterministic) algorithm for multiplying an $N \times N^{0.1}$ matrix and an $N^{0.1} \times N$ matrix using $\tilde{O}(N^2)$ arithmetic operations.*

Proof of Theorem 3.8.1. There are $m := \sum_{i=0}^{d} \binom{n}{i} \leq (en/d)^d \leq N^{0.1}$ monomials of degree at most d. We number these monomials from 1 to m. Let S_j denote the set of indices in the j-th monomial. That is, the j-th monomial is $\prod_{k \in S_j} x_k$.

We construct two matrices $M_1 \in \mathbb{Z}^{N\times m}$ and $M_2 \in \mathbb{Z}^{m\times N}$. For each $i \in [N]$ and $j \in [m]$, $M_1[i,j]$ is the evaluation of the j-th monomial on input x_i. (That is, $M_1[i,j] = \prod_{k\in S_j}(x_i)_k$.) For each $j \in [m]$ and $k \in [N]$, $M_2[j,k]$ is the coefficient of the j-th monomial in p_k.

Let $M := M_1 \cdot M_2$. It follows that for every $i, j \in [N]$, $M[i,j] = p_j(x_i)$. Since $m \le N^{0.1}$, we can compute M in $\tilde{O}(N^2)$ time using Theorem 3.8.2. □

Theorem 3.8.3 (From $\mathsf{SYM} \circ \mathsf{ACC}^0$ to $\mathsf{SYM} \circ \mathsf{AND}$ [BT94, AG91, Wil18c])**.** *Let m, ℓ be any constants, there exists an integer c' such that every $\mathsf{SYM}\circ\mathsf{AC}^0_\ell[m]$ circuit of size s can be simulated by a $\mathsf{SYM} \circ \mathsf{AND}$ circuit of $2^{(\log s)^{c'}}$ size. Moreover, the AND gates of the final circuit have only $(\log s)^{c'}$ fan-in, the final circuit can be constructed from the original one in $2^{O((\log s)^{c'})}$ time, and the final symmetric function at the output can be computed in $2^{O((\log s)^{c'})}$ time.*

Combining Theorem 3.8.3 with Theorem 3.8.1, we can derive the $\#\mathsf{ACC}^0$-Satisfying-Pairs algorithm in non-trivial time as follows.

Theorem 3.1.15. *For every constants m, ℓ, c, there is a constant $\varepsilon \in (0,1)$ such that the following holds. Let $n := 2^{\log^\varepsilon N}$ and $s := 2^{\log^c n}$. There is a deterministic algorithm running in $\tilde{O}((N/n)^2)$ time that given N strings $x_1, x_2, \ldots, x_N \in \{0,1\}^n$ and N $\mathsf{AC}^0_\ell[m]$ circuits $C_1, C_2, \ldots, C_N : \{0,1\}^n \to \{0,1\}$ of size s, outputs the number of pairs $(i,j) \in [N] \times [N]$ such that $C_i(x_j) = 1$.*

Proof. Let ε be a constant to be determined. We divide $C_1, C_2, \ldots, C_N$ into N/n groups where each group has size n. Let C_{ij} denote the j-th circuit in the i-th group. We also partition the inputs $x_1, x_2, \ldots, x_N$ into N/n groups of size n and define x_{ij} similarly. Let $X_i := x_{i1} \circ x_{i2} \circ \cdots \circ x_{in}$.

For each group i, we can construct $g := \lceil 2\log n \rceil$ $\mathsf{SYM} \circ \mathsf{AC}^0_\ell[m]$ circuits $D_{i1}, D_{i2}, \ldots, D_{ig} : \{0,1\}^{n^2} \to \{0,1\}$, each of size $s' := O(n^2 \cdot s)$, such that for any group j, we have:

$$\sum_{i'=1}^{n}\sum_{j'=1}^{n} C_{ii'}(x_{jj'}) = \sum_{k=0}^{g} 2^k D_{ik}(X_j).$$

That is, $D_{ik}(X_j)$ computes the k-th bit of the number of satisfying pairs between the i-th group of circuits and the j-th group of inputs.

Let c' be the constant in Theorem 3.8.3 depending on ℓ and m. We can transform each $\mathsf{SYM} \circ \mathsf{AC}^0_\ell[m]$ circuit D_{ij} into a $\mathsf{SYM} \circ \mathsf{AND}$ circuit D'_{ij} of size $2^{(\log s')^{c'}}$ such that each AND gate has fan-in at most $d := (\log s')^{c'}$. We can write each $D'_{ij}(x)$ as $f_{ij}(p_{ij}(x))$, where $p_{ij}(x) : \{0,1\}^{n^2} \to \{0,1,\ldots,2^{(\log s')^{c'}}\}$ is a polynomial of degree at most d that only outputs integers upper bounded by $2^{(\log s')^{c'}}$ on Boolean inputs, and f_{ij} is some function that can be evaluated in $2^{O((\log s')^{c'})}$ time. We can construct the polynomials p_{ij} and (the truth tables of) the functions f_{ij} in $(N/n)g2^{O((\log s')^{c'})}$ time. Let $\varepsilon := 1/(10cc')$ (and recall $n = 2^{\log^\varepsilon N}$ and $s = 2^{\log^c n}$), this time bound is at most $(N/n)^2$.

Then, for each $k = 1, 2, \ldots, g$, since $(n^2)^{20d} \le N/n$, we can compute the table of $p_{ik}(X_j)$ for every $i, j \in [N/n]$ in $\tilde{O}((N/n)^2)$ time by invoking Theorem 3.8.1. In fact, by checking the

truth-tables of f_{ij}, we actually get the table for $D'_{ik}(X_j) = D_{ik}(X_j)$. Finally, it follows that:

$$\sum_{i=1}^{N}\sum_{j=1}^{N} C_i(x_j) = \sum_{i=1}^{N/n}\sum_{j=1}^{N/n}\sum_{i'=1}^{n}\sum_{j'=1}^{n} C_{ii'}(x_{jj'}) = \sum_{i=1}^{N/n}\sum_{j=1}^{N/n}\sum_{k=0}^{g} 2^k D_{ik}(X_j).$$

The total run-time is bounded by $2(N/n)^2 + g\tilde{O}((N/n)^2) = \tilde{O}((N/n)^2)$. □

3.8.2 Remote Point for ACC^0

Theorem 3.1.16 (ACC^0-Remote-Point $\in \mathsf{FP}^{\mathsf{NP}}$)**.** *There is a constant $c_u \geq 1$ such that for every constant $d, m \geq 1$, there is a constant $c_{\mathsf{str}} := c_{\mathsf{str}}(d, m) \geq 1$, such that the following holds.*

Let $n < s(n) \leq 2^{n^{o(1)}}$ be a size parameter, $\varepsilon := \varepsilon(n) \geq 2n^{-c_u}$ be an error parameter and $\ell := \ell(n) \geq 2^{\log^{c_{\mathsf{str}}} s}$ be a stretch function, then there is an $\mathsf{FP}^{\mathsf{NP}}$ algorithm that takes as input a circuit $C : \{0,1\}^n \to \{0,1\}^\ell$, where each output bit of C is computed by an $\mathsf{AC}^0_d[m]$ circuit of size s, and outputs a string y that is $(1/2 - \varepsilon)$-far from Range(C).

Proof. Let c_u be the constant from Theorem 3.5.1, and c_{str} be a constant to be determined later. We then set parameters for invoking Theorem 3.5.1.

We set $n_{\mathsf{sat}} := \max\{2^{\log^{c_u+2} n}, 2^{\log^{c_u+1} s}\}$. Then we can invoke Theorem 3.1.15 with input length n_{sat} and size parameter also n_{sat} to get a $\#\mathsf{AC}^0_{d+1}[m]$-Satisfying-Pairs algorithm for N circuits and N inputs, where $N := N(n) = 2^{\log^{1/\varepsilon_{\mathsf{sat}}} n_{\mathsf{sat}}}$ for some constant $\varepsilon_{\mathsf{sat}} \in (0,1)$. This algorithm runs in time $T = \tilde{O}((N/n_{\mathsf{sat}})^2)$.

Set $c_{\mathsf{str}} := (c_u + 2)/\varepsilon_{\mathsf{sat}} + 3$. Let $\varepsilon' := \varepsilon'(n) \geq n^{-c_u}$ be the error parameter and $\ell' := \ell'(n) = N^{c_u \log(1/\varepsilon')}$ be the stretch, then we can check these parameters satisfy the requirements of Theorem 3.5.1 as follows.

$$\begin{aligned}
&2^{\log^{c_u} n} \leq N \leq 2^{n^{0.99}} \\
&\ell'(n) = N^{c_u \log(1/\varepsilon')} \geq c_u s \\
&T \leq N^2/2^{\log^{c_u+2} n} \leq N^2/n^{c_u^2 \log^{c_u} n} \leq N^2/(\log N)^{c_u^2 \log^{c_u} n} \leq N^2/P^{c_u}
\end{aligned}$$

(That is, we use the aforementioned algorithm to solve Satisfying-Pairs with N circuits of size s and N inputs of length n by padding $n_{\mathsf{sat}} - n$ dummy bits to each input, and then apply Theorem 3.5.1). By Theorem 3.5.1, we get an $\mathsf{FP}^{\mathsf{NP}}$ algorithm for ACC^0-Remote-Point with error $\varepsilon'(n) > n^{-c_u}$, and stretch $\ell'(n) = N^{c_u \log(1/\varepsilon)}$. We can check that both $\ell(n) > \ell'(n+1)$ and $\varepsilon(n) > 2\varepsilon'(n+1)$ hold, so we can invoke Lemma 3.3.7 and get a desired $\mathsf{FP}^{\mathsf{NP}}$ algorithm for remote point with the original parameters. □

We can easily recover the state-of-the-art almost-everywhere average-case lower bounds against ACC^0 [CLW20] by giving the truth table generator as the input.

Corollary 3.1.18. *For every constants $d, m \geq 1$, there is an $\varepsilon > 0$ and a language $L \in \mathsf{E}^{\mathsf{NP}}$ such that L_n cannot be $(1/2 + 2^{-n^\varepsilon})$-approximated by $\mathsf{AC}^0_d[m]$ circuits of size 2^{n^ε}, for all sufficiently large n.*

Proof Sketch. Let $\mathsf{TT}_s : \{0,1\}^{O(s \log s)} \to \{0,1\}^{2^n}$ be the truth table generator of $\mathsf{AC}^0_d[m]$ circuits, where $s = 2^{n^\varepsilon}$ for some constant ε to be determined later. Each output bit of TT_s is computable

by an $\mathsf{AC}^0_{d'}[m]$ circuit of size $s' = \mathrm{poly}(s)$ for some $d' = O(1)$.

For clarity we define $n_{\mathsf{tt}} = O(s \log s)$ to be the input length of TT_s, $s_{\mathsf{tt}}(n_{\mathsf{tt}}) := s'$ and $d_{\mathsf{tt}} := d'$ to be the size and depth of TT_s, respectively. Let c_u and $c_{\mathsf{str}} := c_{\mathsf{str}}(d_{\mathsf{tt}}, m)$ be the constants in Theorem 3.1.16. Then there is an $\mathsf{FP}^{\mathsf{NP}}$ algorithm A_{hard} that takes as input a circuit $C : \{0,1\}^{n_{\mathsf{tt}}} \to \{0,1\}^{\ell_{\mathsf{tt}}}$ and outputs a string y that is $(1/2 - \varepsilon_{\mathsf{tt}})$-far from $\mathrm{Range}(C)$, where $\ell_{\mathsf{tt}} \geq 2^{\log^{c_{\mathsf{str}}} s_{\mathsf{tt}}}$ and $\varepsilon_{\mathsf{tt}} := 2n_{\mathsf{tt}}^{-c_u}$. By choosing ε to be a sufficiently small constant, we can make

$$2^n \geq 2^{\log^{c_{\mathsf{str}}} s_{\mathsf{tt}}} \qquad \text{and} \qquad 2^{-n^{\varepsilon}} > \varepsilon_{\mathsf{tt}}.$$

We then fix the input of the $\mathsf{FP}^{\mathsf{NP}}$ algorithm A_{hard} above to be TT_s to obtain an $\mathsf{FP}^{\mathsf{NP}}$ algorithm A that takes 1^{2^n} as input and produces a truth table of length 2^n that cannot be $(1/2 + 2^{-n^{\varepsilon}})$-approximated by any size-s circuits. The required hard language is then defined as

$$L := \left\{ x \in \{0,1\}^n : n \in \mathbb{N}, \mathsf{tt} \leftarrow A(1^{2^n}) \in \{0,1\}^{2^n}, \mathsf{tt}_x = 1 \right\}. \qquad \square$$

3.8.3 Hard Partial Truth Tables for ACC^0

Theorem 3.1.17 (ACC^0-Partial-AvgHard $\in \mathsf{FP}^{\mathsf{NP}}$)**.** *There is a constant $c_u \geq 1$ such that for every constants $d, m \geq 1$, there is a constant $c_{\mathsf{str}} := c_{\mathsf{str}}(d, m) \geq 1$, such that the following holds.*

Let $n < s(n) \leq 2^{n^{o(1)}}$ be a size parameter, $\varepsilon := \varepsilon(n) \geq 2n^{-c_u}$ be an error parameter and $\ell := \ell(n) \geq 2^{\log^{c_{\mathsf{str}}} s}$ be a stretch function, then there is an $\mathsf{FP}^{\mathsf{NP}}$ algorithm that given inputs $x_1, \ldots, x_\ell \in \{0,1\}^n$, it outputs a string $y \in \{0,1\}^\ell$ such that for any $s(n)$-size $\mathsf{AC}^0_d[m]$ circuit C, y is $(1/2 - \varepsilon)$-far from $C(x_1) \circ \cdots \circ C(x_\ell)$.

Proof Sketch. The proof is similar to Theorem 3.1.16, so we only sketch the proof.

Let c_u be the constant from Theorem 3.7.1, and then we set the values[16] of c_{str}, n_{sat}, N, $\varepsilon_{\mathsf{sat}}$, T, $\varepsilon'(n)$ and $\ell'(n)$ in the same way as proof Theorem 3.1.16.

Since parameter constraints of Theorem 3.7.1 are similar to those of Theorem 3.5.1, These parameter settings can be used to invoke Theorem 3.7.1 and get an $\mathsf{FP}^{\mathsf{NP}}$ algorithm for $O(s(n))$-size $\mathsf{AC}_{d+O(1)}[m]$-Partial-AvgHard with stretch $\ell'(n)$ and error $\varepsilon'(n)$. We can check that both $\ell(n) > \ell'(n+1)/2$ and $\varepsilon(n) > 2\varepsilon'(n+1)$ hold, so it is valid to invoke Lemma 3.3.8 and get a desired $\mathsf{FP}^{\mathsf{NP}}$ algorithm for average-case hard partial truth tables with the original parameters.

Alternatively, we can reduce ACC^0-Partial-AvgHard to ACC^0-Remote-Point (see Section 3.2) and simply apply Theorem 3.1.16, since the evaluation of ACC^0 circuits can be implemented in ACC^0. $\square$

As a consequence, we show (following the observation in [AS10]) that there is no efficient mapping reduction from E^{NP} to any language decidable by small-size non-uniform ACC^0 circuits.

Corollary 3.1.19. *Let $d, m \in \mathbb{N}$ be constants, $\mathsf{AC}^0_d[m]$ denote the class of languages computable by a* non-uniform *family of polynomial-size $\mathsf{AC}^0_d[m]$ circuits. Then, there is a language $L^{\mathsf{hard}} \in \mathsf{E}^{\mathsf{NP}}$ that does not have polynomial-time mapping reductions to any language in $\mathsf{AC}^0_d[m]$.*

[16] In order to invoke Lemma 3.3.8, we actually use $s'(n) = O(s(n))$ as size function and $d' := d + O(1)$ as depth. These are rather minor changes, so we can still use the same parameter settings strategy.

Proof. Our E^{NP} language L^{hard} receives two inputs: a Turing machine R and a string y. Here, the lengths of $\langle R\rangle$ (the encoding of R) and y are $\lceil n/2\rceil$ and $n' := \lfloor n/2\rfloor$ respectively, thus L^{hard} receives n-bit strings as inputs. The machine R is interpreted as a reduction that runs in $T(n) := n^{\log n}$ time (which we diagonalise against).

We run R on all inputs of the form $(\langle R\rangle, x')$, where $|x'| = n'$. Let $x_1, x_2, \ldots, x_{2^{n'}}$ be an enumeration of length-n' strings, and $z_i := R(\langle R\rangle, x_i)$ be a string of length at most $T(n)$. Note that the strings z_i may not be of the same length, but the length of each z_i is at most $T(n)$. By an averaging argument, there is an $\ell \leq T(n)$ such that there are at least $2^{n'}/T(n) \geq 2^{n^{0.99}}$ strings z_i with length exactly ℓ. Let N be the number of strings z_i with length exactly ℓ and denote these strings to be $z_{i_1}, z_{i_2}, \ldots, z_{i_N}$. We can check the technical constraints and invoke Theorem 3.1.17 to get an $\mathsf{FP}^{\mathsf{NP}}$ algorithm for solving the $\mathsf{AC}^0_d[m]$-Partial-Hard problem on inputs $z_{i_1}, z_{i_2}, \ldots, z_{i_N}$. We obtain a sequence of bits $y_{i_1}, y_{i_2}, \ldots, y_{i_N} \in \{0,1\}$ such that for every size-$\ell^{\log \ell}$ $\mathsf{AC}^0_d[m]$ circuit C, there is some $j \in [N]$ such that $C(z_{i_j}) \neq y_{i_j}$. This can be done in deterministic $2^{O(n)}$ time with an NP oracle. Finally, we define L^{hard} as follows: suppose x is the i-th string of length n' (i.e., $x = x_i$), then $x \in L^{\mathsf{hard}}$ if and only if $|z_i| = \ell$ and $y_i = 1$.

Clearly, L^{hard} runs in deterministic $2^{O(n)}$ time with an NP oracle. We still need to show that for every language $L \in \mathsf{AC}^0_d[m]$, there is no polynomial time reduction from L^{hard} to L. Suppose, for the sake of contradiction, that there is a polynomial-time reduction R from L^{hard} to L. Let n be a sufficiently large number such that $n/2 > \langle R\rangle$ and $T(n) = n^{\log n}$ is larger than the running time of R. Consider running R on inputs of the form $(\langle R\rangle, x)$ where $|x| = \lfloor n/2\rfloor$. Let x_i, y_i, z_i, ℓ, and N be defined as above, and C be an $\mathsf{AC}^0_d[m]$ circuit that decides L on input length ℓ. Since the size of C is at most $\mathrm{poly}(\ell) \leq \ell^{\log \ell}$, there is some $j \leq N$ such that $C(z_{i_j}) \neq y_{i_j}$. In other words,

$$\exists i \in N, C(R(\langle R\rangle, x_i)) \neq L^{\mathsf{hard}}(\langle R\rangle, x_i).$$

It follows that R is not a correct reduction from L^{hard} to L. □

Chapter 4

The "Complete" Algorithmic Method

4.1 Overview

In this chapter, we show that circuit lower bounds for E^{NP} and CAPP algorithms with E^{NP} preprocessing are equivalent. By slightly modifying the proof of Theorem 3.4.2 we can see that a non-trivial $\mathsf{GapUNSAT}$ algorithm for $\mathscr{C}$, even *with* E^{NP} *preprocessing*, would imply $\mathsf{E}^{\mathsf{NP}} \not\subseteq \mathscr{C}$. (A precise definition can be seen at Definition 4.3.1.) We show that for powerful enough circuit classes $\mathscr{C}$ (e.g., TC^0, NC^1, or $\mathsf{P}/_{\mathrm{poly}}$), the converse is also true:

Theorem 4.1.1 (Informal)**.** *Let $\mathscr{C} \in \{\mathsf{TC}^0, \mathsf{NC}^1, \mathsf{P}/_{\mathrm{poly}}\}$. The following are equivalent:*

- E^{NP} *cannot be computed by polynomial-size $\mathscr{C}$ circuits on almost every input length.*
- *There is a non-trivial* $\mathsf{GapUNSAT}$ *algorithm for $\mathscr{C}$ circuits with* E^{NP} *preprocessing.*

For circuit classes $\mathscr{C}$ that are less powerful (i.e., that might not be able to efficiently compute $\mathsf{MAJORITY}$), we show that *strong average-case* circuit lower bounds against $\mathscr{C}$ and CAPP algorithms for $\mathscr{C}$ with inverse-circuit-size error are equivalent:

Theorem 4.1.2 (Main Results 2.2, Informal)**.** *Let $\mathscr{C}$ be a "weak" circuit class under some mild closure properties. The following are equivalent:*

- E^{NP} *cannot be $(1/2 + 1/\mathrm{poly}(n))$-approximated by $\mathscr{C}$ circuits on almost every input length.*
- *There is a non-trivial* CAPP *algorithm for $\mathscr{C}$ circuits with* E^{NP} *preprocessing and inverse-circuit-size error.*

Actually, we can show equivalences among a lot of notions, including strong average-case lower bounds for E^{NP}, non-trivial CAPP algorithms with E^{NP} preprocessing, subexponential-time CAPP algorithms with E^{NP} preprocessing, and E^{NP}-computable PRGs. See Theorem 4.6.1 and Theorem 4.6.3 for details.

One advantage of our equivalence is that it also holds for larger size bounds and the case of infinitely-often lower bounds:

Theorem 4.1.3 (Informal)**.** *Let $\mathscr{C}$ be a "weak" circuit class under some mild closure properties. The following are equivalent:*

- E^{NP} *cannot be* $(1/2 + 1/2^{n^{o(1)}})$*-approximated by* $\mathscr{C}$ *circuits of size* $2^{n^{o(1)}}$.
- *There is a* CAPP *algorithm for* $\mathscr{C}$ *circuits of size* $2^{n^{o(1)}}$ *with* $2^{n-n^{\Omega(1)}}$ *query time,* E^{NP} *preprocessing, and inverse-circuit-size error, that works for infinitely many* n.

Remark 4.1.4 (Equivalences between Derandomisation and Lower Bounds).
Equivalences between derandomisation and lower bounds are known in many settings.

- Impagliazzo, Kabanets, and Wigderson [IKW02] showed that $\mathsf{NEXP} \not\subseteq \mathsf{P}_{/\mathrm{poly}}$ if and only if there is a non-deterministic subexponential-time algorithm for CAPP with $n^{o(1)}$ bits of advice and error $1/6$ that works infinitely often.
- Korten's result [Kor21] can also be interpreted as an equivalence between derandomisation and lower bounds: A full derandomisation of the trivial $\mathsf{FZPP}^{\mathsf{NP}}$ algorithm for Avoid is equivalent to both $\mathsf{E}^{\mathsf{NP}} \not\subseteq \mathsf{SIZE}[2^{0.1n}]$ and $\mathsf{E}^{\mathsf{NP}} \not\subseteq \mathsf{SIZE}[2^n/3n]$.
- Equivalences between derandomisation and *uniform* lower bounds are also known. Impagliazzo and Wigderson [IW01] showed that $\mathsf{EXP} \neq \mathsf{BPP}$ is equivalent to an infinitely-often, subexponential time derandomisation of BPP on average ($\mathsf{BPP} \subseteq$ i.o.-$\mathsf{heurDTIME}[2^{n^{o(1)}}]$). Williams [Wil16] showed that $\mathsf{NEXP} \neq \mathsf{BPP}$ is equivalent to an infinitely-often, subexponential time nondeterministic derandomisation of BPP on average, with $n^{o(1)}$ bits of advice ($\mathsf{BPP} \subseteq$ i.o.-$\mathsf{heurNTIME}[2^{n^{o(1)}}]/_{n^{o(1)}}$).

In our opinion, compared to the above equivalences, our results have the following features that make them particularly attractive:

- First, they work in both infinitely-often and almost-everywhere settings; in contrast, [IW01] and [Wil16] only hold for infinitely-often lower bounds.
- Second, they scale better with large circuit size bounds (such as $2^{n^{o(1)}}$); no similar equivalences to [IKW02] for $\mathsf{NEXP} \not\subseteq \mathsf{SIZE}[2^{n^{o(1)}}]$ or to [IW01] for $\mathsf{EXP} \not\subseteq \mathsf{BPTIME}[2^{n^{o(1)}}]$ are known.
- Third, they are also true for weaker circuit classes such as formulas or ACC^0 circuits; in contrast, the arguments in [Kor21] do not seem to yield any characterisation of, e.g., the lower bound $\mathsf{E}^{\mathsf{NP}} \not\subseteq \mathsf{Formula}[2^{0.1n}]$.
- Finally, our equivalences include both subexponential-time derandomisation and non-trivial derandomisation; none of the equivalences above are known to include non-trivial derandomisation.

An interesting corollary of Theorem 4.1.1 and Theorem 4.1.2 is the following "speed-up" result for derandomisation with E^{NP} preprocessing:

Corollary 4.1.5 (Informal)**.** *The following are true:*

- *If there is a non-trivial* $\mathsf{GapUNSAT}$ *algorithm for* TC^0 *circuits with* E^{NP} *preprocessing, then there is a subexponential-time* CAPP *algorithm for* TC^0 *circuits with* E^{NP} *preprocessing.*
- *Let* $\mathscr{C}$ *be a "weak" circuit class under some mild closure properties. If there is a non-trivial* CAPP *algorithm for* $\mathscr{C}$ *circuits with* E^{NP} *preprocessing and inverse-circuit-size error, then there is a subexponential-time* CAPP *algorithm for* $\mathscr{C}$ *circuits with* E^{NP} *preprocessing and inverse-circuit-size error.*

Remark 4.1.6 (Comparison with Other Speed-Ups in Complexity Theory).

Williams [Wil13a] showed that if CAPP has a nondeterministic algorithm with non-trivial running time, then CAPP also has a nondeterministic subexponential time algorithm. One caveat of this result is that the speed-up algorithm is only infinitely-often correct and requires n^ε bits of advice. Therefore, the speed-up algorithm does not imply the non-trivial algorithm. In contrast, in Corollary 4.1.5, the speed-up algorithms always imply the non-trivial algorithms.

Oliveira and Santhanam [OS17a] showed a similar speed-up result in learning theory: a typical circuit class is "non-trivially learnable" if and only if it is learnable in subexponential time. Their result is proved using the connection between natural proofs and learning [RR97, CIKK16], while our result is a strengthening of the Algorithmic Method.

4.2 Preliminaries

4.2.1 Pseudorandom Generators

Let $\mathscr{C}$ be a circuit class, $\varepsilon > 0$, and $r(n) < n$ be a good function. A *pseudorandom generator* (PRG) with seed length $r(n)$ that ε-fools $\mathscr{C}$ is a function $G : \{0,1\}^r \to \{0,1\}^n$ such that for every circuit $C \in \mathscr{C}$,

$$\left| \Pr_{x \leftarrow \{0,1\}^n}[C(x) = 1] - \Pr_{\mathsf{seed} \leftarrow \{0,1\}^r}[C(G(\mathsf{seed})) = 1] \right| \leq \varepsilon.$$

We also say G is an *i.o. PRG* if the above condition holds for infinitely many lengths n.

In this chapter, we will mostly consider E^{NP}-computable PRGs, where G is computable in $2^{O(r)}$ time with access to an NP oracle. (It is without loss of generality to assume that $r \geq \Omega(\log n)$.)

We need the classical construction of PRGs from average-case lower bounds [NW94]. Let Junta_k be the class of k-juntas, i.e., functions that only depend on k input bits. We have:

Theorem 4.2.1 ([NW94], see also [CR22, Theorem 6.4]). *Let m, ℓ, a be integers such that $a \leq \ell$, and let $t := O(\ell^2 \cdot m^{1/a}/a)$. Let $\mathscr{C}$ be a circuit class closed under negation. There is a function $G : \{0,1\}^{2^\ell} \times \{0,1\}^t \to \{0,1\}^m$ computable in deterministic $\mathrm{poly}(m, 2^t)$ time such that the following holds.*

For any function $Y : \{0,1\}^\ell \to \{0,1\}$ represented as a length-2^ℓ truth table, if Y cannot be $(1/2 + \varepsilon/m)$-approximated by $\mathscr{C}[S] \circ \mathsf{Junta}_a$ circuits (i.e., the top $\mathscr{C}$ circuit has size S), then $G(Y, -)$ is a PRG that ε-fools every $\mathscr{C}[S]$ circuit. That is, for any circuit $C \in \mathscr{C}[S]$,

$$\left| \Pr_{s \leftarrow \{0,1\}^t}[C(G(Y, s)) = 1] - \Pr_{x \leftarrow \{0,1\}^m}[C(x) = 1] \right| \leq \varepsilon.$$

4.2.2 Elementary Properties of Norm and Inner Product

We discuss some properties of norms and the inner product of functions on Boolean cubes, which will be useful for us. For a function $f : \{0,1\}^n \to \mathbb{R}$, we define its ℓ_p-norm as

$$\|f\|_p := \left(\mathop{\mathbb{E}}_{x \leftarrow \{0,1\}^n}[|f(x)|^p] \right)^{1/p}$$

In particular, the ℓ_∞-norm is defined as the maximum absolute value of f.

$$\|f\|_\infty := \max_{x\in\{0,1\}^n} \{|f(x)|\}.$$

For $m \geq 2$ and functions $f_1, f_2, \ldots, f_d : \{0,1\}^n \to \mathbb{R}$, define their *inner product* as:

$$\langle f_1, f_2, \ldots, f_d \rangle := \mathop{\mathbb{E}}_{x\leftarrow\{0,1\}^n} \left[\prod_{i=1}^{d} f_i(x) \right].$$

We need the following generalisation of Hölder's inequality:

Fact 4.2.2. *Let d be an integer. For functions $f_1, f_2, \ldots, f_d : \{0,1\}^n \to \mathbb{R}$ we have that*

$$\left\| \prod_{i=1}^{d} f_i \right\|_1 \leq \prod_{i=1}^{d} \|f_i\|_d$$

We need the following simple lemma.

Lemma 4.2.3 (a generalisation of [CW19b, Lemma 28])**.** *For any integer $d \geq 2$ and functions $f_1, f_2, \ldots, f_d$ and $g_1, g_2, \ldots, g_d$ from $\{0,1\}^n \to \mathbb{R}$ and $\varepsilon, \alpha > 0$, suppose for all $i \in [d]$ we have:*

- $\|f_i\|_d \leq \alpha$ *and* $\|g_i\|_d \leq \alpha$,
- $\|f_i - g_i\|_d \leq \varepsilon$.

Then $|\langle f_1, f_2, \ldots, f_d \rangle - \langle g_1, g_2, \ldots, g_d \rangle| \leq d \cdot \alpha^{d-1} \cdot \varepsilon$.

Proof. We have

$$\begin{aligned}
|\langle f_1, \ldots, f_d \rangle - \langle g_1, \ldots, g_d \rangle| &\leq \sum_{i=1}^{d} |\langle f_1, \ldots, f_i, g_{i+1}, \ldots, g_d \rangle - \langle f_1, \ldots, f_{i-1}, g_i, \ldots, g_d \rangle| \\
&\leq \sum_{i=1}^{d} |\langle f_1, \ldots, f_i - g_i, g_{i+1}, \ldots, g_d \rangle| \\
&\leq d \cdot \alpha^{d-1} \cdot \varepsilon \text{ (by Fact 4.2.2).}
\end{aligned}$$

□

4.2.3 Linear Sum of Circuits

$\mathsf{Sum} \circ \mathscr{C}$ circuits. Let $\mathscr{C}$ be a circuit class. A $\mathsf{Sum} \circ \mathscr{C}$ circuit $C : \{0,1\}^n \to \mathbb{R}$ ([Wil18b]) is a circuit of the following form:

$$C(x) = \sum_{i=1}^{\ell} \alpha_i C_i(x),$$

where each $\alpha_i \in \mathbb{R}$ and each C_i is a $\mathscr{C}$ circuit. We say that C has *complexity* at most s, denoted as $\mathsf{complexity}(C) \leq s$, if all of the following holds:

- the total size of all bottom $\mathscr{C}$ circuits C_i is at most s;
- $\sum_{i=1}^{\ell} |\alpha_i| \leq s$;

- the bit-complexity of each α_i is at most s, i.e., one can write the rational number α_i as a fraction u_i/v_i where u_i, v_i are integers and $\log(|u_i|) + \log(|v_i|) \leq s$.

The definition of "complexity" in [CR22, CLW20] only required the first two bullets above (i.e., $\mathsf{complexity}(C) = \max\{\sum_{i=1}^{\ell} |\alpha_i|, \sum_{i=1}^{\ell} |C_i|\}$). However, it is easy to see that the linear sum circuits produced by the decoders—[CR22, Lemma 3.1] and [CLW20, Lemma 3.8] (which is Theorem 4.2.9 in this thesis)—also satisfy the third bullet.

If $C(x) \in [0,1]$ for every input $x \in \{0,1\}^n$, then we say C is a $[0,1]$-$\mathsf{Sum} \circ \mathscr{C}$ circuit ([CLW20]). Recall that the ℓ_1-distance between a $\mathsf{Sum} \circ \mathscr{C}$ circuit C and a function f is

$$\|C - f\|_1 = \mathop{\mathbb{E}}_{x \leftarrow \{0,1\}^n}[|C(x) - f(x)|].$$

We define bin_C as the Boolean function that is closest to C. That is, for every $x \in \{0,1\}^n$, if $C(x) \leq 0.5$ then $\mathsf{bin}_C(x) = 0$, otherwise $\mathsf{bin}_C(x) = 1$.

$\widetilde{\mathsf{Sum}} \circ \mathscr{C}$ circuits. Let $\delta \in [0, 0.5)$. A $\mathsf{Sum} \circ \mathscr{C}$ circuit C is said to be a $\widetilde{\mathsf{Sum}}_\delta \circ \mathscr{C}$ circuit ([CW19b, CR22]), if for every input $x \in \{0,1\}^n$, either $|L(x) - 1| \leq \delta$ or $|L(x)| < \delta$. We say $C(x) = 1$ if $|L(x) - 1| \leq \delta$, and $C(x) = 0$ otherwise.

4.2.4 Algorithms for Linear Sum of Circuits

In the context of the Algorithmic Method, one advantage of $\mathsf{Sum} \circ \mathscr{C}$ circuits is that they preserve *algorithms*: circuit analysis algorithms for $\mathscr{C}$ often imply circuit analysis algorithms for $\mathsf{Sum} \circ \mathscr{C}$. Below are a few examples that will be useful for us.

PRGs fooling $\mathscr{C}$ also fools $\widetilde{\mathsf{Sum}} \circ \mathscr{C}$. Suppose that G is a PRG that fools $\mathscr{C}$ circuits, we show that it also fools $\widetilde{\mathsf{Sum}} \circ \mathscr{C}$ circuits.

Lemma 4.2.4. *Let $\varepsilon, \delta > 0$, $\mathscr{C}$ be a circuit class, $s(n)$ be a good function, and $G : \{0,1\}^r \to \{0,1\}^n$ be a PRG that ε-fools $\mathscr{C}$ circuits of size $s(n)$. For $\varepsilon' := 2\delta + \varepsilon \cdot s(n)$, G also ε'-fools $\widetilde{\mathsf{Sum}}_\delta \circ \mathscr{C}$ circuits of complexity $s(n)$.*

Proof. Let C be a $\widetilde{\mathsf{Sum}}_\delta \circ \mathscr{C}$ circuit where the underlying (real-valued) $\mathsf{Sum} \circ \mathscr{C}$ circuit is

$$\tilde{C} := \sum_{i=1}^{\ell} \alpha_i \cdot C_i.$$

Let $\mathcal{U}_n$ denote the uniform distribution over $\{0,1\}^n$; we abuse notation and let G_n denote the distribution of $G_n(y)$ for a uniformly random $y \in \{0,1\}^r$. From the definition of $\widetilde{\mathsf{Sum}}_\delta$ gates, we have

$$\left|\mathbb{E}[C(\mathcal{U}_n)] - \mathbb{E}[\tilde{C}(\mathcal{U}_n)]\right| \leq \delta \quad \text{and} \quad \left|\mathbb{E}[C(G_n)] - \mathbb{E}[\tilde{C}(G_n)]\right| \leq \delta.$$

Using the property of G_n, we have

$$\begin{aligned}\left|\mathbb{E}[\tilde{C}(\mathcal{U}_n)] - \mathbb{E}[\tilde{C}(G_n)]\right| &= \left|\mathbb{E}\left[\sum_{i=1}^{\ell} \alpha_i \cdot C_i(\mathcal{U}_n)\right] - \mathbb{E}\left[\sum_{i=1}^{\ell} \alpha_i \cdot C_i(G_n)\right]\right| \\ &= \left|\sum_{i=1}^{\ell} \alpha_i \cdot (\mathbb{E}[C_i(\mathcal{U}_n)] - \mathbb{E}[C_i(G_n)])\right| \\ &\leq \left(\max_{i=1}^{\ell} |\mathbb{E}[C_i(\mathcal{U}_n)] - \mathbb{E}[C_i(G_n)]|\right) \cdot \left(\sum_{i=1}^{\ell} |\alpha_i|\right) \\ &\leq \varepsilon \cdot s(n).\end{aligned}$$

Therefore,

$$\begin{aligned}|\mathbb{E}[C(\mathcal{U}_n)] - \mathbb{E}[C(G_n)]| &\leq \left|\mathbb{E}[C(\mathcal{U}_n)] - \mathbb{E}[\tilde{C}(\mathcal{U}_n)]\right| + \left|\mathbb{E}[\tilde{C}(\mathcal{U}_n)] - \mathbb{E}[\tilde{C}(G_n)]\right| \\ &\quad + \left|\mathbb{E}[\tilde{C}(G_n)] - \mathbb{E}[C(G_n)]\right| \\ &\leq 2\delta + \varepsilon \cdot s(n) = \varepsilon'.\end{aligned}$$

□

Average-of-Product of $\mathsf{Sum} \circ \mathscr{C}$ circuits. Let $d \geq 2$ be a constant. The Average-of-Product problem for d $\mathsf{Sum} \circ \mathscr{C}$ circuits [Wil18b, CW19b, CR22, CLW20] is the following problem: Given d $\mathsf{Sum} \circ \mathscr{C}$ circuits $C_1, C_2, \ldots, C_d : \{0,1\}^n \to \mathbb{R}$, the task is to estimate $\mathbb{E}_{x \leftarrow \{0,1\}^n}[\prod_{i=1}^{d} C_i(x)]$. It is not hard to reduce this problem to CAPP for $\mathsf{AND}_d \circ \mathscr{C}$ circuits:

Theorem 4.2.5. *Let $\mathscr{C}$ be a typical circuit class. Suppose there is an oracle solving CAPP on $\mathsf{AND}_d \circ \mathscr{C}$ circuits of size $S(n)$ and n inputs within an additive error of $\varepsilon(n)$. Then there is an algorithm that given d $\mathsf{Sum} \circ \mathscr{C}$ circuits $C_1, C_2, \ldots, C_d$ of complexity at most $S(n)$, outputs an estimation of $\mathbb{E}_{x \leftarrow \{0,1\}^n}[\prod_{i=1}^{d} C_i(x)]$ within additive error $S(n)^d \cdot \varepsilon(n)$ in deterministic $S(n)^{d+O(1)}$ time with $O(S(n)^d)$ oracle calls to the CAPP oracle.*

Proof. Lemma 6.2 of [CLW20] proved this theorem for $d = 4$, but it is easy to see that the proof generalises to arbitrary d. We provide a full proof here for completeness.

It is without loss of generality to assume that each C_i contains exactly $S(n)$ $\mathscr{C}$ sub-circuits. For each $i \in [d]$, write $C_i = \sum_{j=1}^{S(n)} \alpha_{i,j} C_{i,j}$, where $\alpha_{i,j} \in \mathbb{R}$ and $C_{i,j}$ is a $\mathscr{C}$ circuit of size at most $S(n)$. Moreover, for each $i \in [d]$, we have that $\sum_{j=1}^{S(n)} |\alpha_{i,j}| \leq S(n)$. Then we have

$$\begin{aligned}\mathop{\mathbb{E}}_{x \leftarrow \{0,1\}^n}\left[\prod_{i=1}^{d} C_i(x)\right] &= \mathop{\mathbb{E}}_{x \leftarrow \{0,1\}^n}\left[\prod_{i=1}^{d} \sum_{j=1}^{S(n)} \alpha_{i,j} C_{i,j}(x)\right] \\ &= \mathop{\mathbb{E}}_{x \leftarrow \{0,1\}^n}\left[\sum_{j_1,\ldots,j_d \in [S(n)]} \prod_{i=1}^{d} \alpha_{i,j_i} C_{i,j_i}(x)\right] \\ &= \sum_{j_1,\ldots,j_d \in [S(n)]} \left(\prod_{i=1}^{d} \alpha_{i,j_i}\right) \cdot \mathop{\mathbb{E}}_{x \leftarrow \{0,1\}^n}\left[\bigwedge_{i=1}^{d} C_{i,j_i}(x)\right]. \qquad (4.1)\end{aligned}$$

We enumerate $j_1, \ldots, j_d \in [S(n)]$ and use the CAPP oracle for $\mathsf{AND}_d \circ \mathscr{C}$ circuits to estimate $\mathbb{E}_{x \leftarrow \{0,1\}^n}[\bigwedge_{i=1}^{d} C_{i,j_i}(x)]$ within additive error $\varepsilon(n)$. This gives us an estimation of (4.1) within

an additive error of at most

$$\sum_{j_1,\ldots,j_d\in[S(n)]}\left|\prod_{i=1}^{d}\alpha_{i,j_i}\right|\cdot\varepsilon(n)\le S(n)^d\cdot\varepsilon(n).$$

Clearly, our algorithm runs in deterministic $S(n)^{d+O(1)}$ time and makes $O(S(n)^d)$ calls to the oracle. □

Verification of $\mathsf{Sum}\circ\mathscr{C}$ circuits. We need the following lemma for testing whether a $\mathsf{Sum}\circ\mathscr{C}$ circuit has low ℓ_d-distance to its closest Boolean function. The lemma has a similar proof to [CR22, Lemma 5.3].

Lemma 4.2.6. *Let $d\ge 2$ be an even number. Suppose we are given a $\mathsf{Sum}\circ\mathscr{C}$ circuit $C:\{0,1\}^n\to\mathbb{R}$ of complexity $S(n)$ and a parameter $\delta<0.01/d$. Let $\varepsilon:=\frac{\delta^d}{2\cdot 3^d\cdot(S+1)^{2d}}$. Suppose there is an oracle that solves the CAPP problem for $\mathsf{AND}_{2d}\circ\mathscr{C}$ with error ε. Then there is an algorithm A running in deterministic $S^{2d+O(1)}$ time and making $O(S^{2d})$ queries to the oracle such that:*

- *If one of the following conditions holds, then A always accepts;*
  - $\|C-\mathsf{bin}_C\|_\infty\le\delta/3$
  - $\|C-\mathsf{bin}_C\|_1\le(\delta/3)^d$ *and* $C(x)\in[0,1]$ *for any* $x\in\{0,1\}^n$
- *if $\|C-\mathsf{bin}_C\|_d\ge\delta$, then A always rejects;*
- *otherwise, A can output anything.*

Proof. We define a polynomial $P(z)=z^d(1-z)^d$. We also define $d_{\mathsf{bin}}(z):=\min\{|z|,|z-1|\}$. Recall from Fact 3.5.5 that $d_{\mathsf{bin}}(z)^d/2^d\le P(z)\le d_{\mathsf{bin}}(z)^d\cdot(1+d_{\mathsf{bin}}(z))^d$.

We need the following properties of $P(C(x))$:

1. When $d_{\mathsf{bin}}(z)\le\delta/3$, we have $P(z)\le(\delta/3)^d\cdot(1+\delta/3)^d$. Hence if $\|C-\mathsf{bin}_C\|_\infty\le\delta/3$, then

$$\mathop{\mathbb{E}}_{x\leftarrow\{0,1\}^n}[P(C(x))]\le(\delta/3)^d\cdot(1+\delta/3)^d\le(\delta/3)^d\cdot(1+0.01/d)^d\le(\delta/3)^d\cdot e^{0.01}.$$

2. If $z\in[0,1]$, then $P(z)\le d_{\mathsf{bin}}(z)$. Hence, if $C(x)\in[0,1]$ holds for every $x\in\{0,1\}^n$ and if $\|C-\mathsf{bin}_C\|_1\le(\delta/3)^d$, then we have

$$\mathop{\mathbb{E}}_{x\leftarrow\{0,1\}^n}[P(C(x))]\le\|C-\mathsf{bin}_C\|_1\le(\delta/3)^d.$$

3. If $\|C-\mathsf{bin}_C\|_d\ge\delta$, then by definition we have

$$\mathop{\mathbb{E}}_{x\leftarrow\{0,1\}^n}\left[d_{\mathsf{bin}}(C(x))^d\right]\ge\delta^d.$$

   Since $d\ge 2$, we have

$$\mathop{\mathbb{E}}_{x\leftarrow\{0,1\}^n}[P(C(x))]\ge(\delta/2)^d\ge(9/4)\cdot(\delta/3)^d.$$

Given the above properties, we can see that it suffices to compute

$$\mathop{\mathbb{E}}_{x \leftarrow \{0,1\}^n}[P(C(x))] \tag{4.2}$$

within error $(\delta/3)^d/2$ to distinguish between these two cases in the lemma. Clearly, this reduces to the Average-of-Product of $2d$ $\mathsf{Sum} \circ \mathscr{C}$ circuits of complexity at most $S+1$. Hence, by Theorem 4.2.5, (4.2) can be approximated within additive error $(S+1)^{2d} \cdot \varepsilon$ in deterministic $S^{2d+O(1)}$ time with $O(S^{2d})$ oracle calls to the CAPP oracle. Since $(S+1)^{2d} \cdot \varepsilon \leq (\delta/3)^d/2$, this finishes the proof. □

4.2.5 Worst-Case Hardness from PRGs

The following simple fact states that PRGs imply worst-case hardness.

Fact 4.2.7 ([CLLO21, Proposition 9]). *Let $\mathscr{C}$ be a circuit class and $r = r(n)$ be a good function. Suppose there is an E^{NP}-computable PRG (i.o. PRG respectively) $G : \{0,1\}^r \to \{0,1\}^n$ that ε-fools $\mathscr{C}$, where $\varepsilon < 1 - 2^{r-n}$. Then there is a language $L \in \mathsf{E}^{\mathsf{NP}}$ that cannot be computed by $\mathscr{C}$ circuits on almost every input length (infinitely many input lengths respectively).*

4.2.6 Hardness Amplification

Hardness amplification with a TC^0 decoder. We need the following result.

Theorem 4.2.8 ([GR08]). *Let $\varepsilon > 2^{-c\sqrt{n}}$ for some absolute constant c. There are two algorithms Amp and Dec such that:*

- *For some constant $d > 1$, Amp takes as input the truth table of a function $f : \{0,1\}^n \to \{0,1\}$ and outputs the truth table of a function $\mathsf{Amp}(f) : \{0,1\}^{dn} \to \{0,1\}$.*
- *$\mathsf{Dec}^{(-)}$ receives an oracle h, an input $x \in \{0,1\}^n$, an advice string $\alpha \in \{0,1\}^{O(\log \varepsilon^{-1})}$, as well as two random strings r_1, r_2, and outputs a bit b.*
- *For every function $h : \{0,1\}^{dn} \to \{0,1\}$ that $(1/2+\varepsilon)$-approximates $\mathsf{Amp}(f)$,*

$$\Pr_{r_1}\left[\exists \alpha \in \{0,1\}^{O(\log \varepsilon^{-1})} \text{ s.t. } \forall x \in \{0,1\}^n, \Pr_{r_2}[\mathsf{Dec}^h(\alpha, x, r_1, r_2) = f(x)] > 9/10\right] > 99/100.$$

- *Amp runs in deterministic $2^{O(n)}$ time and Dec is a TC^0 oracle circuit of size $\mathrm{poly}(n, \varepsilon^{-1})$.*

A non-standard XOR lemma. For a function $f : \{0,1\}^n \to \{0,1\}$ and an integer k, we define the function $f^{\oplus k}$ to take k inputs $x_1, x_2, \ldots, x_k \in \{0,1\}^n$ and compute

$$f^{\oplus k}(x_1, x_2, \ldots, x_k) = f(x_1) \oplus f(x_2) \oplus \cdots \oplus f(x_k).$$

We need the following XOR lemma with a linear sum corrector.

Theorem 4.2.9 ([Lev87], [CLW20, Lemma 3.8]). *Let $\mathscr{C}$ be a circuit class that is closed under negation and projection. Let $\delta < 1/2$, $k \in \mathbb{N}$ be a parameter, and*

$$\varepsilon_k := (1-\delta)^{k-1}(1/2 - \delta).$$

For any function $f : \{0,1\}^n \to \{0,1\}$, if f cannot be $(1-\delta)$-approximated in ℓ_1-distance by $[0,1]$-$\mathsf{Sum} \circ \mathscr{C}$ circuits of complexity $O\left(\frac{ns}{(\delta \cdot \varepsilon_k)^2}\right)$, then $f^{\oplus k}$ cannot be $(1/2+\varepsilon_k)$-approximated by $\mathscr{C}$ circuits of size s.

(Theorem 4.2.9 is just a more familiar version of Theorem 3.3.5 stated in the language of hardness amplification instead of local list decoding.)

4.3 Derandomisation with Preprocessing Implies Circuit Lower Bounds

Let $\mathscr{C}$ be a circuit class, we define the circuit-analysis problems for $\mathscr{C}$ *with* E^{NP} *preprocessing*:

Definition 4.3.1. Let Prep be a preprocessing algorithm, Query be a query algorithm, $\mathscr{C}$ be a circuit class, $s(n)$ be a size parameter, and $\varepsilon > 0$. We say $(\mathsf{Prep}, \mathsf{Query})$ is a CAPP *data structure for* $\mathscr{C}[s(n)]$ *with* E^{NP} *preprocessing* and error ε, if the following are true:

- $\mathsf{Prep}(1^n)$ runs in deterministic $2^{O(n)}$ time with access to an NP oracle, and produces a string DS of length $2^{O(n)}$.
- Let C be a $\mathscr{C}$ circuit with n inputs and size $s(n)$. $\mathsf{Query}(\langle C \rangle)$ runs in deterministic $2^n/n^{\omega(1)}$ time with random access to DS and outputs an estimation of $\Pr_{x \leftarrow \{0,1\}^n}[C(x) = 1]$ within an additive error of ε.

If the requirement for $\mathsf{Query}(\cdot)$ is replaced by the following, then we say $(\mathsf{Prep}, \mathsf{Query})$ is a $\mathsf{GapUNSAT}$ *data structure for* $\mathscr{C}[s(n)]$ *with* E^{NP} *preprocessing* and error $1-\varepsilon$:

- If C is unsatisfiable, then $\mathsf{Query}(\langle C \rangle)$ outputs 0; if $\Pr_{x \leftarrow \{0,1\}^n}[C(x) = 1] \geq 1 - \varepsilon$, then $\mathsf{Query}(\langle C \rangle)$ outputs 1.

If the requirement for $\mathsf{Query}(\cdot)$ is replaced by the following, then we say $(\mathsf{Prep}, \mathsf{Query})$ is a CAPP data structure for $\mathscr{C}[s(n)]$ with E^{NP} preprocessing *and inverse-circuit-size error*:

- $\mathsf{Query}(\langle C \rangle)$ outputs an estimation of $\Pr_{x \leftarrow \{0,1\}^n}[C(x) = 1]$ within an additive error of $1/s(n)$.

Finally, if the data structure is correct only for infinitely many numbers n, then we say the data structure is an *i.o.* CAPP (or *i.o.* $\mathsf{GapUNSAT}$) data structure.

Theorem 4.3.2. *Let $\mathscr{C}$ be a circuit class and $\mathrm{poly}(n) \leq s(n) \leq 2^{0.01n}$ be a good function such that the following technical conditions hold:*

($\mathscr{C}$ is complete) *For every truth table of length 2^k, there is a $\mathscr{C}$ circuit of size $\mathrm{poly}(2^k)$ that computes this truth table; moreover, the description of such a $\mathscr{C}$ circuit can be computed in deterministic $\mathrm{poly}(2^k)$ time from the truth table.*

($\mathscr{C}$ computes PARITY) *The PARITY function can be computed by a $\mathscr{C}$ circuit of size $\mathrm{poly}(n)$.*

There is a constant $\varepsilon \in (0,1)$ such that the following holds. If there is a $\mathsf{GapUNSAT}$ data structure for $\mathsf{NC}^0_3 \circ \mathscr{C}$ circuits of size $\mathrm{poly}(s(n))$ with E^{NP} preprocessing, query time $2^n/n^{\omega(1)}$, and error $1-\varepsilon$, then E^{NP} does not have size-$s(n)$ $\mathscr{C}$ circuits on almost every input length.

Proof of Theorem 4.3.2. Let $c \geq 1$ be a constant such that $\mathsf{Prep}(1^n)$ always outputs a data structure of length at most 2^{cn}. Suppose that any $\mathscr{C}$ circuit of size $s(n)$ can be described in bit-length $\ell \leq O(s(n)^2)$. In this proof, we set the following parameters (where $K \geq 1$ is a large enough constant):

$$\begin{aligned} m &:= 10(c+1), \\ w_{\mathsf{proof}} &:= n, & W_{\mathsf{proof}} &:= 2^{w_{\mathsf{proof}}} = 2^n, \\ h_{\mathsf{proof}} &:= cn, & H_{\mathsf{proof}} &:= 2^{h_{\mathsf{proof}}} = 2^{cn}, \\ w_{\mathsf{input}} &:= \log \ell + K \log n, & W_{\mathsf{input}} &:= 2^{w_{\mathsf{input}}} = \ell \cdot \mathrm{poly}(n), \\ N &:= 10 H_{\mathsf{proof}} \cdot \ell = 10 \cdot 2^{cn} \ell, \\ T &:= H_{\mathsf{proof}} \cdot W_{\mathsf{proof}} / \mathrm{poly}(n) = 2^{(c+1)n} / \mathrm{poly}(n). \end{aligned}$$

Our goal is to find, in $\mathsf{DTIME}[2^{O(n)}]^{\mathsf{NP}}$, a truth table of length W_{proof} that does not have size-$s(n)$ $\mathscr{C}$ circuits. Let L^{hard} be the language constructed in Theorem 3.3.1, i.e.,

$$L^{\mathsf{hard}} \in \mathsf{NTIME}_{\mathsf{TM}}[T] \setminus \text{i.o.-}\mathsf{NTIMEGUESS}_{\mathsf{RAM}}[T/\mathrm{polylog}(T), N/10]/_{(N/10)}.$$

(Note that $T \geq N^{1+\Omega(1)}$, so the technical conditions for applying Theorem 3.3.1 are satisfied.)

Now we construct a nondeterministic RAM M^{PCPP} that attempts to solve L^{hard}. Let $L^{\mathsf{enc}} := \{\mathsf{Encode}(x) : x \in L^{\mathsf{hard}}\}$ where Encode is the error-correcting code specified in Theorem 2.4.1. Suppose that the encodings of length-N strings have length $\tilde{N} = O(N)$ and let $\delta_{\mathsf{code}} > 0$ be the relative Hamming distance of Encode. Let VPCPP be the rectangular PCPP verifier for L^{enc} with proximity δ_{code}, perfect completeness, and soundness error $1-\varepsilon$ specified in Theorem 2.5.10, where $\varepsilon \in (0,1)$ is an absolute constant. The proof oracle π is an $\hat{H}_{\mathsf{proof}} \times W_{\mathsf{proof}}$ matrix and the input oracle is an $H_{\mathsf{input}} \times W_{\mathsf{input}}$ matrix, where

$$\begin{aligned} \hat{h}_{\mathsf{proof}} &= \log T + \Theta(m \log\log T) - w_{\mathsf{proof}} = cn + O(\log n) & \hat{H}_{\mathsf{proof}} &:= 2^{\hat{h}_{\mathsf{proof}}} = 2^{cn} \cdot \mathrm{poly}(n) \\ h_{\mathsf{input}} &= \lceil \log N \rceil - w_{\mathsf{input}} = cn - O(\log n) & H_{\mathsf{input}} &:= 2^{h_{\mathsf{input}}} = 2^{cn}/\mathrm{poly}(n). \end{aligned}$$

We verify the technical conditions of Theorem 2.5.10 hold:

- Clearly, $w_{\mathsf{proof}} \leq \log T$ and $w_{\mathsf{input}} \leq \log \tilde{N}$.
- $w_{\mathsf{proof}} \geq (5/m) \log T$: this is because $(5/m)\log T \leq \frac{5}{10(c+1)} \cdot (c+1)n = n/2$.
- $\hat{h}_{\mathsf{proof}} \geq (5/m) \log T$: this is because $(5/m)\log T \leq n/2 < cn$.
- $\frac{w_{\mathsf{input}}}{w_{\mathsf{proof}}} \leq 1 - \frac{Cm \log\log T}{\log T}$: this is because $\frac{w_{\mathsf{input}}}{w_{\mathsf{proof}}} = \frac{\log \ell + K \log n}{n} < 0.9$.
- $\frac{h_{\mathsf{input}}}{\hat{h}_{\mathsf{proof}}} \leq 1 - \frac{Cm \log\log T}{\log T}$: this is because $1 - \frac{h_{\mathsf{input}}}{\hat{h}_{\mathsf{proof}}} \geq 1 - \frac{cn - K\log n}{cn + O(\log n)} \geq \frac{K \log n}{(c+1)n} \geq \frac{Cm \log\log T}{\log T}$.

The number of column randomness is $r_{\mathsf{col}} = w_{\mathsf{proof}} - (5/m)\log T = n/2 + O(\log n)$.

The speed-up machine M^{PCPP} assumes that every row of the proof oracle is the truth table of some circuit in $\mathscr{C}[s(n)]$. It guesses $\hat{H}_{\mathsf{proof}}$ size-$s(n)$ $\mathscr{C}$ circuits $C_1, C_2, \ldots, C_{\hat{H}_{\mathsf{proof}}}$ such that the i-th row of the proof matrix π is the truth table of C_i. Now it remains to estimate

$$p_{\mathsf{acc}} := \Pr_{\mathsf{seed}}[\mathsf{VPCPP}^{\mathsf{Encode}(x) \circ \Pi}(\mathsf{seed}) \text{ accepts}].$$

We first enumerate $\mathsf{seed.shared}$ and $\mathsf{seed.row}$. Suppose VDec takes as inputs q query answers

and p parity-check bits, then $p + q = 3$. (Note that p and q might depend on $\mathsf{seed.shared}$ but not $\mathsf{seed.row}$ or $\mathsf{seed.col}$.) Define

$$\begin{aligned}(\mathsf{itype}[1], \ldots, \mathsf{itype}[q]) &:= V_{\mathsf{type}}(\mathsf{seed.shared}),\\ (\mathsf{irow}[1], \ldots, \mathsf{irow}[q]) &:= V_{\mathsf{row}}(\mathsf{seed.row}, \mathsf{seed.shared}), \text{ and}\\ (\mathsf{icol}[1], \ldots, \mathsf{icol}[q]) &:= V_{\mathsf{col}}(\mathsf{seed.col}, \mathsf{seed.shared}).\end{aligned}$$

(Note that as we only enumerated $\mathsf{seed.shared}$ and $\mathsf{seed.row}$, for each $\iota \in [q]$, $\mathsf{itype}[\iota]$ and $\mathsf{irow}[\iota]$ are already fixed, but $\mathsf{icol}[\iota]$ are functions of $\mathsf{seed.col}$.) We need to estimate

$$p_{\mathsf{acc}}(\mathsf{seed.row}, \mathsf{seed.shared}) := \Pr_{\mathsf{seed.col}}[\mathsf{VPCPP}^{\mathsf{Encode}(x)\circ\Pi}(\mathsf{seed}) \text{ accepts}]. \tag{4.3}$$

Now, we create the following circuit $C' := C'_{\mathsf{seed.row},\mathsf{seed.shared}}$ that takes $\mathsf{seed.col}$ as input and accepts if and only if $\mathsf{VPCPP}^{\mathsf{Encode}(x)\circ\Pi}(\mathsf{seed})$ accepts. For every $\iota \in [q]$:

- Suppose $\mathsf{itype}[\iota] = \mathsf{proof}$. Let D_ι be the circuit such that $D_\iota(\mathsf{seed.col}) = C_{\mathsf{irow}[\iota]}(\mathsf{icol}[\iota])$. Since $\mathsf{icol}[\iota]$ is a projection over $\mathsf{seed.col}$ and $\mathscr{C}$ is closed under projections, D_ι can be computed by a $\mathscr{C}$ circuit of size $\mathrm{poly}(\ell)$.
- Suppose $\mathsf{itype}[\iota] = \mathsf{input}$. Let $D'_{\mathsf{irow}[\iota]}$ be the $\mathscr{C}$ circuit whose truth table is the $\mathsf{irow}[\iota]$-th row of the input matrix $\mathsf{Encode}(x)$. Since $\mathscr{C}$ is complete, the size of $D'_{\mathsf{irow}[\iota]}$ is at most $\mathrm{poly}(W_{\mathsf{input}}) \le \mathrm{poly}(n, \ell)$ and the description of $D'_{\mathsf{irow}[\iota]}$ can be computed efficiently. Let D_ι be the circuit such that $D_\iota(\mathsf{seed.col}) := D'_{\mathsf{irow}[\iota]}(\mathsf{icol}[\iota])$, then D_ι can also be computed by a $\mathscr{C}$ circuit of size $\mathrm{poly}(n, \ell)$.

We also construct a circuit $PC_\iota(\mathsf{seed.col})$ to compute the ι-th parity-check bit. In particular, as we have already fixed $\mathsf{seed.row}$ and $\mathsf{seed.shared}$, the ι-th parity-check bit is simply the XOR of a subset of indices in $\mathsf{seed.col}$. Since PARITY can be computed by a $\mathscr{C}$ circuit of polynomial size[1], it follows that PC_i is also a $\mathscr{C}$ circuit of size $\mathrm{poly}(r_{\mathsf{col}}) \le \mathrm{poly}(n)$.

Let VDec be the decision predicate of VPCPP. The circuit C' is simply

$$C'(\mathsf{seed.col}) := \mathsf{VDec}(D_1(\mathsf{seed.col}), \ldots, D_q(\mathsf{seed.col}), PC_1(\mathsf{seed.col}), \ldots, PC_q(\mathsf{seed.col})).$$

We can see that C' is an $\mathsf{NC}^0_3 \circ \mathscr{C}$ circuit of size $\mathrm{poly}(s(n))$. Therefore, we can use the $\mathsf{GapUNSAT}$ data structure to distinguish between the case that $p_{\mathsf{acc}}(\mathsf{seed.row}, \mathsf{seed.shared}) = 1$ and the case that $p_{\mathsf{acc}}(\mathsf{seed.row}, \mathsf{seed.shared}) \le 1 - \varepsilon$, in time $2^{r_{\mathsf{col}}}/(r_{\mathsf{col}})^{\omega(1)}$. Since $r_{\mathsf{col}} \ge \Omega(n)$, this is also in time $2^{r_{\mathsf{col}}}/n^{\omega(1)}$. If the overall acceptance probability p_{acc} is at most $1 - \varepsilon$, then at least one of $p_{\mathsf{acc}}(\mathsf{seed.row}, \mathsf{seed.shared})$ should be at most $1 - \varepsilon$; if $p_{\mathsf{acc}} = 1$, then every $p_{\mathsf{acc}}(\mathsf{seed.row}, \mathsf{seed.shared})$ should be equal to 1. Therefore, we can also distinguish between $p_{\mathsf{acc}} = 1$ and $p_{\mathsf{acc}} \le 1 - \varepsilon$.

The total running time of M^{PCPP} is thus $2^{|\mathsf{seed}|}/n^{\omega(1)} < T/\log^{\omega(1)} T$. The number of nondeterministic bits that M^{PCPP} guesses is $H_{\mathsf{proof}} \cdot \ell \le N/10$. The number of advice bits that M^{PCPP}

[1] Actually, we do not need the small $\mathscr{C}$ circuit for computing PARITY to be efficiently computable (i.e., uniform) here, since we can guess-and-check a $\mathscr{C}$ circuit computing PARITY during the preprocessing phase in $\mathsf{DTIME}[2^{O(n)}]^{\mathsf{NP}}$.

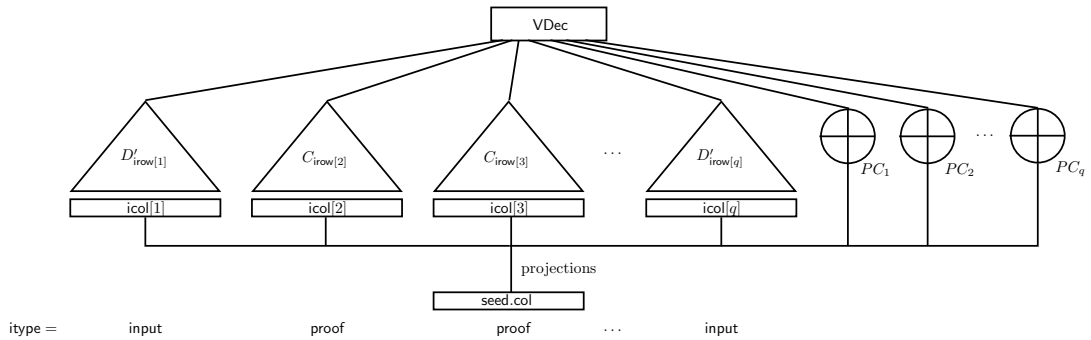


Figure 4.1: The circuit C'. It is easy to see that $C' \in \mathsf{NC}^0_d \circ \mathscr{C}$.

uses is $2^{cn} \leq N/10$. Therefore, M^{PCPP} decides a language in

$$\mathsf{NTIMEGUESS}_{\mathsf{RAM}}[T/\log^{\omega(1)} T, N/10]/_{(N/10)}.$$

It follows from the definition of L^{hard} that M^{PCPP} fails to compute L^{hard} on some input of length N, for every large enough N. We use the P^{NP} refuter in Theorem 3.3.1 to find an input x_{hard} where $L^{\mathsf{hard}}(x_{\mathsf{hard}}) \neq M^{\mathsf{PCPP}}(x_{\mathsf{hard}})$. By the construction of M^{PCPP}, it has to be the case that $x_{\mathsf{hard}} \in L^{\mathsf{hard}}$ but $M^{\mathsf{PCPP}}(x_{\mathsf{hard}}) = 0$. In this case, for any valid PCPP proof π of "$\mathsf{Encode}(x_{\mathsf{hard}}) \in L^{\mathsf{enc}}$", if we treat π as an $\hat{H}_{\mathsf{proof}} \times W_{\mathsf{proof}}$ matrix, then there has to be some row of π which is not the truth table of any size-$s(n)$ $\mathscr{C}$ circuit. We simply find a valid PCPP proof π and output the row of π that does not have size-$s(n)$ $\mathscr{C}$ circuits. This can be done in $\mathsf{TIME}[2^{O(n)}]^{\mathsf{NP}}$. □

Remark 4.3.3. We remark that the hypothesis about GapUNSAT data structure in Theorem 4.3.2 can be replaced by the following: There is a constant $\varepsilon \in (0,1)$ such that for every constant $d \geq 1$, there is a GapUNSAT data structure for $\mathsf{NC}^0_d \circ \mathscr{C}$ circuits of size $\mathrm{poly}(s(n))$ with E^{NP} preprocessing and error $1-\varepsilon$. (For comparison, in Theorem 4.3.2 we assumed a GapUNSAT data structure with good accuracy, but only for $\mathsf{NC}^0_3 \circ \mathscr{C}$; here we assume a GapUNSAT data structure with potentially bad accuracy but handles $\mathsf{NC}^0_d \circ \mathscr{C}$ for every constant $d \geq 1$.) Using a rectangular PCPP with smaller soundness error and larger query complexity (e.g., by parallel repetition) in the proof, we can still show that E^{NP} cannot be computed by size-$s(n)$ $\mathscr{C}$ circuits on almost every input length.

To show equivalences in Section 4.6, we also need an infinitely-often version of the above theorem.

Corollary 4.3.4. *Let $\mathscr{C}$ be a circuit class that is complete and computes* PARITY. *Let* $\mathrm{poly}(n) \leq s(n) \leq 2^{0.01n}$ *be a good function, then there is a constant $\varepsilon \in (0,1)$ such that the following holds. If there is an i.o.* GapUNSAT *data structure for $\mathsf{NC}^0_3 \circ \mathscr{C}$ circuits of size* $\mathrm{poly}(s(n))$ *with E^{NP} preprocessing, query time $2^n/n^{\omega(1)}$, and error $1-\varepsilon$, then E^{NP} does not have size-$s(n)$ $\mathscr{C}$ circuits.*

Proof Sketch. As the proof is almost the same as the proof of Theorem 4.3.2, we will only sketch the differences here.

We define M^{PCPP} in the same way as Theorem 4.3.2. Then, M^{PCPP} fails to compute L^{hard} on all large enough input lengths. Note that the correctness of M^{PCPP} on input length n only depends on the correctness of the (i.o.) GapUNSAT data structure on input length $r_{\mathsf{col}} =$

$n/2 + O(\log n)$. For every integer r_{col} such that the i.o. GapUNSAT data structure is correct, the "easy-witness" assumption fails when $n = 2r_{\mathsf{col}} - O(\log r_{\mathsf{col}})$. That is, for such integers n, any valid PCPP proof π for $x_{\mathsf{hard}} \in \{0,1\}^n$ contains some row that is not the truth table of any size-$s(n)$ circuit. □

Remark 4.3.5 (Comparison with [BV14]). Both [BV14] and our Theorem 4.3.2 need to use PCP with projection queries to reduce the circuit complexity overhead of the Algorithmic Method. To achieve this, [BV14] constructed PCPs where the query indices are computable by a projection over seed. To achieve this property, [BV14] needed to use the PCP in [BS08]. Unfortunately, this PCP requires polylog(n) queries; even worse, the "projection" property is broken when we use PCP composition to reduce query complexity to $O(1)$.

However, if we allow the queries to depend arbitrarily on a small portion of seed (namely seed.shared), but have to be a projection over the rest of the bits, then this is also achievable using the PCP in [BGH+06]. The [BGH+06] PCP has the advantage of being almost rectangular. We are also able to compose PCPs now, by simply adding the (very short) randomness of the inner PCP into seed.shared. Thus, the query complexity can be reduced to $O(1)$. Having such a small portion (i.e., seed.shared) does not hurt the Algorithmic Method at all.

4.4 Strong Average-Case Circuit Lower Bounds

We strengthen Theorem 4.3.2 to the case of strong average-case lower bounds. We first show that non-trivial CAPP algorithms with E^{NP} preprocessing implies (weak average-case) lower bounds against $[0,1]$-Sum $\circ\, \mathscr{C}$ circuits:

Theorem 4.4.1. *Let $\mathscr{C}$ be a circuit class and* poly$(n) \le s(n) \le 2^{0.01n}$ *be a good function that is monotone, such that the following technical conditions hold:*

($\mathscr{C}$ is complete) *For every truth table of length 2^k, there is a $\mathscr{C}$ circuit of size $O(2^{10k})$ that computes this truth table; moreover, the description of such a $\mathscr{C}$ circuit can be computed in deterministic* poly(2^k) *time from the truth table.*

($\mathscr{C}$ computes PARITY) *The* PARITY *function can be computed by a $\mathscr{C}$ circuit of size* poly(n).

There are absolute constants $\delta > 0$ and $d \ge 2$ such that the following holds. Suppose that for every constant $k \ge 1$, there is a CAPP *data structure for* AND$_d \circ \mathscr{C}$ *circuits of size $s(2r)^k$ and r inputs in $2^r/s(2r)^k$ query time with E^{NP} preprocessing and inverse-circuit-size error, then there is a language in E^{NP} that has ℓ_1-distance at least δ from $[0,1]$-Sum $\circ\, \mathscr{C}$ circuits of complexity $s(n)$ on almost every input length.*

Proof. Let $c \ge 1$ be a constant such that $\mathsf{Prep}(1^n)$ always outputs a data structure of length at most 2^{cn}. Let $\ell \le O(s(n)^{10})$ be such that any Sum $\circ\, \mathscr{C}$ circuit of complexity $O(s(n))$ can be described in bit-length ℓ. We set the following parameters (where K is a large enough universal constant):

$$m := 10(c+1),$$

$$w_{\mathsf{proof}} := n, \qquad W_{\mathsf{proof}} := 2^{w_{\mathsf{proof}}} = 2^n,$$

$$h_{\mathsf{proof}} := cn, \qquad H_{\mathsf{proof}} := 2^{h_{\mathsf{proof}}} = 2^{cn},$$

$$w_{\mathsf{input}} := \log \ell + K \cdot \log n, \qquad W_{\mathsf{input}} := 2^{w_{\mathsf{input}}} = \ell \cdot \mathrm{poly}(n),$$
$$N := 10 H_{\mathsf{proof}} \cdot \ell \cdot \mathrm{poly}(n^K) = 10 \cdot 2^{cn} \mathrm{poly}(n) \ell,$$
$$T := H_{\mathsf{proof}} \cdot W_{\mathsf{proof}} / \mathrm{poly}(n) = 2^{(c+1)n} / \mathrm{poly}(n).$$

Let L^{hard} be the language constructed in Theorem 3.3.1, i.e.,

$$L^{\mathsf{hard}} \in \mathsf{NTIME}_{\mathsf{TM}}[T] \setminus \text{i.o.-}\mathsf{NTIMEGUESS}_{\mathsf{RAM}}[T/\mathrm{polylog}(T), N/10]/_{(N/10)}.$$

Now we construct a nondeterministic RAM M^{PCPP} that attempts to solve L^{hard}. Let $L^{\mathsf{enc}} := \{\mathsf{Encode}(x) : x \in L^{\mathsf{hard}}\}$ where Encode is the error-correcting code specified in Theorem 2.4.1. Suppose that the encodings of length-N strings have length $\tilde{N} = O(N)$. By Theorem 2.5.11, there are good functions $\hat{h}_{\mathsf{proof}}$ and h_{input} satisfying:

$$\hat{h}_{\mathsf{proof}} = \log T + \Theta(m \log\log T) - w_{\mathsf{proof}} = cn + O(\log n), \quad \hat{H}_{\mathsf{proof}} = 2^{h_{\mathsf{proof}}} = 2^{cn} \cdot \mathrm{poly}(n), \quad \text{and}$$
$$h_{\mathsf{input}} = \lceil \log \tilde{N} \rceil - w_{\mathsf{input}} = cn - \Omega(K \log n), \qquad H_{\mathsf{input}} = 2^{h_{\mathsf{input}}} = 2^{cn} / \mathrm{poly}(n),$$

such that L^{hard} has a smooth and rectangular PCPP with an $H_{\mathsf{input}} \times W_{\mathsf{input}}$ input matrix and an $\hat{H}_{\mathsf{proof}} \times W_{\mathsf{proof}}$ proof matrix, and has the following parameters:

- soundness error $= 1/2$,
- proximity parameter $= \delta_{\mathsf{code}}$ (which is the relative distance of Encode),
- query complexity $= q := O(1)$,
- parity-check complexity $= q = O(1)$,
- total randomness $= r := \log T + O(m \log\log T) = (c+1)n + O(\log n)$,
- row randomness $= r_{\mathsf{row}} := h_{\mathsf{proof}} - (5/m) \log T$,
- column randomness $= r_{\mathsf{col}} := w_{\mathsf{proof}} - (5/m) \log T = n/2 + O(\log n)$,
- shared randomness $= r_{\mathsf{shared}} := (10/m) \log T + O(\log\log T + m \log m)$.

Of course, we need to check that the technical conditions of Theorem 2.5.11 are satisfied:

- $w_{\mathsf{proof}} \geq (5/m) \log T$: in fact, $\frac{w_{\mathsf{proof}} \cdot m}{5 \log T} \geq \frac{10n(c+1)}{5n(c+1) - O(\log n)} \geq 2$.
- $h_{\mathsf{proof}} \geq (5/m) \log T$: since $h_{\mathsf{proof}} > w_{\mathsf{proof}}$, this follows easily.
- $\frac{w_{\mathsf{input}}}{w_{\mathsf{proof}}} \leq 1 - \frac{Cm^2 \log\log T}{\log T}$: because $\frac{w_{\mathsf{input}}}{w_{\mathsf{proof}}} \leq \frac{\log \ell + O(\log n)}{n} \leq \frac{10 \log s + O(\log n)}{n}$ and $s \leq 2^{0.01n}$.
- $\frac{h_{\mathsf{input}}}{\hat{h}_{\mathsf{proof}}} \leq 1 - \frac{Cm^2 \log\log T}{\log T}$: because $\frac{h_{\mathsf{input}}}{\hat{h}_{\mathsf{proof}}} = 1 - \frac{\Omega(K \log n)}{cn + O(\log n)} \leq 1 - \frac{Cm^2 \log n}{(c+1) \log n}$ if K is a large enough constant compared to C.

We remark that q only depends on the soundness error (which is $1/2$) and the proximity parameter (which is δ_{code}), both of which are absolute constants; hence q is also an absolute constant. We set $\delta := 0.1^{10q^4}/q$, which is also an absolute constant.

By our hypothesis, for $r_{\mathsf{col}} = n/2 + O(\log n)$, there is a CAPP data structure for $\mathsf{AND}_{4q} \circ \mathscr{C}$ circuits of size $s(n)^{K'}$ and r_{col} inputs in $T^{\mathrm{alg}} := 2^{r_{\mathsf{col}}}/s(n)^{K'}$ query time with E^{NP} preprocessing and additive error $\varepsilon_{\mathrm{alg}} := s(n)^{-K'}$, where K' is a large enough constant.

The speed-up machine M^{PCPP} receives an input $x \in \{0,1\}^N$. If $x \in L^{\mathsf{hard}}$, then there is an $\hat{H}_{\mathsf{proof}} \times W_{\mathsf{proof}}$ PCPP proof Π such that $\mathsf{VPCPP}^{\mathsf{Encode}(x), \Pi}(\mathsf{seed})$ accepts with probability 1. We say that x has *easy witness*, if furthermore, every row of this PCPP proof Π is the truth table of a function $\pi_i : \{0,1\}^{w_{\mathsf{proof}}} \to \{0,1\}$ that is δ-close to a $[0,1]$-$\mathsf{Sum} \circ \mathscr{C}$ circuit of complexity $s(n)$ in ℓ_1-distance. Assuming that x has easy witness, we nondeterministically guess $\hat{H}_{\mathsf{proof}}$ $\mathsf{Sum} \circ \mathscr{C}$

circuits $C_1, C_2, \ldots, C_{\hat{H}_{\mathsf{proof}}}$, each of complexity $s(n)$, hoping that for every $i \in [\hat{H}_{\mathsf{proof}}]$, C_i is a $[0,1]$-$\mathsf{Sum} \circ \mathscr{C}$ circuit that $(1-\delta)$-approximates π_i in ℓ_1-distance. Then, for each $i \in [\hat{H}_{\mathsf{proof}}]$, we define $B_i := \mathsf{bin}_{C_i}$ to be the Boolean function closest to C_i. Note that B_i might be different from π_i. However, in the case that $x \in L^{\mathsf{hard}}$ and x has easy witness, we have

$$\|\pi_i - B_i\|_1 \leq \|\pi_i - C_i\|_1 + \|C_i - B_i\|_1 \leq 2\delta.$$

Since both π_i and B_i are Boolean, the relative Hamming distance between π_i and B_i is at most 2δ. Let Π_B be the $\hat{H}_{\mathsf{proof}} \times W_{\mathsf{proof}}$ matrix where the i-th row is the truth table of B_i. Since VPCPP is smooth, it follows from Lemma 2.5.8 that VPCPP accepts the proof Π_B w.p. at least $1 - 2q\delta \geq 0.98$. On the other hand, if $x \notin L^{\mathsf{hard}}$, then VPCPP accepts Π_B (in fact any proof) w.p. at most $1/2$. From now on, we forget Π and our goal becomes to estimate

$$\Pr_{\mathsf{seed}}[\mathsf{VPCPP}^{\mathsf{Encode}(x),\Pi_B}(\mathsf{seed}) \text{ accepts}] \tag{4.4}$$

within an additive error of 0.1.

For every $\mathsf{seed} = (\mathsf{seed.row}, \mathsf{seed.col}, \mathsf{seed.shared})$, define

$$\begin{aligned}(\mathsf{itype}[1], \mathsf{itype}[2], \ldots, \mathsf{itype}[q]) &:= V_{\mathsf{type}}(\mathsf{seed.shared}),\\ (\mathsf{irow}[1], \mathsf{irow}[2], \ldots, \mathsf{irow}[q]) &:= V_{\mathsf{row}}(\mathsf{seed.row}, \mathsf{seed.shared}), \text{ and}\\ (\mathsf{icol}[1], \mathsf{icol}[2], \ldots, \mathsf{icol}[q]) &:= V_{\mathsf{col}}(\mathsf{seed.col}, \mathsf{seed.shared}).\end{aligned}$$

For now, let us think of $\mathsf{seed.shared}$ and $\mathsf{seed.row}$ as fixed and $\mathsf{seed.col}$ as variables. Then, for each $\iota \in [q]$, $\mathsf{itype}[\iota]$ and $\mathsf{irow}[\iota]$ are also fixed, and $\mathsf{icol}[\iota]$ is a function of $\mathsf{seed.col}$. Moreover, each $\mathsf{icol}[\iota]$ can be computed by a projection over $\mathsf{seed.col}$, and the description of this projection can be computed in polynomial time given $\mathsf{seed.shared}$.

Let VDec be the decision predicate of VPCPP (which depends on $\mathsf{seed.shared}$), then VDec takes $2q$ input bits where the first q input bits correspond to the query answers and the last q bits correspond to parity check bits. Our goal is to estimate

$$\Pr_{\mathsf{seed}}[\mathsf{VDec}(D_1(\mathsf{seed.col}), D_2(\mathsf{seed.col}), \ldots, D_{2q}(\mathsf{seed.col})) = 1].$$

Here, for each $\iota \in [2q]$, D_ι is a Boolean function over $\mathsf{seed.col}$ (which depends on $\mathsf{seed.shared}$ and $\mathsf{seed.row}$) specified as follows.

- If ι corresponds to a query and $\mathsf{itype}[\iota] = \mathsf{proof}$, then $D_\iota(\mathsf{seed.col}) = B_{\mathsf{irow}[\iota]}(\mathsf{icol}[\iota])$.
- If ι corresponds to a query and $\mathsf{itype}[\iota] = \mathsf{input}$, then $D_\iota(\mathsf{seed.col}) = \mathsf{Encode}(x)_{\mathsf{irow}[\iota],\mathsf{icol}[\iota]}$.
- If ι corresponds to a parity check bit, then $D_\iota(\mathsf{seed.col})$ is the appropriate parity function over $\mathsf{seed.col}$.

Since every Boolean function over $2q$ bits is equivalent to a degree-$2q$ polynomial over the reals, we can write

$$\mathsf{VDec}(a_1, a_2, \ldots, a_{2q}) = \sum_{S \subseteq [2q]} \theta_S \prod_{\iota \in S} a_\iota,$$

where $\theta_S \in [-2^{2q}, 2^{2q}]$. For each $S \subseteq [2q]$, we will estimate

$$\mathop{\mathbb{E}}_{\mathsf{seed.col}}\left[\prod_{\iota \in S} D_\iota(\mathsf{seed.col})\right]. \tag{4.5}$$

We define the $\mathsf{Sum} \circ \mathscr{C}$ circuits $\tilde{D}_\iota$ as follows. If ι corresponds to a query bit such that $\mathsf{itype}[\iota] = \mathsf{proof}$, then $\tilde{D}_\iota(\mathsf{seed.col}) = C_{\mathsf{irow}[\iota]}(\mathsf{icol}[\iota])$; otherwise $\tilde{D}_\iota = D_\iota$. We will actually use

$$\mathop{\mathbb{E}}_{\mathsf{seed.col}}\left[\prod_{\iota \in S} \tilde{D}_\iota(\mathsf{seed.col})\right] \tag{4.6}$$

as an estimation of (4.5). Intuitively, we need to estimate the accept probability of VPCPP when given Π_B as the oracle, but since we do not have direct access to Π_B, we use the "real-valued proof" encoded by $\{C_i\}$ instead. We note that since VPCPP has projection queries (i.e., $\mathsf{icol}[\iota]$ is computable by a projection over $\mathsf{seed.col}$), each $\tilde{D}_\iota$ is indeed a $\mathsf{Sum} \circ \mathscr{C}$ circuit over $\mathsf{seed.col}$ whose description can be computed efficiently given $\mathsf{seed.shared}$ and $\mathsf{seed.row}$. The complexity of each $\tilde{D}_\iota$ is at most $\hat{s} := \max\{s(n), \mathrm{poly}(W_{\mathsf{input}})\} \leq \mathrm{poly}(s(n))$.

Testing each C_i. It is *not* guaranteed that our guessed circuits C_i will satisfy the $[0,1]$-Sum promise and will be δ-close to B_i in ℓ_1 norm. Hence, we also need to test whether each C_i satisfies these promises. Like in [CLW20] and in Chapter 3, our verification algorithm will be *approximate*: If each C_i satisfies the $[0,1]$-Sum promise and is δ-close to B_i, then we will accept them; on the other hand, if we accept the circuits C_i, then they are "somewhat close" to B_i.

More precisely, we perform the following verification:

1. First, we apply Lemma 4.2.6 on each C_i with parameter $d := 2q$ and $\delta' := 3\delta^{1/d}$. If any C_i is rejected by the test, then we reject. This takes $\hat{H}_{\mathsf{proof}} \cdot T^{\mathrm{alg}} \cdot O(\hat{s}^{4q})$ time in total.

   If C_i is a $[0,1]$-$\mathsf{Sum} \circ \mathscr{C}$ circuit and $\|C_i - B_i\|_1 \leq \delta$, then the test accepts; if the test accepts, then $\|C_i - B_i\|_{2q} \leq \delta'$.

2. Then, we enumerate $\mathsf{seed.shared}$ and $\mathsf{seed.row}$. For each $(\mathsf{seed.shared}, \mathsf{seed.row})$ and each $\iota \in [2q]$ such that ι corresponds to a query with $\mathsf{itype}[\iota] = \mathsf{proof}$, we apply Theorem 4.2.5 to estimate $\mathbb{E}_{\mathsf{seed.col}}[(\tilde{D}_\iota(\mathsf{seed.col}))^{2q}]$. Note that since $\mathsf{seed.shared}$ and $\mathsf{seed.row}$ are fixed, each $\tilde{D}_\iota$ is a $\mathsf{Sum} \circ \mathscr{C}$ circuit over the input $\mathsf{seed.col}$, hence we can apply Theorem 4.2.5 on $\tilde{D}_\iota$. This takes $2^{r-r_{\mathsf{col}}} q \hat{s}^{2q+O(1)} \cdot T^{\mathrm{alg}}$ time and provides an estimation within additive error $\hat{s}^{2q} \cdot \varepsilon_{\mathrm{alg}}$. If the estimation exceeds $1 + \hat{s}^{2q} \cdot \varepsilon_{\mathrm{alg}}$ then we reject immediately.

   Clearly, if every C_i is a $[0,1]$-$\mathsf{Sum} \circ \mathscr{C}$ circuit, then $\mathbb{E}_{\mathsf{seed.col}}[(\tilde{D}_\iota(\mathsf{seed.col}))^{2q}] \leq 1$ and our test always accepts. If the test accepts, then for every combination of $(\mathsf{seed.shared}, \mathsf{seed.row}, \iota)$, we have $\mathbb{E}_{\mathsf{seed.col}}[(\tilde{D}_\iota(\mathsf{seed.col}))^{2q}] \leq 1 + 2\hat{s}^{2q} \cdot \varepsilon_{\mathrm{alg}}$.

If the $\mathsf{Sum} \circ \mathscr{C}$ circuits $\{C_i\}$ pass both tests above, then we proceed to estimate (4.6). We enumerate $\mathsf{seed.shared}$, $\mathsf{seed.row}$, and S, which determines the circuits $\tilde{D}_\iota$ and the coefficient θ_S. Then we apply Theorem 4.2.5 on the circuits $\{\tilde{D}_\iota\}_{\iota \in S}$, which allows us to obtain an estimation

of (4.6) within additive error $\hat{s}^{2q} \cdot \varepsilon_{\text{alg}}$. Finally, we output the estimation of

$$\mathop{\mathbb{E}}_{\substack{\mathsf{seed.shared}\\ \mathsf{seed.row}}}\left[\sum_{S\subseteq[2q]} \theta_{S,\mathsf{seed.shared}} \cdot \mathop{\mathbb{E}}_{\mathsf{seed.col}}\left[\prod_{\iota\in S} \tilde{D}_\iota(\mathsf{seed.col})\right]\right], \tag{4.7}$$

within an additive error of $4^q \cdot \hat{s}^{2q} \cdot \varepsilon_{\text{alg}} < 0.001$. (Recall that θ_S depends on not only S but also $\mathsf{seed.shared}$.) Overall, this takes $(2^{r-r_{\text{col}}} + \hat{H}_{\mathsf{proof}}) \cdot 2^{2q} \cdot \hat{s}^{4q} \cdot T^{\text{alg}}$ time.

Estimating the acceptance probability. Next, we show that if the tests above are passed, then (4.7) is a good estimation of (4.4). For ease of notation, let us denote $\overline{\mathsf{seed.col}} := (\mathsf{seed.shared}, \mathsf{seed.row})$. Fixing $\overline{\mathsf{seed.col}}$, we can treat D_ι and $\tilde{D}_\iota$ as functions over $\mathsf{seed.col}$ and talk about their norms (we alert the reader again that D_ι and $\tilde{D}_\iota$ depend on $\overline{\mathsf{seed.col}}$). In particular, we define

$$\begin{aligned} f(\overline{\mathsf{seed.col}}, \iota) &:= \|D_\iota - \tilde{D}_\iota\|_{2q}, \\ \text{which by definition } &= \mathop{\mathbb{E}}_{\mathsf{seed.col}}[(D_\iota(\mathsf{seed.col}) - \tilde{D}_\iota(\mathsf{seed.col}))^{2q}]^{1/(2q)}. \end{aligned}$$

Since each D_ι is Boolean, we have that $\|D_\iota\|_{2q} \le 1$. By triangle inequality, $\|\tilde{D}_\iota\|_{2q} \le 1 + f(\overline{\mathsf{seed.col}}, \iota)$. Define $f(\overline{\mathsf{seed.col}}) := \sum_{\iota\in[2q]} f(\overline{\mathsf{seed.col}}, \iota)$, then by Lemma 4.2.3,

$$\begin{aligned} &|(4.7) - (4.4)| \\ \le &\mathop{\mathbb{E}}_{\mathsf{seed.col}}\left[\sum_{S\subseteq[2q]} \theta_{S,\mathsf{seed.shared}} \cdot \left|\mathop{\mathbb{E}}_{\mathsf{seed.col}}\left[\prod_{\iota\in S} D_\iota(\mathsf{seed.col})\right] - \mathop{\mathbb{E}}_{\mathsf{seed.col}}\left[\prod_{\iota\in S} \tilde{D}_\iota(\mathsf{seed.col})\right]\right|\right] \\ \le &16^q \cdot \mathop{\mathbb{E}}_{\mathsf{seed.col}}[(2q) \cdot (1 + f(\overline{\mathsf{seed.col}}))^{2q-1} \cdot f(\overline{\mathsf{seed.col}})]. \end{aligned} \tag{4.8}$$

We can bound (4.8) using Lemma 3.5.7 which we restate here for convenience:

Lemma 3.5.7. *Let $f : [N] \times [q] \to \mathbb{R}_{\ge 0}$ be a function and $d \ge 1$ be a constant. Suppose that*

1. *for every $s \in [N]$ and $i \in [q]$, $f(s,i) \le \alpha$ (where $\alpha \ge 1$);*
2. *$\mathbb{E}_{s,i}[f(s,i)^d] \le \delta$.*

Let $f(s) := \sum_{i\in[q]} f(s,i)$. Then

$$\mathop{\mathbb{E}}_{s}[(1 + f(s))^{d-1} \cdot f(s)] \le q\delta^{1/d}(2q\alpha)^{d-1}.$$

Since $\{C_i\}$ passes the above tests, by triangle inequality, for every $\overline{\mathsf{seed.col}}$ we have

$$f(\overline{\mathsf{seed.col}}, \iota) \le 2 + 2\hat{s}^{2q} \cdot \varepsilon_{\text{alg}} \le 3.$$

We also have that

$$\begin{aligned} &\mathop{\mathbb{E}}_{\mathsf{seed.col},\iota}[f(\overline{\mathsf{seed.col}}, \iota)^{2q}] \\ = &\mathop{\mathbb{E}}_{\mathsf{seed},\iota}[(D_\iota(\mathsf{seed.col}) - \tilde{D}_\iota(\mathsf{seed.col}))^{2q}] \end{aligned}$$

$$\leq \mathop{\mathbb{E}}_{i\leftarrow[\hat{H}_{\mathsf{proof}}],j\leftarrow[W_{\mathsf{proof}}]}[(B_i(j)-C_i(j))^{2q}] \tag{4.9}$$
$$\leq \mathop{\mathbb{E}}_{i\leftarrow[\hat{H}_{\mathsf{proof}}]}[\|B_i-C_i\|_{2q}^{2q}] = (\delta')^{2q}.$$

Here, (4.9) is because of the smoothness of VPCPP: the distribution over $(\mathsf{irow}[\iota],\mathsf{icol}[\iota])$ conditioned on ι corresponds to a query to the proof oracle is equal to the uniform distribution over $[\hat{h}_{\mathsf{proof}}]\times[W_{\mathsf{proof}}]$.

It follows from Lemma 3.5.7 that

$$(4.8) \leq 16^q\cdot 2q\cdot(2q)\delta'(6q)^{2q-1}\leq 0.001,$$

hence we can estimate (4.4) within additive error 0.002. This suffices for distinguishing between the case that VPCPP accepts Π_B with probability at least 0.98 and the case that any PCPP proof is accepted w.p. at most $1/2$.

Clearly, M^{PCPP} runs in at most $(2^{r-r_{\mathsf{col}}}+\hat{H}_{\mathsf{proof}})\cdot O(\hat{s})^{4q}\cdot T^{\mathrm{alg}}\leq T/\mathrm{polylog}(T)$ time. Also, M^{PCPP} needs to guess $\hat{H}_{\mathsf{proof}}$ many $\mathsf{Sum}\circ\mathscr{C}$ circuits of complexity s. Since each such circuit can be described in ℓ bits, M^{PCPP} only guesses $\ell\cdot\hat{H}_{\mathsf{proof}}<N/10$ many nondeterministic bits. Also, M^{PCPP} takes the output of $\mathsf{Prep}(1^n)$ as non-uniform advice, which has length at most $H_{\mathsf{proof}}<N/10$. It follows that M^{PCPP} computes a language in

$$\mathsf{NTIMEGUESS}_{\mathsf{RAM}}[T/\mathrm{polylog}(T),N/10]/_{(N/10)}.$$

Wrap up. We design the following algorithm that on input 1^n, runs in deterministic $\mathrm{poly}(2^n)$ time with an NP oracle, and outputs the truth table of a function $f:\{0,1\}^n\to\{0,1\}$ that is δ-far in ℓ_1-distance from any $[0,1]$-$\mathsf{Sum}\circ\mathscr{C}$ circuit of complexity s. The algorithm first runs $\mathsf{Prep}(1^n)$ and computes the description of the machine M^{PCPP} as well as its advice α. Then it computes $x_{\mathsf{hard}}:=\mathcal{R}(1^n,M^{\mathsf{PCPP}},\alpha)$ where $\mathcal{R}$ is the refuter algorithm specified in Theorem 3.3.1. It follows that $M^{\mathsf{PCPP}}(x_{\mathsf{hard}})\neq L^{\mathsf{hard}}(x_{\mathsf{hard}})$. The only possibility is that $x_{\mathsf{hard}}\in L^{\mathsf{hard}}$ but does not have an easy witness. Let Π be the PCPP proof for $x_{\mathsf{hard}}\in L^{\mathsf{hard}}$ (which can be computed in $\mathsf{TIME}[\mathrm{poly}(N)]^{\mathsf{NP}}$). Then there exists a row $i\in[\hat{H}_{\mathsf{proof}}]$ such that, denoting π_i as the i-th row of Π, then $\pi_i:\{0,1\}^n\to\{0,1\}$ is a Boolean function that has ℓ_1 distance at least δ with every $\mathsf{Sum}\circ\mathscr{C}$ circuit of size at most $s(n)$. Whether a truth table possesses such hardness can be decided in $\mathsf{TIME}[2^{O(n)}]^{\mathsf{NP}}$, hence we can find such a truth table in $\mathsf{TIME}[2^{O(n)}]^{\mathsf{NP}}$. This proves the desired lower bound for E^{NP}. □

As a consequence, non-trivial CAPP algorithms with E^{NP} preprocessing also imply that E^{NP} is strongly average-case hard against $\mathscr{C}$ circuits.

Theorem 4.4.2. *Let $\mathscr{C}$ be a circuit class satisfying the technical conditions in Theorem 4.4.1, and $d\geq 2$ be the absolute constant in Theorem 4.4.1. For every constant $\varepsilon>0$, there exists a constant $\varepsilon'>0$ such that: If there is a CAPP data structure for $\mathsf{AND}_d\circ\mathscr{C}$ circuits of size 2^{n^ε} and n inputs in 2^{n-n^ε} query time with E^{NP} preprocessing and inverse-circuit-size error, then E^{NP} cannot be $(1/2+2^{-n^{\varepsilon'}})$-approximated by $\mathscr{C}$ circuits of $2^{n^{\varepsilon'}}$ size on almost every input length.*

Proof. This follows from Theorem 4.4.1 and Theorem 4.2.9. In fact, it follows from Theorem 4.4.1 that there is a constant $\varepsilon''>0$ and a language $L\in\mathsf{E}^{\mathsf{NP}}$ that has ℓ_1-distance at least δ from

$[0,1]$-$\mathsf{Sum} \circ \mathscr{C}$ circuits of complexity $2^{n^{\varepsilon''}}$ on almost every input length, where $\delta > 0$ is the absolute constant in Theorem 4.4.1. Let $\varepsilon' := \varepsilon''/2$, $k(n) := O(n^{\varepsilon'})$, then by Theorem 4.2.9, the language $L^{\oplus k(n)} \in \mathsf{E}^{\mathsf{NP}}$ cannot be $(1/2 + 2^{-n^{\varepsilon'}})$-approximated by $\mathscr{C}$ circuits of size $2^{n^{\varepsilon'}}$. □

4.5 Applications

We present two applications of the result that non-trivial circuit-analysis problems imply circuit lower bounds, *even with* E^{NP} *preprocessing*.

4.5.1 Shaving Logs Implies Lower Bounds, Even with Preprocessing

As a direct corollary of Theorem 4.3.2, we tighten the connections between circuit lower bounds and non-trivial speed-ups for certain problems in fine-grained complexity. Typically, such a connection states that certain algorithms that are *slightly better than brute force* implies breakthrough circuit lower bounds: For certain well-studied problems $\mathcal{L}$ where an $\tilde{O}(n^2)$-time algorithm is already known, if we could solve $\mathcal{L}$ in $n^2/\log^{\omega(1)} n$ time (i.e., "shaving all logs"), then breakthrough circuit lower bounds follow.

Theorem 4.3.2 implies that we could still obtain the breakthrough lower bounds even if we allow a *preprocessing phase of time* n^{100} *with an* NP *oracle* before we receive the input of $\mathcal{L}$. For an optimist, this can be seen as an improved approach to prove such lower bounds: Now, we can rely on the power of P^{NP} preprocessing to solve $\mathcal{L}$, and we (still) only need to obtain a modest improvement over the naïve algorithms.

There are many such connections in the literature, but for illustration, we only consider the following examples:

- LCS (Longest Common Subsequence) over alphabet Σ: Given two sequences $a, b \in \Sigma^N$, find the length of their longest common subsequence.

  Abboud and Bringmann [AB18], improving on [AHWW16], showed that the SAT problem for formulas of size s and n inputs can be reduced to an instance of LCS with two sequences of length $N := 2^{n/2} \cdot s^{1+O(1/\log\log\sigma)}$ over alphabet $[\sigma]$ in $O(N)$ time.

- $\mathsf{Closest\text{-}LCS\text{-}Pair}$: Given two sets of N length-D strings $\mathcal{A}, \mathcal{B}$, find (or approximate) the maximum length of the longest common subsequence among all pairs $(a, b) \in \mathcal{A} \times \mathcal{B}$.

  Chen, Goldwasser, Lyu, Rothblum, and Rubinstein [CGL+19] showed that for every constant $c \geq 1$, the SAT problem for formulas of size s and n inputs can be reduced to an instance of c-approximate $\mathsf{Closest\text{-}LCS\text{-}Pair}$ with $N := 2^{n/2}$ and $D := 2^{\mathrm{polylog}(n)}$.

  Proof Sketch in [CGL+19]. We use Barrington's theorem [Bar89] to transform the formula into a width-5 branching program of size $\mathrm{poly}(s)$. Then we reduce its SAT problem to the following problem (called $\mathsf{BP\text{-}Satisfying\text{-}Pair}$ in [CGL+19]): Given a size-$\mathrm{poly}(s)$ width-5 branching program P on n Boolean inputs, and a set of $N := 2^{n/2}$ strings $\mathcal{A}, \mathcal{B} \subseteq \{0,1\}^n$, determine if there is a pair $(a, b) \in \mathcal{A} \times \mathcal{B}$ such that $P(a, b) = 1$. The next step is to invoke [CGL+19, Theorem 5.6] to reduce $\mathsf{BP\text{-}Satisfying\text{-}Pair}$ to a problem called ε-$\mathsf{Gap\text{-}Max\text{-}TropSim}$, whose input

consists of two sets of N tensors of size $D := 2^{O(\log^2 n \log\log n)}$. Finally, this problem reduces to $O(1)$-approximate Closest-LCS-Pair over N length-D strings. □

- $\mathbb{Z}$-OV (Hopcroft's Problem): Given N points $\vec{v}^1, \vec{v}^2, \ldots, \vec{v}^N \in \mathbb{Z}^D$, find an orthogonal pair, i.e., two vectors $\vec{v}^a$ and $\vec{v}^b$ such that

$$\sum_{i=1}^{D} v_i^a \cdot v_i^b = 0.$$

Chen [Che18] showed that the SAT problem on THR∘THR circuits of size s and n inputs can be reduced to poly(s) $\mathbb{Z}$-OV instances on $N := 2^{n/2}$ vectors of dimension $D :=$ poly(s).

Proof Sketch in [Che18]. Given a THR ∘ THR circuit of size s, we use [Che18, Theorem 1.6] to transform it into an OR∘THR∘MAJ circuit of size poly(s), then use [HP10] to transform it into an OR∘ETHR∘MAJ circuit of size poly(s). Here, ETHR denotes "exact threshold gates", which outputs 1 if a certain linear combination of its input is exactly equal to its threshold parameter, and outputs 0 otherwise; the transformation can be performed in deterministic poly(s) time. Finally, we reduce the satisfiability of each bottom ETHR ∘ MAJ circuit to $\mathbb{Z}$-OV. □

The naïve algorithms for LCS, Closest-LCS-Pair, and $\mathbb{Z}$-OV run in $O(N^2)$, $O(N^2D^2)$, and $O(N^2D)$ time respectively. The above reductions show that a modest improvement of these quadratic-time algorithms (i.e., "shaving all logs") would imply new SAT algorithms for frontier circuit classes, e.g., NC^1 or THR ∘ THR. By the Algorithmic Method [Wil13a, CW19b], these SAT algorithms imply long-standing circuit lower bounds for these classes.

To state our corollary, we need the following definition of solving a problem with P^{NP} preprocessing.

Definition 4.5.1. We say that a problem $\mathcal{L}$ can be solved in $T(n)$ time *with* P^{NP} *preprocessing* if there are two algorithms (Prep, Query) such that the following holds:

- Prep receives an input 1^n, runs in time poly(n) with access to an NP oracle, and outputs a string DS of length poly(n).
- Query receives an input x of $\mathcal{L}$, has random access to DS, runs in time $T(n)$, and correctly decides whether $x \in \mathcal{L}$.

Now we present the following corollary of Theorem 4.3.2, which states that even if we allow a P^{NP} preprocessing phase (which runs in arbitrary polynomial time in N but does not see the input), such a modest improvement would still imply breakthrough circuit lower bounds:

Corollary 4.5.2. *The following are true:*

- *Suppose* LCS *of length-N strings over any $O(1)$-size alphabet can be solved in $N^2/\log^{\omega(1)} N$ time, even with P^{NP} preprocessing, then $\mathsf{E}^{\mathsf{NP}} \not\subseteq \mathsf{NC}^1$.*
- *Suppose there is a constant $c \geq 1$ such that for any $D = 2^{\mathrm{poly}(\log\log N)}$,* Closest-LCS-Pair *of N length-D strings can be c-approximated in $N^2/\log^{\omega(1)} N$ time, even with P^{NP} preprocessing, then $\mathsf{E}^{\mathsf{NP}} \not\subseteq \mathsf{NC}^1$.*

- *Suppose for any $D = \mathrm{polylog}(N)$, $\mathbb{Z}$-OV for N points in $\mathbb{Z}^D$ can be solved in $N^2/\log^{\omega(1)} N$ time, even with P^{NP} preprocessing, then $\mathsf{E}^{\mathsf{NP}} \not\subseteq \mathsf{THR} \circ \mathsf{THR}$.*

Remark 4.5.3. Since NC^1 satisfies all technical conditions in Theorem 4.3.2 (NC^1 is complete, computes PARITY efficiently, and is closed under adding NC^0 circuits at the top), the first two items of Corollary 4.5.2 are straightforward. For $\mathsf{THR} \circ \mathsf{THR}$:

- Since $\mathsf{THR} \circ \mathsf{THR}$ contains CNF (i.e., $\mathsf{AND} \circ \mathsf{OR}$), it is clearly complete.
- There is a polynomial-size $\mathsf{THR} \circ \mathsf{THR}$ circuit that computes PARITY [Mur71, PS94].
- Finally, for $d = O(1)$, the SAT problem (thus, the GapUNSAT problem) for $\mathsf{NC}^0_d \circ \mathsf{THR} \circ \mathsf{THR}$ reduces to $\mathrm{poly}(n^d)$ instances of the SAT problem for $\mathsf{THR} \circ \mathsf{THR}$.

  More precisely, since $\mathsf{THR} \circ \mathsf{THR}$ is closed under negation, the SAT problem for $\mathsf{NC}^0_d \circ \mathsf{THR} \circ \mathsf{THR}$ reduces to the SAT problem for $2^{O(d)}$ instances of $\mathsf{AND}_d \circ \mathsf{THR} \circ \mathsf{THR}$. Using the fact that $\mathsf{THR} \subseteq \mathsf{OR} \circ \mathsf{ETHR}$ [HP10], we reduce the problem to the SAT problem for $\mathsf{AND}_d \circ \mathsf{OR} \circ \mathsf{ETHR} \circ \mathsf{THR}$. Enumerating the set of d bottom $\mathsf{ETHR} \circ \mathsf{THR}$ circuits that are satisfied, we reduce the problem to $\mathrm{poly}(n)^d \le \mathrm{poly}(n)$ instances of the SAT problem for $\mathsf{AND}_d \circ \mathsf{ETHR} \circ \mathsf{THR}$ circuits. Since $\mathsf{AND} \circ \mathsf{ETHR} \subseteq \mathsf{ETHR}$ and $\mathsf{ETHR} \circ \mathsf{THR} = \mathsf{ETHR} \circ \mathsf{ETHR} \subseteq \mathsf{THR} \circ \mathsf{THR}$ [HP10], every $\mathsf{AND}_d \circ \mathsf{ETHR} \circ \mathsf{THR}$ circuit is equivalent to a $\mathsf{THR} \circ \mathsf{THR}$ circuit. Finally, notice that every transformation mentioned above can be implemented in deterministic polynomial time.

4.5.2 The Complexity of Adleman's Argument

Theorem 4.3.2 has an interesting application to the complexity of the following problem, denoted as Adleman. Adleman [Adl78] showed that $\mathsf{BPP} \subseteq \mathsf{P}/_{\mathrm{poly}}$; in particular, for every $n, s \in \mathbb{N}$, there exists a non-uniform algorithm[2] of time complexity $\mathrm{poly}(n, s)$ that solves CAPP for n-input size-s circuits. What is the complexity of *finding* such a non-uniform algorithm?

Definition 4.5.4. Let $n \le s_1(n) \le s_2(n)$, $\textsc{Adleman}_{s_1,s_2}$ is the following (unary) total search problem: Given 1^n, $1^{s_1(n)}$, and $1^{s_2(n)}$, the goal is to output the description of a non-uniform algorithm $\mathcal{A}$ of size at most $s_2(n)$ that solves CAPP for n-input size-$s_1(n)$ circuits. Namely, for every size-$s_1(n)$ circuit $C : \{0,1\}^n \to \{0,1\}$, $\mathcal{A}(C)$ outputs a rational number that is 1/6-close to $\Pr_{x \leftarrow \{0,1\}^n}[C(x) = 1]$.

Note that we are also interested in the case when $s_2(n)$ is super-polynomial in n and we want to find a non-uniform algorithm for CAPP in $\mathrm{poly}(s_2(n))$ time. Hence in the above definition, besides 1^n, we also provide $1^{s_1(n)}$ and $1^{s_2(n)}$ as inputs of $\textsc{Adleman}_{s_1,s_2}$.

Adleman's argument [Adl78] shows that $\textsc{Adleman}_{s_1,s_2}$ is a *total* search problem as long as $s_2(n) \ge s_1(n)^c$ for some absolute constant c. In fact, one can also show that $\textsc{Adleman}_{s_1,s_2}$ is in APEPP (i.e., $\textsc{Adleman}_{s_1,s_2}$ reduces to Avoid):[3]

[2] Non-uniform circuits and non-uniform algorithms are equivalent computational models. However, to avoid confusion, in this subsection we use "non-uniform algorithms" to refer to non-uniform algorithms solving CAPP and use "circuits" to refer to inputs of CAPP.

[3] A subtlety is that it is unclear whether Adleman is in $\mathsf{TF\Sigma_2P}$, i.e., whether it is possible to verify in P^{NP} that a non-uniform algorithm $\mathcal{A}$ indeed solves CAPP. The class APEPP, as defined in [KKMP21], consists of *all total functions* that are polynomial-time reducible to Avoid (there is no requirement that the total function itself should be in $\mathsf{TF\Sigma_2P}$). Hence it makes sense to say that Adleman is in APEPP or even APEPP-complete without knowing whether Adleman is in $\mathsf{TF\Sigma_2P}$.

Proposition 4.5.5. *Let $s_2(n) \geq s_1(n)^7$. Then there is a polynomial-time reduction from* $\textsc{Adleman}_{s_1,s_2}$ *to* Avoid.

Proof. Let PRG be the following problem: Given 1^N, output a sequence $(x_1, x_2, \ldots, x_{N^6})$ such that for every N-input circuit C of size N,

$$\left| \Pr_{i \leftarrow [N^6]}[C(x_i) = 1] - \Pr_{y \leftarrow \{0,1\}^N}[C(y) = 1] \right| \leq 1/6.$$

Korten [Kor21] showed that PRG reduces to Avoid. It is easy to see that $\textsc{Adleman}_{s_1,s_2}$ reduces to PRG: Let $(x_1, \ldots, x_{s_1(n)^6})$ be a valid output of PRG on input $1^{s_1(n)}$. one can construct a non-uniform algorithm $\mathcal{A}$ that given a size-$s_1(n)$ circuit C as an input, outputs the fraction of indices $i \in [s_1(n)^6]$ such that $C(x_i) = 1$. This algorithm is a valid output of $\textsc{Adleman}_{s_1,s_2}$. □

An interesting corollary of Theorem 4.3.2 is that Adleman is, in fact, APEPP-complete under P^{NP} reductions! Moreover, this is true even in a restricted parameter setting, where $s_1(n) = 2^{\delta n}$ and $s_2(n) = 2^{(1-\delta)n}$ for some constant $\delta > 0$.

Theorem 4.5.6. *Let $\delta > 0$ be a constant, $s_1(n) = 2^{\delta n}$ and $s_2(n) = 2^{(1-\delta)n}$, then* $\textsc{Adleman}_{s_1,s_2}$ *is* APEPP*-complete under* P^{NP} *reductions.*

Proof Sketch. Let $\varepsilon > 0$, ε-Hard be the following problem: Given 1^N where $N = 2^n$, output the truth table of a Boolean function $f : \{0,1\}^n \to \{0,1\}$ that has circuit complexity at least $2^{\varepsilon n}$. Korten [Kor21] showed that for every constant $\varepsilon > 0$, ε-Hard is APEPP-hard under P^{NP} reductions. Therefore, it suffices to design a P^{NP} reduction from ε-Hard to $\textsc{Adleman}_{2^{\delta n},2^{(1-\delta)n}}$. This is essentially *circuit lower bounds from circuit-analysis algorithms*!

Let $\varepsilon \in (0, 0.01)$ be a constant such that, via Theorem 4.3.2, a GapUNSAT data structure for size-$2^{\delta n}$ circuits implies a circuit lower bound of size $s(n) := 2^{\varepsilon n}$. Let $\mathcal{A}$ be a solution to $\textsc{Adleman}_{2^{\delta n},2^{(1-\delta)n}}$. Then $\mathcal{A}$ is a non-uniform algorithm that runs in $2^{(1-\delta)n}$ time and solves CAPP on circuits $C : \{0,1\}^n \to \{0,1\}$ of size $s(n) := 2^{\delta n}$. We follow the same proof of Theorem 4.3.2 with $\mathscr{C}$ being the class of (general) circuits; the only difference is that M^{PCPP} uses $\mathcal{A}$ to estimate the accept probability of the PCPP verifier. The running time of M^{PCPP} and the number of nondeterministic bits guessed by M^{PCPP} remain unchanged; M^{PCPP} takes the description of $\mathcal{A}$ as advice, so its advice complexity is at most 2^n (i.e., $c = 1$ in the proof of Theorem 4.3.2).

Our P^{NP} reduction from ε-Hard to $\textsc{Adleman}_{2^{\delta n},2^{(1-\delta)n}}$ works as follows. We first compute the description of M^{PCPP} from the description of $\mathcal{A}$ in polynomial time. Then we feed M^{PCPP} to the P^{NP} refuter in Theorem 3.3.1 to obtain an input x_{hard} such that $L^{\mathsf{hard}}(x_{\mathsf{hard}}) \neq M^{\mathsf{PCPP}}(x_{\mathsf{hard}})$. It has to be the case that $x_{\mathsf{hard}} \in L^{\mathsf{hard}}$, but any PCPP proof for x_{hard} (treated as a matrix) contains a row whose circuit complexity is at least $2^{\varepsilon n}$. We can find such a row in deterministic $\mathrm{poly}(2^n)$ time with an NP oracle, and this gives us a solution for ε-Hard. □

4.6 Equivalences between Circuit Lower Bounds and Derandomisation with Preprocessing

We show that circuit lower bounds for E^{NP} are actually *equivalent* to derandomisation with E^{NP} preprocessing.

Our first theorem shows that if $\mathscr{C}$ is a "strong enough" circuit class, then the worst-case lower bound $\mathsf{E}^{\mathsf{NP}} \not\subseteq \mathscr{C}$ is equivalent to non-trivial derandomisation of $\mathscr{C}$ circuits with E^{NP} preprocessing. For simplicity, we only consider a few typical circuit classes (TC^0, NC^1, and $\mathsf{P}/_{\mathrm{poly}}$), but it is clear from the argument that we only rely on a few closure properties of $\mathscr{C}$.

Theorem 4.6.1. *Let $\mathscr{C} \in \{\mathsf{TC}^0, \mathsf{NC}^1, \mathsf{P}/_{\mathrm{poly}}\}$. The following are equivalent:*

1. *For every constant $k \geq 1$, there is a language $L \in \mathsf{E}^{\mathsf{NP}}$ that cannot be $(1/2 + 1/n^k)$-approximated by* i.o.-$\mathscr{C}[n^k]$.
2. *For every constant $k \geq 1$, there is a language $L \in \mathsf{E}^{\mathsf{NP}}$ such that $L \notin$ i.o.-$\mathscr{C}[n^k]$.*
3. *For every constant $k \geq 1$, there is a $\mathsf{GapUNSAT}$ algorithm for $\mathscr{C}[n^k]$ with E^{NP} preprocessing, error $1 - 0.01$, and $2^n/n^{\omega(1)}$ query time.*
4. *For every constant $\delta > 0$ and every good function $s = s(n)$, there is a CAPP algorithm for $\mathscr{C}[s]$ circuits with $\mathsf{TIME}[\exp(s^\delta)]^{\mathsf{NP}}$ preprocessing, $\exp(s^\delta)$ query time, and inverse-circuit-size error.*
5. *For every constant $k \geq 1$, there is an E^{NP}-computable PRG with seed length $n - 1$ that $(1/3)$-fools $\mathscr{C}[n^k]$ circuits.*
6. *For every constant $k \geq 1$, there is an E^{NP}-computable PRG with seed length $n^{1/k}$ that $(1/n^k)$-fools $\mathscr{C}[n^k]$ circuits.*

Proof. (4) $\implies$ (3) and (6) $\implies$ (5) are trivial.

(3) $\implies$ (2) follows from Theorem 4.3.2, Remark 4.3.3, and the fact that $\mathscr{C}$ is "typical". In particular, $\mathscr{C}$ is complete (any Boolean function can be computed by $\mathscr{C}$ circuits of large enough size), PARITY has polynomial-size $\mathscr{C}$ circuits, and $\mathscr{C}$ is closed under composition of an NC^0 circuit at the top.

(2) $\implies$ (1) follows from the locally list-decodable code of [GR08] with TC^0 decoders. In particular, let $\varepsilon := 1/n^{k+1}$, and $(\mathsf{Amp}, \mathsf{Dec})$ be the locally list-decodable code specified in Theorem 4.2.8. Let $f : \{0,1\}^n \to \{0,1\}$ be a hard function in $\mathsf{E}^{\mathsf{NP}} \setminus$ i.o.-$\mathscr{C}[n^{O(k)}]$, and $tt_f \in \{0,1\}^{2^n}$ be its truth table. Let $tt_{f'} := \mathsf{Amp}(tt_f)$, then for some constant $d > 1$, $tt_{f'}$ is the truth table of a function $f' : \{0,1\}^{dn} \to \{0,1\}$ which is computable in E^{NP}.

We claim that f' cannot be $(1/2 + 1/n^{k+1})$-approximated by $\mathscr{C}$ circuits of size n^{k+1} (thus cannot be $(1/2 + 1/(dn)^k)$-approximated by $\mathscr{C}$ circuits of $(dn)^k$). Suppose for contradiction that there is a circuit $C \in \mathscr{C}[n^{k+1}]$ that $(1/2 + 1/n^{k+1})$-approximates f'. Fix a good r_1, then there is a string $\alpha \in \{0,1\}^{O(\log \varepsilon^{-1})}$ such that for every $x \in \{0,1\}^n$,

$$\Pr_{r_2}[\mathsf{Dec}^C(\alpha, x, r_1, r_2) = f(x)] > 9/10.$$

By Adleman's argument [Adl78] over r_2, and recall that $\mathsf{Dec}^{(-)}$ is a TC^0 oracle circuit, it follows that f can be computed by a $\mathscr{C}$ circuit of size $n^{O(k)}$. This contradicts the worst-case hardness of f.

(1) $\implies$ (6) follows from the Nisan–Wigderson generator (Theorem 4.2.1). In particular, we set the following parameters:

$$\ell := n^{1/3k}, m := n, a := \log n, t := O(\ell^2 \cdot m^{1/a}/a) := O(\ell^2) \leq n^{1/k}.$$

Let $f : \{0,1\}^\ell \to \{0,1\}$ be a function in E^{NP} that cannot be $(1/2 + 1/n^{k+1})$-approximated by $\mathscr{C}$ circuits of size $n^k \cdot \mathrm{poly}(2^a) \leq \mathrm{poly}(n)$. Then it cannot be $(1/2 + 1/n^{k+1})$-approximated by $\mathscr{C} \circ \mathsf{Junta}_a$ circuits where the top $\mathscr{C}$ circuit has size n^k. By Theorem 4.2.1, there is a function $G : \{0,1\}^{2^\ell} \times \{0,1\}^t \to \{0,1\}^m$ computable in $\mathrm{poly}(m, 2^t) \leq 2^{O(t)}$ time, such that $G(tt(f), \mathcal{U}_t)$ $(1/n^k)$-fools every $\mathscr{C}[n^k]$ circuit. Since $f \in \mathsf{E}^{\mathsf{NP}}$, the generator $G(tt(f), \cdot)$ is computable in E^{NP}.

(6) $\implies$ (4): Let $G : \{0,1\}^{s^{\delta/2}} \to \{0,1\}^s$ be the E^{NP}-computable PRG that $(1/s)$-fools $\mathscr{C}[2s]$ circuits with s inputs. (Given a $\mathscr{C}[s]$ circuit C with n inputs, we can pad some dummy inputs to C so that C becomes a $\mathscr{C}$ circuit of size $2s$ with s inputs; C only depends on the first n inputs.)

In the E^{NP} preprocessing phase, we compute the whole PRG (i.e., $G(x)$ for every $x \in \{0,1\}^{s^{\delta/2}}$). Given a circuit $C \in \mathscr{C}[s]$ as a CAPP query, we simply compute

$$\Pr_{x \leftarrow \{0,1\}^{s^{\delta/2}}}[C(G(x)) = 1],$$

which estimates the accept probability of C within additive error $1/s$. The query algorithm runs in $\mathrm{poly}(s) \cdot \exp(s^{\delta/2}) < \exp(s^\delta)$ time.

(5) $\implies$ (2) follows from Fact 4.2.7. In particular, suppose there is an E^{NP}-computable PRG $G : \{0,1\}^{n-1} \to \{0,1\}^n$ that $(1/3)$-fools $\mathscr{C}[n^k]$ circuits. Since $1/3 < 2^{(n-1)-n} = 1/2$, it follows from Fact 4.2.7 that E^{NP} cannot be computed by $\mathscr{C}[n^k]$ circuits on almost every input length. □

Our equivalence also holds in the high-end regime (e.g., for subexponential-size circuit lower bounds) and in the infinitely-often setting:

Theorem 4.6.2. *Let $\mathscr{C} \in \{\mathsf{TC}^0, \mathsf{NC}^1, \mathsf{P}/_{\mathrm{poly}}\}$. The following are equivalent:*

1. *There is a constant $\varepsilon > 0$ and a language $L \in \mathsf{E}^{\mathsf{NP}}$ such that $L \notin \mathscr{C}[2^{n^\varepsilon}]$.*
2. *There is a constant $\varepsilon > 0$ and a language $L \in \mathsf{E}^{\mathsf{NP}}$ such that L cannot be $(1/2 + 1/2^{n^\varepsilon})$-approximated by $\mathscr{C}[2^{n^\varepsilon}]$.*
3. *There is a constant $\varepsilon > 0$ and an i.o. $\mathsf{GapUNSAT}$ algorithm for $\mathscr{C}[2^{n^\varepsilon}]$ with E^{NP} preprocessing, error $1 - 0.01$, and $2^n/n^{\omega(1)}$ query time.*
4. *There is a constant $k \geq 1$ such that for every good function $s = s(n)$, there is an i.o. CAPP algorithm for $\mathscr{C}[s]$ with $\mathsf{TIME}[2^{\log^k s}]^{\mathsf{NP}}$ preprocessing, $2^{\log^k s}$ query time, and inverse-circuit-size error.*

5. *There is a constant $c \geq 1$ and an E^{NP}-computable i.o. PRG with seed length $\log^c n$ that $(1/n)$-fools $\mathscr{C}$ circuits of size n.*

Proof Sketch. (3) $\implies$ (1) follows from Corollary 4.3.4.

(1) $\implies$ (2) follows from Theorem 4.2.8.

(2) $\implies$ (5) follows from Theorem 4.2.1.

(5) $\implies$ (4) follows by simply applying the i.o. PRG to fool the input circuit.

(4) $\implies$ (3) is trivial. □

For weaker circuit classes, we also get an equivalence by considering *strong average-case* lower bounds: hard functions that cannot be $(1/2+1/\mathrm{poly}(n))$-approximated by $\mathscr{C}$, non-trivial CAPP data structures for $\mathscr{C}$ with inverse-circuit-size error, and PRGs fooling $\mathscr{C}$ with very small error are all equivalent. Note that the following equivalence only holds in the low-end regime (i.e., for polynomial size but not for subexponential size) and only holds for infinitely-often lower bounds. The reason is that we need to use a win-win argument in [CR22, CLLO21]: if a certain NC^1-hard problem (called DCMD in [CR22]) can be approximated by $\mathscr{C}$ circuits, we proceed in one way; if not, we proceed in another way. The reader is referred to the discussion in [CLW20, Section 2.2.2] for more details on the limitation of this win-win argument.

Below we list the properties of the weak circuit class $\mathscr{C}$ that we need:

($\mathscr{C}$ is typical) $\mathscr{C}$ is closed under negation and projection.

($\mathscr{C}$ is complete) For every truth table of length 2^k, there is a $\mathscr{C}$ circuit of size $\mathrm{poly}(2^k)$ that computes this truth table; moreover, the description of such a $\mathscr{C}$ circuit can be computed in deterministic $\mathrm{poly}(2^k)$ time from the truth table.

($\mathscr{C}$ computes PARITY) The PARITY function can be computed by a $\mathscr{C}$ circuit of size $\mathrm{poly}(n)$.

($\mathscr{C}$ is closed under bottom juntas) For every constant $d \geq 1$, every $\mathscr{C} \circ \mathsf{Junta}_d$ circuit can be computed by a polynomially-larger $\mathscr{C}$ circuit.

($\mathscr{C}$ is closed under top NC^0 circuits) For every constant $d \geq 1$, every $\mathsf{NC}^0_d \circ \mathscr{C}$ circuit can be computed by a polynomially-larger $\mathscr{C}$ circuit.

Theorem 4.6.3. *Let $\mathscr{C}$ be a circuit class that satisfies the properties above. If $\mathscr{C} \subseteq \mathsf{NC}^1$, then the following are equivalent:*

1. *For every constant $k \geq 1$, there is a language $L \in \mathsf{E}^{\mathsf{NP}}$ that cannot be $\left(1/2+1/n^k\right)$-approximated by $\mathscr{C}[n^k]$.*
2. *For every constant $k \geq 1$, there is a language $L \in \mathsf{E}^{\mathsf{NP}}$ such that $L \notin \mathsf{MAJ} \circ \mathscr{C}[n^k]$.*
3. *There is $\delta \geq 1/\mathrm{poly}(n)$ such that for every constant $k \geq 1$, there is a language $L \in \mathsf{E}^{\mathsf{NP}}$ such that $L \notin \widetilde{\mathsf{Sum}}_\delta \circ \mathscr{C}[n^k]$.*
4. *For every constant $k \geq 1$, there is a language $L \in \mathsf{E}^{\mathsf{NP}}$ such that $L \notin \widetilde{\mathsf{Sum}}_{1/3} \circ \mathscr{C}[n^k]$.*
5. *For every constant $k \geq 1$, there is a language $L \in \mathsf{E}^{\mathsf{NP}}$ such that L has ℓ_1-distance at least δ from any $[0,1]$-$\mathsf{Sum} \circ \mathscr{C}$ circuit of complexity n^k, where $\delta > 0$ is the absolute constant in Theorem 4.4.1.*

6. *For every constant $k \geq 1$, there is an infinitely-often* CAPP *algorithm for $\mathscr{C}[n^k]$ with E^{NP} preprocessing, $2^n/n^{\omega(1)}$ query time, and inverse-circuit-size error.*

7. *For every constant $\delta > 0$ and every good function $s = s(n)$, there is an infinitely-often* CAPP *algorithm for $\mathscr{C}[s]$ circuits with $\mathsf{TIME}[\exp(s^\delta)]^{\mathsf{NP}}$ preprocessing, $\exp(s^\delta)$ query time, and inverse-circuit-size error.*

8. *For every constant $k \geq 1$, there is an E^{NP}-computable i.o. PRG with seed length $n-1$ that $(1/n^k)$-fools $\mathscr{C}[n^k]$ circuits.*

9. *For every constant $k \geq 1$, there is an E^{NP}-computable i.o. PRG with seed length $n^{1/k}$ that $(1/n^k)$-fools $\mathscr{C}[n^k]$ circuits.*

Proof. (9) $\implies$ (8), (7) $\implies$ (6), and (4) $\implies$ (3) are trivial. (2) $\implies$ (4) follows from the fact that $\bigcup_{k\in\mathbb{N}} \widetilde{\mathsf{Sum}}_{1/3} \circ \mathscr{C}[n^k] \in \bigcup_{k\in\mathbb{N}} \mathsf{MAJ} \circ \mathscr{C}[n^k]$.

(5) $\implies$ (3) is because every language in $\widetilde{\mathsf{Sum}}_\delta \circ \mathscr{C}$ is also close to a $[0,1]$-$\mathsf{Sum} \circ \mathscr{C}$ circuit of similar complexity. In particular, let $f : \{0,1\}^n \to \{0,1\}$ be computed by a $\widetilde{\mathsf{Sum}}_\delta \circ \mathscr{C}$ circuit $C(x) := \sum_{i=1}^{\ell} \alpha_i C_i(x)$ of complexity s. Consider the circuit $C'(x) := (C(x) + \delta)/(1 + 2\delta)$, then C' is a $[0,1]$-$\mathsf{Sum} \circ \mathscr{C}$ circuit of complexity $\mathrm{poly}(s)$, and

$$\begin{aligned} \|C' - f\|_1 &= \mathop{\mathbb{E}}_{x \leftarrow \{0,1\}^n} [|C'(x) - f(x)|] \\ &\leq \mathop{\mathbb{E}}_{x \leftarrow \{0,1\}^n} \left[\left| \frac{C(x) - f(x) + \delta}{1 + 2\delta} \right| \right] + 2\delta \leq 4\delta. \end{aligned}$$

Hence, if $L \in \mathsf{E}^{\mathsf{NP}}$ has ℓ_1-distance at least δ from any $[0,1]$-$\mathsf{Sum} \circ \mathscr{C}$ circuit of complexity n^k, then $L \notin \widetilde{\mathsf{Sum}}_{\delta/4} \circ \mathscr{C}[n^{k-1}]$.

(9) $\implies$ (7): The proof is exactly the same as (6) $\implies$ (4) in Theorem 4.6.1. That is, we apply the PRG to solve CAPP.

(8) $\implies$ (3): From Lemma 4.2.4, a PRG fooling $\mathscr{C}$ is also a PRG fooling $\widetilde{\mathsf{Sum}} \circ \mathscr{C}$. Then, from Fact 4.2.7, a PRG fooling $\widetilde{\mathsf{Sum}} \circ \mathscr{C}$ implies a lower bound for it. Details follow.

For any fixed constant k, let $G_n : \{0,1\}^{n-1} \to \{0,1\}^n$ be an E^{NP}-computable i.o. PRG that $(1/n^{k+1})$-fools $\mathscr{C}[n^{k+1}]$ circuits. Then, by Lemma 4.2.4, G is also an i.o. PRG that ε'-fools $\widetilde{\mathsf{Sum}}_{1/n} \circ \mathscr{C}$ circuits of complexity n^k, where $\varepsilon' := 2 \cdot (1/n) + n^k/n^{k+1} \leq 3/n$. By Fact 4.2.7, there is a language in E^{NP} that is not computable in $\widetilde{\mathsf{Sum}}_{1/n} \circ \mathscr{C}[n^k]$.

(6) $\implies$ (5) follows from Theorem 4.4.1 and the hypotheses that $\mathscr{C}$ is typical, complete, computes PARITY, and is closed under top NC^0 circuits.

(3) $\implies$ (1) follows from the proof of (2) $\implies$ (6) in [CLLO21, Theorem 1]; for completeness we provide a sketch here. If a certain problem called DCMD, which is in P, cannot be $(1/2 + 1/n^k)$-approximated by $\mathscr{C}[n^k]$ for every constant k, then (1) follows directly. Otherwise, by [CR22, Lemma 3.1], $\mathsf{NC}^1 \subseteq \widetilde{\mathsf{Sum}}_\delta \circ \mathscr{C}$. Let L be a language in E^{NP} that does not have $\widetilde{\mathsf{Sum}}_\delta \circ \mathscr{C}[n^{O(k)}]$ circuits, then L does not have NC^1 circuits of size $n^{O(k)}$. By standard hardness amplification (with NC^1 decoders) such as Theorem 4.2.8, it follows that L cannot be $(1/2 + 1/n^k)$-approximated by NC^1 circuits of size $n^{O(k)}$. Since $\mathscr{C} \subseteq \mathsf{NC}^1$, (1) is true.

(1) $\implies$ (2) follows from the discriminator lemma [HMP+93]. For any function $f \in \mathsf{MAJ} \circ \mathscr{C}$ where the top MAJ gate has fan-in s, f can be $(1/2 + 1/O(s))$ approximated by $\mathscr{C}$. This implies

the contrapositive of (1) $\implies$ (2).

(1) $\implies$ (9) follows from the Nisan–Wigderson generator [NW94]. We use the following parameters in Theorem 4.2.1:

$$\ell := n^{1/3k}, m := n, a := 4k, t := O(\ell^2 \cdot m^{1/a}/a) \leq n^{1/k}.$$

We use a hard truth table of length 2^ℓ that is not $(1/2 + 1/n^{k+1})$-approximated by $\mathscr{C} \circ \mathsf{Junta}_a$ circuits, where the top $\mathscr{C}$ circuits have size n^k. As any $\mathscr{C} \circ \mathsf{Junta}_a$ can be computed by a polynomially larger $\mathscr{C}$, the same proof applies. □

We also present a characterisation of E^{NP} lower bounds against weak circuit classes that holds both in the high-end regime and almost everywhere. Note that it is unclear whether this equivalence extends to lower bounds against $\widetilde{\mathsf{Sum}} \circ \mathscr{C}$ circuits.

Theorem 4.6.4. *Let $\mathscr{C}$ be a circuit class satisfying the above properties. Moreover, suppose the property that $\mathscr{C}$ is closed under bottom juntas is strengthened to:*

($\mathscr{C}$ is closed under bottom juntas) *Every $\mathscr{C} \circ \mathsf{Junta}_{\log n}$ circuit can be computed by a polynomially large $\mathscr{C}$ circuit.*

Then the following are equivalent:

1. *There is a constant $\varepsilon > 0$ and a language $L \in \mathsf{E}^{\mathsf{NP}}$ such that L cannot be $(1/2 + 1/2^{n^\varepsilon})$-approximated by $\mathscr{C}[2^{n^\varepsilon}]$ circuits on almost every input length.*
2. *There is a constant $\varepsilon > 0$ and a language $L \in \mathsf{E}^{\mathsf{NP}}$ such that L cannot be approximated by $[0,1]\text{-}\mathsf{Sum} \circ \mathscr{C}[2^{n^\varepsilon}]$ circuits within ℓ_1-distance δ on almost every input length, where $\delta > 0$ is the absolute constant in Theorem 4.4.1.*
3. *There is a constant $\varepsilon > 0$ and a CAPP algorithm for $\mathscr{C}[2^{n^\varepsilon}]$ with E^{NP} preprocessing, 2^{n-n^ε} query time, and inverse-circuit-size error.*
4. *There is a constant $c \geq 1$ such that for every good function $s(n)$, there is a CAPP algorithm for $\mathscr{C}[s]$ circuits with E^{NP} preprocessing, $2^{\log^c s}$ query time, and inverse-circuit-size error.*
5. *There is a constant $\varepsilon > 0$ and an E^{NP}-computable PRG with seed length $n-1$ that $(1/2^{n^\varepsilon})$-fools $\mathscr{C}$ circuits of size 2^{n^ε}.*
6. *There is a constant $c \geq 1$ and an E^{NP}-computable PRG with seed length $\log^c n$ that $(1/n)$-fools $\mathscr{C}$ circuits of size n.*

Proof. (4) $\implies$ (3) is trivial.

(6) $\implies$ (5) can be proved by padding the circuit with dummy inputs.

(6) $\implies$ (4): The proof is the same as (6) $\implies$ (4) in Theorem 4.6.1.

(5) $\implies$ (3): Let $G : \{0,1\}^{n-1} \to \{0,1\}^n$ be an E^{NP}-computable PRG that $(1/2^{n^\varepsilon})$-fools $\mathscr{C}$ circuits of size 2^{n^ε}. Let $k := n^{\varepsilon/2}$, it can be shown using a hybrid argument that

$$G'(s_1, s_2, \ldots, s_k) = G(s_1)G(s_2)\ldots G(s_k)$$

is a PRG that $(k/2^{n^\varepsilon})$-fools $\mathscr{C}$ circuits of size 2^{n^ε} and nk inputs. The desired CAPP algorithm works as follows: $\mathsf{Prep}(1^N)$ calculates $n := N^{1/(1+\varepsilon/2)}$, and uses its NP oracle to compute the entire range of $G(\{0,1\}^{n-1})$. Then, given a $\mathscr{C}$ circuit $C : \{0,1\}^N \to \{0,1\}$ of size $2^{N^{\varepsilon/4}} < 2^{n^\varepsilon}$, it uses G' to derandomise C in $2^{N-n^{\varepsilon/2}} \cdot 2^{N^{\varepsilon/4}} \leq 2^{N-N^{\varepsilon/6}}$ time, within additive error $k/2^{n^\varepsilon} \leq 2^{-N^{\varepsilon/6}}$. That is, (3) holds for $\varepsilon' := \varepsilon/6$.

(3) $\implies$ (2) follows from Theorem 4.4.1 and the hypothesis that $\mathscr{C}$ is typical, complete, computes PARITY, and is closed under top NC^0 circuits.

(2) $\implies$ (1) follows from the XOR lemma in [CLW20]. In particular, suppose that $L \in \mathsf{E}^{\mathsf{NP}}$ cannot be approximated by $[0,1]\text{-}\mathsf{Sum} \circ \mathscr{C}[2^{n^\varepsilon}]$ circuits within ℓ_1-distance $1/3$ on almost every input length. Let $\delta := 1/3$, $k := O(n^{\varepsilon/2})$, and

$$\varepsilon_k := (1-\delta)^{k-1}(1/2-\delta) < 2^{-n^{\varepsilon/2}}.$$

By Theorem 4.2.9, $L^{\oplus k}$ cannot be $(1/2+\varepsilon_k)$-approximated by $\mathscr{C}$ circuits of size $2^{n^{\varepsilon/2}}$. Since $L \in \mathsf{E}^{\mathsf{NP}}$, we also have $L^{\oplus k} \in \mathsf{E}^{\mathsf{NP}}$. Note that $L^{\oplus k}$ is a function on $\ell := kn = O(n^{1+\varepsilon/2})$ input bits. Therefore, for $\varepsilon' := 0.49\varepsilon/(1+\varepsilon)$, $L^{\oplus k}$ cannot be $(1/2+1/2^{\ell^{\varepsilon'}})$-approximated by $\mathscr{C}$ circuits of size $2^{\ell^{\varepsilon'}}$.

(1) $\implies$ (6): Let $L \in \mathsf{E}^{\mathsf{NP}}$ be a language that cannot be $(1/2+1/2^{n^\varepsilon})$-approximated by $\mathscr{C}[2^{n^\varepsilon}]$ circuits on almost every input length. Let $c \geq 2$ be a large enough constant such that $\mathscr{C} \circ \mathsf{Junta}_a$ circuits, where the top $\mathscr{C}$ circuit has size n, can be simulated by $\mathscr{C}$ circuits of size n^c. We apply the Nisan–Wigderson generator [NW94] with the following parameters:

$$\ell := \log^{c/\varepsilon} n, m := n, a := \log n, t := O(\ell^2 \cdot m^{1/a}/a) := O(\ell^2) \leq \log^{3c/\varepsilon} n.$$

From (1) we have that there exists a function $f : \{0,1\}^\ell \to \{0,1\}$ in E^{NP} that cannot be $(1/2+1/2^{\ell^\varepsilon}) = (1/2+1/n^c)$-approximated by $\mathscr{C}$ circuits of size n^c. Then f cannot be $(1/2+1/n^c)$-approximated by $\mathscr{C}[n] \circ \mathsf{Junta}_a$ circuits. By Theorem 4.2.1, there is a function $G : \{0,1\}^{2^\ell} \times \{0,1\}^t \to \{0,1\}^m$ computable in $\mathrm{poly}(m, 2^t) \leq 2^{O(t)}$ time, such that $G(tt(f), -)$ is a PRG that $(1/n^{c-1})$-fools every $\mathscr{C}[n]$ circuit. Since $f \in \mathsf{E}^{\mathsf{NP}}$, the generator $G(tt(f), \cdot)$ is computable in E^{NP}. □

Chapter 5

Constructions of Rectangular PCPs of Proximity

5.1 Construction of Smooth and Rectangular PCPP

Recall that a PCPP verifier is *smooth* if every bit of the proof is equally likely to be queried, i.e., the distribution of a random query position over a random seed is uniformly random. We do not impose any smoothness requirement on the input oracle.

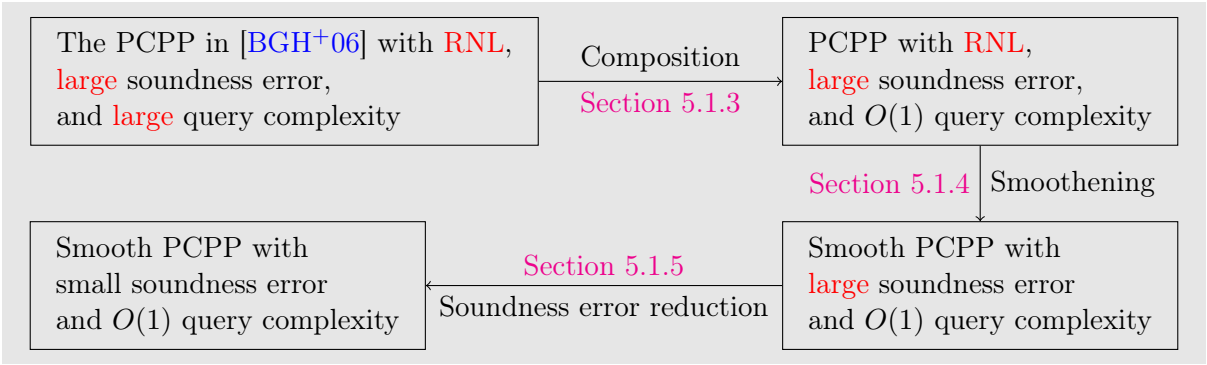
In this section, we construct a smooth and rectangular PCP of proximity. Our construction follows closely from previous ones: Based on [BGH+06], [BHPT24] constructed a smooth and rectangular PCP, and a careful inspection of their techniques reveals that a smooth and rectangular PCP of *proximity* can be constructed in a similar way. Still, we present an (almost) self-contained proof of the construction here. As this section is quite technical, we give a brief overview of the construction.

Overview of Section 5.1

- Instead of constructing a smooth PCPP verifier directly, we will construct a rectangular PCPP verifier with *rectangular neighbour listing* (RNL) property, following [BHPT24]. We present the definition of the RNL property in Section 5.1.1.
- In Section 5.1.2, we show that the PCPP verifier in [BGH+06] is a robust and rectangular PCPP verifier with RNL property. Previously, a robust and rectangular *PCP* with RNL property was constructed in [BHPT24]; we show that one can construct a *PCPP* with the same properties.
- The query complexity of the PCPP verifier in Section 5.1.2 is somewhat large and we need to reduce it by *PCPP composition*. In Section 5.1.3, we prove such a composition theorem: we can compose a robust and rectangular PCPP verifier for a language L (the *outer* PCPP verifier) and a PCPP verifier for a variant of the circuit-evaluation problem (the *inner* PCPP verifier) to obtain a rectangular PCPP verifier for L whose query complexity is at most the query complexity of the inner PCPP verifier. Moreover, the composed PCPP verifier has the RNL property if the outer PCPP verifier has the RNL property. A minor technicality is that this rectangular PCPP verifier will also take some ROP parity-check bits (see Definition 2.5.4).
- In Section 5.1.4, we show how to smoothen a PCPP with RNL property. That is, given a PCPP with RNL property, we construct another PCPP with similar parameters that is smooth.

- The soundness error of the PCPP we constructed is only a large constant (i.e., close to 1). In Section 5.1.5, we show that the soundness error can be reduced to an arbitrarily small constant with an $O(1)$ blow-up to the query complexity.
- We wrap all these components up and set the parameters in Section 5.1.6.

We remark that PCPP composition (Section 5.1.3) does not seem to preserve smoothness (if the inner PCPP is not known to be smooth) and the soundness error reduction (Section 5.1.5) does not seem to preserve the RNL property, hence we choose to apply our machinery in the above specific order, i.e.:



In this section, $\mathsf{NTIME}[T(n)]$ always refers to $\mathsf{NTIME}_{\mathsf{TM}}[T(n)]$, i.e., we only consider the Turing machine model. Recall that a function $f : \mathbb{N} \to \mathbb{N}$ is *good* if given the input n in binary, we can compute $f(n)$ (also in binary) in time $\mathrm{poly}(\log n, \log f(n))$.

5.1.1 Rectangular Neighbour Listing

Definition 5.1.1. Let V be a rectangular PCPP verifier for some language L with randomness complexity r and query complexity q. A *configuration* is defined as a pair $(\mathsf{seed}, k) \in \{0,1\}^r \times [q]$. It is said to be a proof (resp. input) configuration if the verifier with randomness seed will query the proof (resp. input) oracle on the k-th query. Two configurations (seed_1, k_1) and (seed_2, k_2) are said to be *neighbours* if the verifier will access the same bit of the same oracle with randomness seed_1 on the k_1-th query, or with randomness seed_2 on the k_2-th query.

We define the notion of *rectangular neighbour listing* [BHPT24]. A minor difference between our definition and the one from [BHPT24] is that we require a procedure A_{shared} that only sees the shared randomness and outputs ℓ (the length of the neighbour list), self (the index of the configuration in the neighbour list), and k_i (the query indices of every neighbor). In [BHPT24], both A_{row} (which only sees row-part randomness) and A_{col} (which only sees column-part randomness) output these data, and it is required that the outputs of A_{row} and A_{col} are consistent.

Definition 5.1.2 (Rectangular Neighbour Listing)**.** Let L be a language and V be a rectangular PCPP verifier for L with row randomness complexity r_{row}, column randomness complexity r_{col}, and shared randomness complexity r_{shared}. We say V has $t_{\mathsf{RNL}}(n)$*-time rectangular neighbour listing* property if there are three $t_{\mathsf{RNL}}(n)$-time algorithms A_{shared}, A_{row}, and A_{col} such that the following conditions hold:

1. The shared randomness $\mathsf{seed.shared} \in \{0,1\}^{r_{\mathsf{shared}}}$ consists of $\mathsf{seed.shared.row} \in \{0,1\}^{r_{\mathsf{shared}}/2}$ and $\mathsf{seed.shared.col} \in \{0,1\}^{r_{\mathsf{shared}}/2}$, i.e., $\mathsf{seed.shared} = (\mathsf{seed.shared.row}, \mathsf{seed.shared.col})$.

2. Let $(\mathsf{seed}, k) = (\mathsf{seed.row}, \mathsf{seed.col}, \mathsf{seed.shared}, k)$ be a configuration, where $\mathsf{seed.shared} = (\mathsf{seed.shared.row}, \mathsf{seed.shared.col})$. The algorithms A_{shared}, A_{row}, and A_{col} list all the neighbours of (seed, k) in a "rectangular and synchronised" fashion:

   - Given $\mathsf{seed.shared}$ and k, A_{shared} will output an ordered list $\mathsf{NList}_{\mathsf{shared}}(\mathsf{seed}, k) := \{k_i\}$ and a number $\mathsf{self}(\mathsf{seed}, k)$. Let $\ell(\mathsf{seed}, k)$ denote the length of $\mathsf{NList}_{\mathsf{shared}}(\mathsf{seed}, k)$.
   - Given the row-part randomness $(\mathsf{seed.row}, \mathsf{seed.shared}, k)$ as input, A_{row} will output an ordered list $\mathsf{NList}_{\mathsf{row}}(\mathsf{seed}, k) := \{(\mathsf{seed}_i.\mathsf{row}, \mathsf{seed}_i.\mathsf{shared.row})\}_{i\in[\ell(\mathsf{seed},k)]}$.
   - Given the column-part randomness $(\mathsf{seed.col}, \mathsf{seed.shared}, k)$ as input, A_{col} will output an ordered list $\mathsf{NList}_{\mathsf{col}}(\mathsf{seed}, k) := \{(\mathsf{seed}_i.\mathsf{col}, \mathsf{seed}_i.\mathsf{shared.col})\}_{i\in[\ell(\mathsf{seed},k)]}$.
   - The "zipped" list of $\mathsf{NList}_{\mathsf{shared}}(\mathsf{seed}, k)$, $\mathsf{NList}_{\mathsf{row}}(\mathsf{seed}, k)$, and $\mathsf{NList}_{\mathsf{col}}(\mathsf{seed}, k)$

     $$\mathsf{NList}(\mathsf{seed}, k) := \Big\{(\mathsf{seed}_i.\mathsf{row}, \mathsf{seed}_i.\mathsf{col}, \mathsf{seed}_i.\mathsf{shared.row}, \mathsf{seed}_i.\mathsf{shared.col}, k_i)\Big\}_{i\in[\ell(\mathsf{seed},k)]}$$

     is a list of all the neighbours of (seed, k). Moreover, the $\mathsf{self}(\mathsf{seed}, k)$-th entry of $\mathsf{NList}(\mathsf{seed}, k)$ is the configuration (seed, k) itself.
   - For every pair of neighbours (seed_1, k_1) and (seed_2, k_2), the two ordered lists $\mathsf{NList}(\mathsf{seed}_1, k_1)$ and $\mathsf{NList}(\mathsf{seed}_2, k_2)$ are exactly the same.

3. Moreover, we say that the RNL can be computed by *projections*, if for every fixed $\mathsf{seed.shared}$, $\mathsf{NList}_{\mathsf{row}}$ can be computed by a projection over $\mathsf{seed.row}$, $\mathsf{NList}_{\mathsf{col}}$ can be computed by a projection over $\mathsf{seed.col}$, and the descriptions of these two projections can be computed efficiently given $\mathsf{seed.shared}$.

5.1.2 A Rectangular PCPP with RNL Property

We start by reviewing the PCPP for $\mathsf{NTIME}[T(n)]$ constructed in [BGH+05, BGH+06]. We verify that it is a rectangular PCPP with the rectangular neighbour listing property as in [BHPT24]. We summarise its properties into the following theorem:

Theorem 5.1.3. *For all constants $\delta > 0$, there is a constant $\rho \in (0,1)$ such that the following holds. Let $m = m(n)$, $T(n)$, $w_{\mathsf{proof}}(n)$, $w_{\mathsf{input}}(n)$ be good functions such that $1 \le m \le (\log T(n))^{0.1}$, $n \le T(n) \le 2^{\mathrm{poly}(n)}$, $w_{\mathsf{proof}}(n) \le \log T(n)$, and $w_{\mathsf{input}}(n) \le \log n$. Then there are good functions $h_{\mathsf{proof}}(n)$ and $h_{\mathsf{input}}(n)$ satisfying*

$$h_{\mathsf{proof}}(n) = \log T(n) + \Theta(m \log\log T(n)) - w_{\mathsf{proof}}(n), \qquad \text{and}$$
$$h_{\mathsf{input}}(n) = \lceil \log n \rceil - w_{\mathsf{input}}(n).$$

such that the following holds.

Suppose that $h_{\mathsf{proof}}, w_{\mathsf{proof}} \ge (4/m)\log T(n)$, and that for some absolute constant $C \ge 1$,

$$\frac{w_{\mathsf{input}}(n)}{w_{\mathsf{proof}}(n)}, \frac{h_{\mathsf{input}}(n)}{h_{\mathsf{proof}}(n)} \le 1 - \frac{Cm\log\log T(n)}{\log T(n)}.$$

Let $W_{\mathsf{proof}}(n) := 2^{w_{\mathsf{proof}}(n)}$, $H_{\mathsf{proof}}(n) := 2^{h_{\mathsf{proof}}(n)}$, $W_{\mathsf{input}}(n) := 2^{w_{\mathsf{input}}(n)}$, and $H_{\mathsf{input}}(n) := 2^{h_{\mathsf{input}}(n)}$. Then $\mathsf{NTIME}[T(n)]$ has a rectangular neighbour listable, robust, and rectangular PCP

Soundness error	$1-\rho$
Proximity parameter	δ
Robustness parameter	ρ
Row randomness	$h_{\mathsf{proof}} - (4/m)\log T(n)$
Column randomness	$w_{\mathsf{proof}} - (4/m)\log T(n)$
Shared randomness	$(7/m)\log T(n) + O(\log\log T(n) + m\log m)$
Query complexity	$T(n)^{1/m} \cdot \mathrm{polylog}(T(n))$
Decision complexity	
RNL time complexity	$\mathrm{poly}(\log T(n), m^m)$

Table 5.1: Parameters of the PCPP constructed in Theorem 5.1.3.

of proximity with an $H_{\mathsf{proof}}(n) \times W_{\mathsf{proof}}(n)$ proof matrix and an $H_{\mathsf{input}}(n) \times W_{\mathsf{input}}(n)$ input matrix, whose other parameters are specified in Table 5.1. The query indices of this PCPP can be computed by projections (in the sense of Definition 2.5.6) and the RNL can also be computed by projections (in the sense of Definition 5.1.2).

We note that the query complexity of the PCPP in Theorem 5.1.3 is $q \approx T^{1/m}$. Correspondingly, the total amount of randomness it uses is roughly $h_{\mathsf{proof}} + w_{\mathsf{proof}} - (1/m)\log T \approx \log(|\Pi|/q)$ where $|\Pi|$ denotes the proof length. Looking ahead, in Section 5.1.3, we will compose this PCPP with another "inner" PCPP [Mie09] to reduce the query complexity to a constant.

We need an efficient construction of small-biased sets. In particular, let $\lambda > 0$, $m \in \mathbb{N}$, and $\mathbb{F}$ be a finite field of characteristic 2, we need a λ-biased set $S_\lambda \subseteq \mathbb{F}^m$. Besides being λ-biased, S_λ should possess an additional property: for every element $\vec{y} \in S_\lambda$, the first coordinate of $\vec{y}$ is non-zero. It is claimed in [BHPT24] that (under suitable conditions) for any $(\lambda/4)$-biased set $S_{\lambda/4}$, if we remove every element $\vec{y} \in S_{\lambda/4}$ with $y_1 = 0$, then the remaining set is still λ-biased. For completeness, we provide a proof for this claim at the end of this subsection.

Lemma 5.1.4. *Let $\lambda < 0.1$, q, m be integers such that $q \geq \log \frac{4}{\lambda}$, and let $\mathbb{F} = \mathrm{GF}(2^q)$. There is a deterministic polynomial-time algorithm that on input $(1^m, 1^q, 1^{\lceil 1/\lambda \rceil})$, outputs a λ-biased set $S_\lambda \subseteq (\mathbb{F} \setminus \{0\}) \times \mathbb{F}^{m-1}$ of size $O((qm/\lambda)^2)$.*

Let $L \in \mathsf{NTIME}[T(n)]$, we describe the PCPP verifier for L. The PCPP verifier receives two oracles: the input oracle Π_{input} (consisting of the input in verbatim) and the proof oracle Π_{proof}.

Set up. Let α be a universal constant as defined in the proof of [BGH+05, Theorem 6.4], c be the universal constant in [BGH+06, Lemma 8.11]. We set the following parameters:

$$t := \log T(n),\ h := \lceil (t+3)/m \rceil,\ f := h + \alpha \log_2 t,\ \lambda := \min\{1/(ct), 1/(m^{2cm})\}.$$

We work with the field $\mathbb{F} := \mathrm{GF}(2^f)$. We treat $\mathbb{F}$ as a vector space of dimension f over $\mathrm{GF}(2)$ and let $e_1, \ldots, e_f$ be its basis. Each element $v \in \mathbb{F}$ can then be written as $v = \sum_{i=1}^{f} e_i b_i$ for $b_i \in \mathrm{GF}(2)$, and we denote the binary representation of v as $\mathsf{bin}(v) = (b_1 b_2 \ldots b_f)$. Let H be the vector space spanned by $e_1, e_2, \ldots, e_h$. We also define two bijections

$$\mathsf{bin}_{H^m} : H^m \to \{0,1\}^{hm} \quad \text{and} \quad \mathsf{bin}_{\mathbb{F}^m} : \mathbb{F}^m \to \{0,1\}^{fm}.$$

The bijection $\mathsf{bin}_{\mathbb{F}^m}$ is the usual one: it maps $(b_1e_1 + \cdots + b_fe_f, b_{f+1}e_1 + \cdots + b_{2f}e_f, \ldots, b_{(m-1)f+1}e_1 + \cdots + b_{mf}e_f)$ to $(b_1b_2 \ldots b_{mf})$. We also treat binary strings as numbers where the leftmost bit is the least significant one and the rightmost bit is the most significant one. That is, the string $(b_1b_2 \ldots b_{mf})$ is treated as $\sum_{i=1}^{mf} b_i 2^{i-1}$.

We postpone the definition of the bijection bin_{H^m} as it is a bit technical. Roughly speaking, the reason is that we want to map the input to a matrix of dimension $H_{\mathsf{input}} \times W_{\mathsf{input}}$. More specifically, the input string occupies positions $[2^{t+1} + 1, 2^{t+1} + n]$, and we want to map this portion into a subset of H^m which corresponds to a rectangle of dimensions $H_{\mathsf{input}} \times W_{\mathsf{input}}$.

We use the following injection $I_t : [n] \to H^m$ to project the input to H^m:

$$I_t(i) = \mathsf{bin}_{H^m}^{-1}(2^{t+1} + i). \tag{5.1}$$

That is, the i-th input bit will be embedded into the position $I_t(i) \in H^m$. Define the set $I := \{I_t(k) : k \le |\Pi_{\mathsf{input}}|\}$, then the input will be stored on the index set I.

Remark 5.1.5. The definition of I_t (5.1) is derived from [BGH+05] as follows.

1. First, we reduce L to the Generalised de Bruijn Graph Coloring problem ([BGH+05, Definition 4.3]). The i-th bit of Π_{input} is mapped to the $(2^{t+1} + i)$-th node of the first layer.
2. Then, we reduce the above coloring problem to the Multivariate Algebraic Constraint Satisfaction Problem ([BGH+05, Definition 6.4]). In this step, for every i, the i-th node in (the first layer of) the de Bruijn graph is mapped to the vector $\mathsf{bin}_{H^m}^{-1}(i) \in H^m$.

Combining the above two steps, it follows that the i-th bit of Π_{input} is mapped to $I_t(i)$.

The PCPP proof will have length $|\mathbb{F}|^m \cdot \ell$ for some $\ell = \mathrm{polylog}(T)$; we treat it as an oracle $\Pi_{\mathsf{proof}} : \mathbb{F}^m \to \{0,1\}^\ell$.[1] Without loss of generality we may assume ℓ is a power of 2. The i-th bit of the proof (viewed as a string of length $|\mathbb{F}|^m \cdot \ell$) is equal to the k-th bit of $\Pi_{\mathsf{proof}}[\mathsf{bin}_{\mathbb{F}^m}^{-1}(j)]$, where $j := \lfloor i/\ell \rfloor$ and $k := i \bmod \ell$.

Lines. A *line* $\mathcal{L}$ over $\mathbb{F}^m$ is a set of the form $\{\vec{x} + t\vec{y} : t \in \mathbb{F}\}$. Here $\vec{x} \in \mathbb{F}^m$ is called the *intercept* of $\mathcal{L}$ and $\vec{y} \in \mathbb{F}^m$ is called the *direction* of $\mathcal{L}$. The PCPP verifier will make queries along the following two types of lines over $\mathbb{F}^m$:

- A *first-axis parallel line* is a line where $\vec{y} = (1, 0, 0, \ldots, 0)$ and $\vec{x} \in \mathbb{F}^m$. To sample a uniform first-axis parallel line, it suffices to choose $\vec{x}$ from $\{0\} \times \mathbb{F}^{m-1}$ uniformly at random using $(m-1)\log(|\mathbb{F}|)$ bits of randomness.
- Fix a λ-biased set $S_\lambda \subseteq \mathbb{F}^m$ constructed in Lemma 5.1.4. A *pseudorandom line* is a line where $\vec{x} \in \mathbb{F}^m$ and $\vec{y} \in S_\lambda$. Each line has $|\mathbb{F}|$ different representations (since the intercepts $\vec{x} + t\vec{y}$ represent the same line for all $t \in \mathbb{F}$). Therefore, we specify a *canonical* representation for each line.

  To sample a pseudorandom line in the canonical way, we first choose $\vec{y}$ from S_λ uniformly at random, and then sample $\vec{x}$ from $\{0\} \times \mathbb{F}^{m-1}$ uniformly at random. (Note that the first

[1] Actually, in [BGH+05], each entry $\Pi_{\mathsf{proof}}(\vec{x})$ is an error-corrected version of a vector in $\mathbb{F}^{\mathrm{polylog}(T)}$. The use of error-correcting codes ensures that the PCPP verifier is robust.

coordinate of $\vec{y}$ is always non-zero, therefore every pseudorandom line intersects $\{0\} \times \mathbb{F}^{m-1}$.) This uses $\log(|S_\lambda|) + (m-1)\log(|\mathbb{F}|)$ bits of randomness.

Query pattern. To verify rectangularity (and RNL), it suffices to describe the query pattern of the verifier, i.e., the entries of Π_{input} and Π_{proof} that are queried for a given randomness.

Let seed be the randomness of the verifier, which has length $\log\big(|S_\lambda| \cdot |\mathbb{F}|^{m-1}\big)$. We partition seed into

$$\mathsf{seed} := (R_2, R_3, \ldots, R_m, R_y),$$

where each R_i $(2 \le i \le m)$ has length $\log|\mathbb{F}|$ and corresponds to an element in $\mathbb{F}$, and R_y has length $\log|S_\lambda|$ and corresponds to an element in S_λ. Then seed determines a first-axis parallel line $\mathcal{L}_0$ and a canonical pseudorandom line $\mathcal{L}_1$ as follows. The intercepts of both $\mathcal{L}_0$ and $\mathcal{L}_1$ are $\vec{x} = (0, R_2, R_3, \ldots, R_m)$; the direction of $\mathcal{L}_0$ is $(1, 0, 0, \ldots, 0)$ and the direction of $\mathcal{L}_1$ is the R_y-th element of S_λ.

Let $\mathsf{shift} : \mathbb{F}^m \to \mathbb{F}^m$ denote the cyclic shift one step to the left; i.e.,

$$\mathsf{shift}(a_1, a_2, \ldots, a_m) = (a_2, a_3, \ldots, a_m, a_1).$$

For a line $\mathcal{L}$, $\mathsf{shift}(\mathcal{L})$ denotes the set $\{\mathsf{shift}(x) : x \in \mathcal{L}\}$.

The query pattern of the PCPP verifier is easy to describe:

- For every point $\vec{x}$ on $\mathcal{L}_0$, $\mathsf{shift}(\mathcal{L}_0)$, and $\mathcal{L}_1$, it makes a query to $\Pi_{\mathsf{proof}}[\vec{x}]$.
- For every $\vec{x} \in \mathcal{L}_1 \cap I$, it also makes a query to $\Pi_{\mathsf{input}}[I_t^{-1}(\vec{x})]$.

Remark 5.1.6. This query pattern is derived from [BGH+06] (see also [BHPT24, Section C]). In particular:

- Robust Low-Degree Test makes queries to $\Pi_{\mathsf{proof}}[\mathcal{L}_1]$;
- Robust Identity Test makes queries to $\Pi_{\mathsf{proof}}[\mathcal{L}_0]$;
- Robust Edge-Consistency Test makes queries to $\Pi_{\mathsf{proof}}[\mathcal{L}_0]$ and $\Pi_{\mathsf{proof}}[\mathsf{shift}(\mathcal{L}_0)]$;
- Robust Zero Propagation Test makes queries to $\Pi_{\mathsf{proof}}[\mathcal{L}_0]$ and $\Pi_{\mathsf{proof}}[\mathsf{shift}(\mathcal{L}_0)]$;
- Robust Proximity Test makes queries to $\Pi_{\mathsf{proof}}[\mathcal{L}_1]$ and $\Pi_{\mathsf{input}}[I_t^{-1}(\mathcal{L}_1 \cap I)]$.

As Robust Proximity Test was not needed in and hence not described by [BHPT24] (since they were constructing a PCP instead of a PCPP), we describe its details here. This test queries Π on every point in $\mathcal{L}_1$ and unbundles the answers to obtain the values of A_0 (a certain proof polynomial in [BGH+05, Definition 6.3]) on $\mathcal{L}_1$. Then it queries Π_{input} on every point $k \in \mathcal{L}_1 \cap I$ and checks whether $\Pi_{\mathsf{input}}[k] = f^t_{\mathsf{extract}}(A_0[I_t(k)])$ holds, where f^t_{extract} is a certain function defined in [BGH+05, Definition 6.3].

Rectangularity of the PCPP Verifier

Given the query pattern of the verifier described above, we are ready to show that the verifier is rectangular. The verifier makes $3|\mathbb{F}|$ queries to Π_{proof}; let us call them $(\vec{a}_1, \ldots, \vec{a}_{|\mathbb{F}|})$, $(\vec{a}_{|\mathbb{F}|+1}, \ldots, \vec{a}_{2|\mathbb{F}|})$, and $(\vec{a}_{2|\mathbb{F}|+1}, \ldots, \vec{a}_{3|\mathbb{F}|})$, which are on $\mathcal{L}_0$, $\mathsf{shift}(\mathcal{L}_0)$, and $\mathcal{L}_1$ respectively. The following lemma shows that for each $1 \le j \le m$, the j-th coordinate of each query only depends on R_j, R_{j+1}, and R_y.

Lemma 5.1.7. *Fix the random string* $\mathsf{seed} = (R_2, R_3, \ldots, R_m, R_y)$*, and for convenience define* $R_1 = R_{m+1} = 0^{\log|\mathbb{F}|}$*. Denote each* $\vec{a}_i = (a_{i,1}, \ldots, a_{i,m}) \in \mathbb{F}^m$*. Then for every* $j \in [m]$*,*

$(a_{1,j}, a_{2,j}, \ldots, a_{3|\mathbb{F}|,j})$ *only depends on* R_j, R_{j+1}, *and* R_y.

Moreover, for any fixed R_y, $(a_{1,j}, a_{2,j}, \ldots, a_{3|\mathbb{F}|,j})$ *is a projection over* (R_j, R_{j+1}).

Proof. Let $h_1, h_2, \ldots, h_{|\mathbb{F}|}$ be an enumeration of all the elements in $\mathbb{F}$. Then

$$\begin{aligned}(a_{1,j}, a_{2,j}, \ldots, a_{3|\mathbb{F}|,j}) = (&R_j + h_1 y_{1,j}, R_j + h_2 y_{1,j}, \ldots, R_j + h_{|\mathbb{F}|} y_{1,j},\\ &R_{j+1} + h_1 y_{2,j}, R_{j+1} + h_2 y_{2,j}, \ldots, R_{j+1} + h_{|\mathbb{F}|} y_{2,j},\\ &R_j + h_1 y_{3,j}, R_j + h_2 y_{3,j}, \ldots, R_j + h_{|\mathbb{F}|} y_{3,j})\end{aligned} \tag{5.2}$$

where $\vec{y}_1, \vec{y}_2, \vec{y}_3 \in \mathbb{F}^m$ is defined to be $\vec{y}_1 = (1, 0, 0, \ldots, 0), \vec{y}_2 = (0, 0, \ldots, 0, 1)$ and $\vec{y}_3$ is R_y-th vector in S_λ.

It is easy to see that (5.2) only depends on R_j, R_{j+1}, and R_y. Moreover, each coordinate $a_{i,j}$ is the sum of R_j or R_{j+1} with an element of the form $h_k y_{l,j}$. Note that addition over $\mathbb{F} = \mathrm{GF}(2^f)$ is equivalent to bitwise XOR over $\{0,1\}^f$, it follows that (5.2) is indeed a projection over (R_j, R_{j+1}). □

Definition of bin_{H^m}. Now we define $\mathsf{bin}_{H^m} : H^m \to \{0,1\}^{hm}$. Roughly speaking, the goal of this definition is to "shape" the input oracle as a rectangle of size $H_{\mathsf{input}} \times W_{\mathsf{input}}$.

We treat every element in H as a binary string of length h. For a vector $\vec{a} = (a_1, a_2, \ldots, a_m) \in H^m$, the natural encoding of $\vec{a}$ is the concatenation of $a_1, a_2, \ldots, a_m$ (from the lowest bits to the highest bits), where each a_i is treated as an element in $\{0,1\}^h$. We denote this encoding as $\mathsf{bin}^\circ \in \{0,1\}^{mh}$. Let $k := \lceil (w_{\mathsf{proof}} - \log \ell) \cdot (h/f) \rceil$, we define $\mathsf{bin}_{H^m}(\vec{a})$ to be the concatenation of (from the lowest bits to the highest bits):

$$\mathsf{bin}^\circ[1, w_{\mathsf{input}}], \mathsf{bin}^\circ[k+1, k+h_{\mathsf{input}}], \mathsf{bin}^\circ[w_{\mathsf{input}}+1, k], \text{ and } \mathsf{bin}^\circ[k+h_{\mathsf{input}}+1, hm]. \tag{5.3}$$

Some intuitions behind the definition of bin_{H^m} are as follows. As we will show later, V_{col} can compute the lowest k bits of bin° and V_{row} can compute the rest $hm - k$ bits. To make the input matrix size $H_{\mathsf{input}} \times W_{\mathsf{input}}$, among the lowest $\lceil \log n \rceil$ bits, there needs to be w_{input} bits computed by V_{col} and $\lceil \log n \rceil - w_{\mathsf{input}} = h_{\mathsf{input}}$ bits computed by V_{row}. In our definition of bin_{H^m}, we simply put the lowest w_{input} bits computed by V_{col} and the lowest h_{input} bits computed by V_{row} as the lower $\lceil \log n \rceil$ bits of $\mathsf{bin}_{H^m}(\vec{a})$, and put the rest bits as the higher bits of $\mathsf{bin}_{H^m}(\vec{a})$.

We define $c_1 := \lceil w_{\mathsf{input}}/h \rceil$, $c_2 := \lceil k/h \rceil$, $c_3 := \lceil (k + h_{\mathsf{input}})/h \rceil$. Then the w_{input}-th bit of bin° is in a_{c_1}, the k-th bit of bin° is in a_{c_2}, and the $(k + h_{\mathsf{input}})$-th bit of bin° is in a_{c_3}.

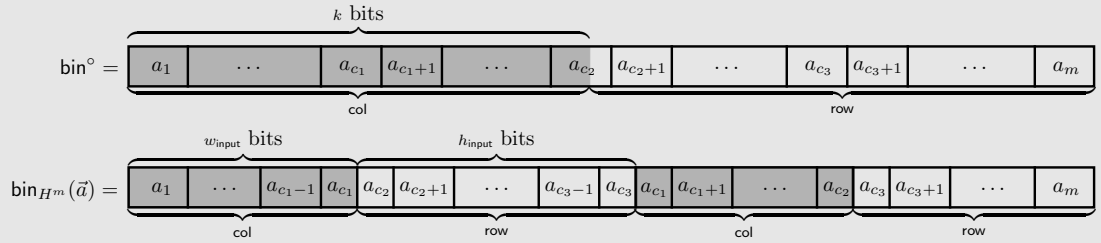


Figure 5.1: The bit-string $\mathsf{bin}_{H^m}(a_1, a_2, \ldots, a_m)$. In this figure, the leftmost bits are the *least* significant ones.

Note that

$$w_{\mathsf{input}} \le w_{\mathsf{proof}}(1-\Theta(m\log f)/f) \le k, \qquad \text{and}$$
$$k + h_{\mathsf{input}} \le (w_{\mathsf{proof}} - \log \ell)\cdot(h/f) + h_{\mathsf{proof}}\cdot(h/f) \le hm. \tag{5.4}$$

Here, (5.4) is because $w_{\mathsf{proof}} + h_{\mathsf{proof}} = \log|\Pi_{\mathsf{proof}}| = fm + \log \ell$. Since $w_{\mathsf{input}} \le k$ and $k + h_{\mathsf{input}} \le hm$, (5.3) is well-defined.

Partition of the random seed. We partition seed into:

$$\begin{aligned}\mathsf{seed.col} &= (R_3, R_4, \ldots, R_{c_2-1}),\\ \mathsf{seed.row} &= (R_{c_2+2}, R_{c_2+3}, \ldots, R_{m-1}), \text{ and}\\ \mathsf{seed.shared} &= (R_2, R_{c_2}, R_{c_2+1}, R_m, R_y).\end{aligned}$$

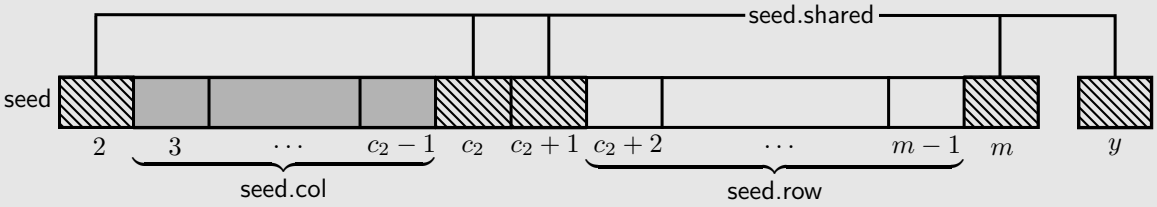


Figure 5.2: The partition of the random seed.

By Lemma 5.1.7, knowing R_j, R_{j+1}, and R_y allows us to calculate $(a_{1,j}, a_{2,j}, \ldots, a_{3|\mathbb{F}|,j})$. Therefore, V_{col} is able to calculate the $1, 2, \ldots, c_2$-th coordinates of each query; V_{row} is able to calculate the $c_2, c_2+1, \ldots, m$-th coordinates of each query. (We remark that for rectangularity, it suffices to add R_{c_2}, R_{c_2+1}, and R_y into $\mathsf{seed.shared}$, but for the RNL property that we discuss later, it will be crucial that R_2 and R_m are also in $\mathsf{seed.shared}$.)

In this partition, we have $|\mathsf{seed.col}| = (c_2-3)f \ge w_{\mathsf{proof}} - 4t/m$ and $|\mathsf{seed.row}| = (m - c_2 - 2)f \ge h_{\mathsf{proof}} - 4t/m$. For technical convenience, we move some random bits into $\mathsf{seed.shared}$ so that $|\mathsf{seed.col}| = w_{\mathsf{proof}} - 4t/m$ and $|\mathsf{seed.row}| = h_{\mathsf{proof}} - 4t/m$; this means that

$$\begin{aligned}|\mathsf{seed.shared}| &= (m-1)f + |R_y| - (w_{\mathsf{proof}} + h_{\mathsf{proof}}) + 8t/m\\ &\le (m-1)f + O(\log(fm/\lambda)) - (\log\ell + mf) + 8t/m\\ &\le 8t/m + O(\log t + m\log m) - f\\ &\le 7t/m + O(\log t + m\log m).\end{aligned}$$

The predicates V_{type}, V_{row}, and V_{col}. Recall that we treat Π_{proof} as an oracle whose entries are length-ℓ strings and we make $3|\mathbb{F}|$ queries to Π_{proof}. This means that we actually make $3|\mathbb{F}|\ell$ queries to the bit-string corresponding to Π_{proof}. We also make $|\mathbb{F}|$ queries to Π_{input}.

It is easy to describe V_{type}: the first $3|\mathbb{F}|\ell$ queries are to the proof oracle and the last $|\mathbb{F}|$ queries are to the input oracle.

Now consider the i-th query where $1 \le i \le 3|\mathbb{F}|\ell$; these are queries made to Π_{proof}. Let $j := \lfloor (i-1)/\ell \rfloor$ and $j' := (i-1) \bmod \ell$, then the i-th query probes the j'-th bit of $\Pi_{\mathsf{proof}}[\vec{a}_j]$,

where $\vec{a}_j \in \mathbb{F}^m$ is defined above. We want to specify V_{row} and V_{col} such that the index of the i-th query is

$$\mathsf{irow}[i] \cdot W_{\mathsf{proof}} + \mathsf{icol}[i] = \mathsf{bin}_{\mathbb{F}^m}(\vec{a}_j) \cdot \ell + j'.$$

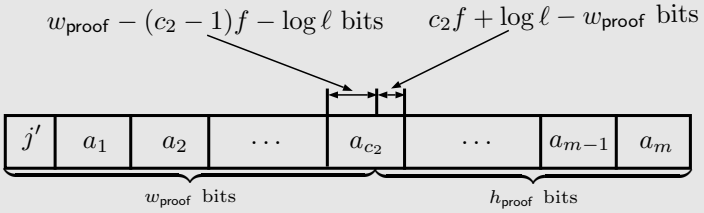


Figure 5.3: The binary representation of the address $\mathsf{bin}_{\mathbb{F}^m}(\vec{a}) \cdot \ell + j'$. In this figure, the leftmost bits are the *least* significant ones. The lowest w_{proof} bits are outputted by V_{col}, while the rest bits are outputted by V_{row}.

Recall that V_{col} can compute $a_{j,1}, a_{j,2}, \ldots, a_{j,c_2}$ using Lemma 5.1.7. Then, it outputs $\mathsf{icol}[i]$ as the concatenation of j', $a_{j,1}, \ldots, a_{j,c_2-1}$ and the lowest $w_{\mathsf{proof}} - (c_2 - 1)f - \log \ell$ bits of a_{j,c_2}.[2] Similarly, V_{row} can compute $a_{j,c_2}, a_{j,c_2+1}, \ldots, a_{j,m}$. It outputs $\mathsf{irow}[i]$ as the concatenation of the highest $c_2 f + \log \ell - w_{\mathsf{proof}}$ bits of a_{j,c_2}, and a_{j,c_2+1}, a_{j,c_2+2}, $\ldots$, $a_{j,m}$. It follows from Lemma 5.1.7 that the first $3|\mathbb{F}|\ell$ entries of V_{row} and V_{col} are projections over $\mathsf{seed.row}$ and $\mathsf{seed.col}$, computable in polynomial time given $\mathsf{seed.shared}$.

Finally, we consider the $(i + 3|\mathbb{F}|\ell)$-th query where $1 \le i \le |\mathbb{F}|$; these are queries made to Π_{input}. In particular, recall that the canonical pseudorandom line $\mathcal{L}_1$ consists of vectors $\vec{a}_{2|\mathbb{F}|+1}, \vec{a}_{2|\mathbb{F}|+2}, \ldots, \vec{a}_{3|\mathbb{F}|}$. If $\vec{a}_{2|\mathbb{F}|+i} \in I$ then we query the $I_t^{-1}(\vec{a}_{2|\mathbb{F}|+i})$-th bit of Π_{input}, otherwise we do nothing.

For notational convenience, we denote $\vec{a}^\star := \vec{a}_{2|\mathbb{F}|+i}$ and $\mathsf{bin}^\star := \mathsf{bin}_{H^m}(\vec{a}^\star)$. Now our goal is to specify V_{row} and V_{col} such that the index of the $(i + 3|\mathbb{F}|\ell)$-th query is

$$\mathsf{irow}\Big[i + 3|\mathbb{F}|\ell\Big] \cdot W_{\mathsf{input}} + \mathsf{icol}\Big[i + 3|\mathbb{F}|\ell\Big] = I_t^{-1}(\vec{a}^\star) = \mathsf{bin}^\star - 2^{t+1}.$$

If either V_{row} or V_{col} outputs $\perp$, or $\mathsf{bin}^\star - 2^{t+1} \notin [0, n)$, then we do not make this query.

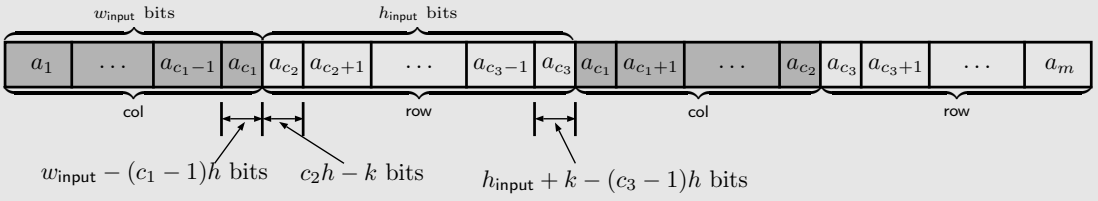


Figure 5.4: The bit-string $\mathsf{bin}_{H^m}(\vec{a})$. Again, the leftmost bits are the *least* significant ones.

Recall that V_{col} can compute $a_1^\star, a_2^\star, \ldots, a_{c_2}^\star$.

- If any of these elements are not in H, it outputs $\perp$.
- If any of the elements $a_{c_1+1}^\star, \ldots, a_{c_2}^\star$ are non-zero, then $\mathsf{bin}^\star - 2^{t+1}$ is not in the range $[0, n)$, and it outputs $\perp$.
- Otherwise, the concatenation of $a_1^\star$, $a_2^\star$, $\ldots$, $a_{c_1-1}^\star$, and the lowest $(w_{\mathsf{input}} - (c_1 - 1)h)$ bits

[2] Note that $c_2 = \lceil (w_{\mathsf{proof}} - \log \ell)/f \rceil$, which means the "dividing point" between w_{proof} and h_{proof} is in a_{c_2}.

of $a^\star_{c_1}$ is equal to $(\mathsf{bin}^\star \bmod W_{\mathsf{input}})$. In this case, it outputs

$$\mathsf{icol}\left[i + 3|\mathbb{F}|\ell\right] = \mathsf{bin}^\star \bmod W_{\mathsf{input}}.$$

Also recall that V_{row} can compute $a^\star_{c_2}, a^\star_{c_2+1}, \ldots, a^\star_m$.

- If any of these elements are not in H, it outputs $\perp$.
- If any of the elements $a^\star_{c_3+1}$, $a^\star_{c_3+2}$, $\ldots$, $a^\star_{m-1}$ is non-zero, or $a^\star_m \neq 2^{t+1-(m-1)h}$, then bin is not of the form $2^{t+1} + i$ $(0 \leq i < n)$, and it outputs $\perp$.[3]
- Otherwise, the concatenation of the lowest $(c_2 h - k)$ bits of $a^\star_{c_2}$, and $a^\star_{c_2+1}$, $a^\star_{c_2+2}$, $\ldots$, $a^\star_{c_3-1}$, and the lowest $(h_{\mathsf{input}} + k - (c_3 - 1)h)$ bits of $a^\star_{c_3}$, is equal to $\lfloor(\mathsf{bin}^\star - 2^{t+1})/W_{\mathsf{input}}\rfloor$. In this case, it outputs

$$\mathsf{irow}\left[i + 3|\mathbb{F}|\ell\right] = \lfloor(\mathsf{bin}^\star - 2^{t+1})/W_{\mathsf{input}}\rfloor.$$

RNL of the PCPP Verifier

Neighbours of (seed, i). Recall that $h_1, h_2, \ldots, h_{|\mathbb{F}|}$ is an enumeration of elements in $\mathbb{F}$. Let $\mathsf{seed} = (R_2, R_3, \ldots, R_y)$, $i \in [3\ell \cdot |\mathbb{F}|]$, $j := \lfloor (i-1)/\ell \rfloor + 1$, and $j' := (i-1) \bmod \ell$. Then, the i-th query on seed probes the j'-th bit of $\Pi_{\mathsf{proof}}[\vec{a}_j]$, where $\vec{a}_j = (a_{j,1}, a_{j,2}, \ldots, a_{j,m})$. We define the *canonical neighbour* (seed_1, i_1) of (seed, i) as follows:

$$\mathsf{seed}_1 := (R^1_2 := a_{j,2}, R^1_3 := a_{j,3}, \ldots, R^1_m := a_{j,m}, R^1_y := 0) \tag{5.5}$$

$$i_1 := (j_1 - 1) \cdot \ell + j'_1 + 1 \tag{5.6}$$

where $j_1 \in [1, |\mathbb{F}|]$ such that $h_{j_1} = a_{j,1}$, and $j'_1 = j'$. It is easy to see that the canonical neighbours are the representative elements of the equivalence class induced by the neighbourhood relation. Denote $\mathcal{S}$ as the set of canonical neighbours, then $(\mathsf{seed}, i) \in \mathcal{S}$ if and only if $i \in [1, \ell|\mathbb{F}|]$ and $R_y = 0$.

To list all the neighbours of (seed, i), it suffices to find its canonical neighbour (seed_1, i_1) and list all the neighbours of (seed_1, i_1). Let $\mathsf{seed}_2 = (R^2_2, R^2_3, \ldots, R^2_m, R^2_y)$, $i_2 \in [3\ell \cdot |\mathbb{F}|]$, $j_2 := \lfloor (i_2 - 1)/\ell \rfloor + 1$, $j'_2 := (i_2 - 1) \bmod \ell$. Suppose that (seed_2, i_2) is a neighbour of (seed_1, i_1), then they represent the queries to the same bit of the same entry of Π_{proof}. This means that $j'_1 = j'_2$, and one of the following conditions holds:

$$j_2 \in [1, |\mathbb{F}|] \quad \text{and} \quad (h_{j_2}, R^2_2, R^2_3, \ldots, R^2_m) = (h_{j_1}, R^1_2, \ldots, R^1_{m-1}, R^1_m) \tag{5.7}$$

$$j_2 \in [|\mathbb{F}| + 1, 2|\mathbb{F}|] \quad \text{and} \quad (R^2_2, R^2_3, \ldots, R^2_m, h_{j_2 - |\mathbb{F}|}) = (h_{j_1}, R^1_2, \ldots, R^1_{m-1}, R^1_m); \tag{5.8}$$

$$j_2 \in [2|\mathbb{F}| + 1, 3|\mathbb{F}|] \quad \text{and} \quad (h \cdot y_1, R^2_2 + h \cdot y_2, \ldots, R^2_m + h \cdot y_m) = (h_{j_1}, R^1_2, \ldots, R^1_m)$$
$$\text{where } h := h_{j_2 - 2|\mathbb{F}|}, (y_1, y_2, \ldots, y_m) := S_\lambda[R^2_y]. \tag{5.9}$$

We will list the neighbours of (seed_1, i_1) in the following order: the $|S_\lambda|$ configurations satisfying (5.7) in the lexicographic order of R^2_y (note that this includes (seed_1, i_1)), the $|S_\lambda|$

[3] Note that the $(t+1)$-st bit of $\mathsf{bin}_{H^m}(\vec{a}^\star)$ is indeed located in $a^\star_m$, since $hm - (t+1) \leq (t+3) + m - (t+1) = m + 2 < h$ when t is large enough. Another minor detail is that if $c_1 = m$, then the test that $a^\star_m = 2^{t+1-(m-1)h}$ should be performed by V_{col} instead of V_{row}.

configurations satisfying (5.8) in the lexicographic order of R_y^2, and then the $|S_\lambda|$ configurations satisfying (5.9) in the lexicographic order of R_y^2.[4]

Rectangular neighbour listing of $\mathcal{S}$. Now we need to verify that the aforementioned listing of the neighbours satisfies the rectangular neighbour listing property (Definition 5.1.2). To start with, we consider the case when $(\mathsf{seed}, i) = (\mathsf{seed}_1, i_1)$, i.e., $(\mathsf{seed}, i) \in \mathcal{S}$. Recall that

$$\begin{aligned}\mathsf{seed}_1.\mathsf{col} &= (R_3^1, R_4^1, \ldots, R_{c_2-1}^1),\\ \mathsf{seed}_1.\mathsf{row} &= (R_{c_2+2}^1, R_{c_2+3}^1, \ldots, R_{m-1}^1),\\ \mathsf{seed}_1.\mathsf{shared} &= (R_2^1, R_{c_2}^1, R_{c_2+1}^1, R_m^1, R_y^1).\end{aligned}$$

Let $\mathsf{low}(\cdot)$ and $\mathsf{high}(\cdot)$ denote the lower and higher halves of a Boolean string, respectively. We partition $\mathsf{seed.shared} = (R_2^1, R_{c_2}^1, R_{c_2+1}^1, R_m^1, R_y^1)$ into two parts $(\mathsf{seed.shared.row}, \mathsf{seed.shared.col})$, where $\mathsf{seed.shared.col} := (R_2^1, R_{c_2}^1, \mathsf{low}(R_y^1))$ and $\mathsf{seed.shared.row} := (R_{c_2+1}^1, R_m^1, \mathsf{high}(R_y^1))$.

Given (seed_1, i_1), the algorithms A_{shared}, A_{col}, and A_{row} output $\mathsf{NList}(\mathsf{seed}_1, i_1)$ as follows:

1. First, consider the $|S_\lambda|$ neighbours (seed_2, i_2) satisfying (5.7), including (seed_1, i_1) itself. We can see that $i_2 = i_1$, $\mathsf{seed}_2.\mathsf{row} = \mathsf{seed}_1.\mathsf{row}$, $\mathsf{seed}_2.\mathsf{col} = \mathsf{seed}_1.\mathsf{col}$, and $\mathsf{seed}_2.\mathsf{shared} = (R_2^1, R_{c_2}^1, R_{c_2+1}^1, R_m^1, R_y^2)$, where R_y^2 enumerates over all $|S_\lambda|$ possibilities in lexicographic order. Clearly, i_2 only depends on i_1, $\mathsf{seed}_2.\mathsf{col}$ ($\mathsf{seed}_2.\mathsf{row}$ respectively) only depends on $\mathsf{seed.col}$ and can be computed by a projection over $\mathsf{seed.col}$ ($\mathsf{seed}_2.\mathsf{row}$ respectively).

2. Consider the $|S_\lambda|$ neighbours (seed_2, i_2) satisfying (5.8). We will enumerate all R_y^2 in lexicographic order and output:

$$\begin{aligned}\mathsf{seed}_2.\mathsf{col} &= (R_3^2 = R_2^1, \ldots, R_{c_2-1}^2 = R_{c_2-2}^1),\\ \mathsf{seed}_2.\mathsf{shared.col} &= (R_2^2 = h_{j_1}, R_{c_2}^2 = R_{c_2-1}^1, \mathsf{low}(R_y^2)),\\ \mathsf{seed}_2.\mathsf{row} &= (R_{c_2+2}^2 = R_{c_2+1}^1, \ldots, R_{m-1}^2 = R_{m-2}^1),\\ \mathsf{seed}_2.\mathsf{shared.row} &= (R_{c_2+1}^2 = R_{c_2}^1, R_m^2 = R_{m-1}^1, \mathsf{high}(R_y^2)).\end{aligned}$$

   Let $j_2 := \lfloor (i_2 - 1)/\ell \rfloor + 1$ and $j_2' := (i_2 - 1) \bmod \ell$, we can see that $j_2' = j_1'$ and $h_{j_2} = R_m^1$, hence i_2 can be computed from $\mathsf{seed.shared}$ and i_1 efficiently. Finally, it is easy to verify that $\mathsf{seed}_2.\mathsf{row}$ and $\mathsf{seed}_2.\mathsf{shared.row}$ only depend on $\mathsf{seed}_1.\mathsf{row}$ and $\mathsf{seed}_1.\mathsf{shared}$, and that $\mathsf{seed}_2.\mathsf{row}$ can be computed by a projection over $\mathsf{seed}_1.\mathsf{row}$. The same holds for the column randomness.

3. Finally, consider the $|S_\lambda|$ neighbours (seed_2, i_2) satisfying (5.9). We enumerate R_y^2 in lexicographic order. Let $(y_1, y_2, \ldots, y_m) = S_\lambda[R_y^2]$. Denote $h := y_1^{-1} \cdot h_{j_1}$, and let j_2 be the unique number in $[2|\mathbb{F}| + 1, 3|\mathbb{F}|]$ such that $h = h_{j_2 - 2|\mathbb{F}|}$, we output:

$$\begin{aligned}i_2 &= (j_2 - 1) \cdot \ell + j_1' + 1,\\ \mathsf{seed}_2.\mathsf{col} &= (R_3^2 = R_3^1 - h \cdot y_3, \ldots, R_{c_2-1}^2 = R_{c_2-1}^1 - h \cdot y_{c_2-1}),\end{aligned}$$

[4] Recall that for every $(y_1, y_2, \ldots, y_m) = S_\lambda[R_y^2]$, $y_1 \neq 0$. Thus for every R_y^2, there is exactly one $j_2 \in [2|\mathbb{F}| + 1, 3|\mathbb{F}|]$ that satisfies (5.9), namely the j_2 such that $h_{j_2 - 2|\mathbb{F}|} = y_1^{-1} \cdot h_{j_1}$.

$$\mathsf{seed}_2.\mathsf{shared.col} = (R^2_2 = R^1_2 - h \cdot y_2, R^2_{c_2} = R^1_{c_2} - h \cdot y_{c_2}, \mathsf{low}(R^2_y)),$$
$$\mathsf{seed}_2.\mathsf{row} = (R^2_{c_2+2} = R^1_{c_2+2} - h \cdot y_{c_2+2}, \ldots, R^2_{m-1} = R^1_{m-1} - h \cdot y_{m-1}),$$
$$\mathsf{seed}_2.\mathsf{shared.row} = (R^2_{c_2+1} = R^1_{c_2+1} - h \cdot y_{c_2+1}, R^2_m = R^1_m - h \cdot y_m, \mathsf{high}(R^2_y)).$$

Since j_2 and j'_1 only depends on i_1, i_2 also only depends on i_1 (and not on seed). Since addition over $\mathbb{F}$ is just bitwise XOR and $(h, y_1, \ldots, y_m)$ can be computed from $(\mathsf{seed}_1.\mathsf{shared}, i_1)$, we have that $\mathsf{seed}_2.\mathsf{col}$ ($\mathsf{seed}_2.\mathsf{row}$ respectively) can be computed by a projection over $\mathsf{seed}_1.\mathsf{col}$ ($\mathsf{seed}_1.\mathsf{row}$ respectively) computable from $(\mathsf{seed}_1.\mathsf{shared}, i_1)$.

It is clear that the "zipped" list of $\mathsf{NList}_{\mathsf{row}}$ and $\mathsf{NList}_{\mathsf{col}}$ is the list of neighbours of (seed_1, i_1). Since (seed_1, i_1) appears at the head of the list, A_{shared} simply outputs $\mathsf{self} = 1$.

We also point out the exact dependence of A_{shared}, A_{row}, and A_{col} on $\mathsf{seed.shared}$, as this will be useful later. Instead of the full $(\mathsf{seed.shared}, i_1)$, A_{shared} only needs to look at (R^1_m, i_1), A_{row} only needs to look at $(R^1_{c_2}, R^1_{c_2+1}, R^1_m, i_1)$, and A_{col} only needs to look at $(R^1_2, R^1_{c_2}, i_1)$.

Rectangular neighbour listing for $\overline{\mathcal{S}}$. For the general case when $(\mathsf{seed}, i) \notin \mathcal{S}$, we first find its canonical neighbour (seed_1, i_1) in a rectangular fashion. Let (seed_1, i_1) be the canonical neighbour of (seed, i), it suffices to show that:

- (R^1_m, i_1) can be computed from $(\mathsf{seed.shared}, i)$, so that we can feed it into the above A_{shared}.

  If $j \in [1, |\mathbb{F}|]$, then $R^1_m = R_m$ and i_1 only depends on $a_{j,1} = h_j$. If $j \in [|\mathbb{F}| + 1, 2|\mathbb{F}|]$, then $R^1_m = h_{j-|\mathbb{F}|}$ and i_1 only depends on $a_{j,1} = R_2$. If $j \in [2|\mathbb{F}| + 1, 3|\mathbb{F}|]$, then $R^1_m = R_m + y_m$ and i_1 only depends on $a_{j,1} = h_j + y_1$. It follows that (R^1_m, i_1) can be computed from (R_2, R_m, R_y, i), hence from $(\mathsf{seed.shared}, i)$.

- $(\mathsf{seed}_1.\mathsf{row}, R^1_{c_2}, R^1_{c_2+1}, R^1_m) = (R^1_{c_2}, \ldots, R^1_m)$ can be computed from $(\mathsf{seed.shared}, \mathsf{seed.row})$, so that we can feed it into the above A_{row}. Moreover, $\mathsf{seed}_1.\mathsf{row}$ can be computed by a projection over $\mathsf{seed.row}$ given $\mathsf{seed.shared}$.

  If $j \in [1, |\mathbb{F}|]$, then $R^1_i = R_i$ for every $c_2 \le i \le m$; if $j \in [|\mathbb{F}| + 1, 2|\mathbb{F}|]$, then $R^1_i = R_{i+1}$ for every $c_2 \le i < m$ and $R^1_m = h_{j-|\mathbb{F}|}$; if $j \in [2|\mathbb{F}| + 1, 3|\mathbb{F}|]$, then $R^1_i = R_i + y_i$ for every $c_2 \le i \le m$. In all of the three cases above, $(R^1_{c_2}, \ldots, R^1_m)$ can be computed from $(R_{c_2}, \ldots, R_m, R_y)$, hence from $(\mathsf{seed.shared}, \mathsf{seed.row})$; moreover the computation of $\mathsf{seed}_1.\mathsf{row}$ is indeed a projection over $\mathsf{seed.row}$.

- $(\mathsf{seed}_1.\mathsf{col}, R^1_2, R^1_{c_2}) = (R^1_2, \ldots, R^1_{c_2})$ can be computed from $(\mathsf{seed.shared}, \mathsf{seed.col})$, so that we can feed it into the above A_{col}. Moreover, $\mathsf{seed}_1.\mathsf{col}$ can be computed by a projection over $\mathsf{seed.col}$ given $\mathsf{seed.shared}$.

  If $j \in [1, |\mathbb{F}|]$, then $R^1_i = R_i$ for every $2 \le i \le c_2$; if $j \in [|\mathbb{F}| + 1, 2|\mathbb{F}|]$, then $R^1_i = R_{i+1}$ for every $2 \le i \le c_2$; if $j \in [2|\mathbb{F}| + 1, 3|\mathbb{F}|]$, then $R^1_i = R_i + y_i$ for every $2 \le i \le c_2$. In all of the three cases above, $(R^1_2, \ldots, R^1_{c_2})$ can be computed from $(R_2, \ldots, R_{c_2+1}, R_y)$, hence from $(\mathsf{seed}, \mathsf{shared}, \mathsf{seed.col})$; moreover the computation of $\mathsf{seed}_1.\mathsf{col}$ is indeed a projection over $\mathsf{seed.col}$.

Then, we can feed seed_1 into the procedures for rectangular neighbour listing of $\mathcal{S}$ and obtain rectangular neighbour listing of $\overline{\mathcal{S}}$.

Finally, $A_{\mathsf{shared}}(\mathsf{seed.shared})$ needs to compute self, i.e., the position of (seed, i) in the list. The list contains three parts: neighbours specified by (5.7), (5.8), and (5.9). The part that (seed, i) belongs to only depends on $\lfloor (j-1)/|\mathbb{F}| \rfloor + 1 \in \{1,2,3\}$. The exact position of (seed, i) in the corresponding part depends on the lexicographic order of R_y.

Complexity of A_{shared}, A_{row}, and A_{col}. Recall that $\log|\mathbb{F}| = f = (\log T(n) + 3)/m + O(\log\log T(n))$, $|S_\lambda| = O((mf/\lambda)^2) = \mathrm{poly}(m^m, \log T(n))$. It is then easy to check that the running times of A_{shared}, A_{row}, and A_{col} are $\mathrm{poly}(m^{O(m)}, f, \log T(n)) = \mathrm{poly}(m^m, \log T(n))$.

Robust Soundness

The PCPP described above only guarantees an *expected* version of robust soundness: the expected fraction of bits that we need to flip in order to make the verifier accept is at least ρ, where the expectation is over the choice of seed. (See [BGH+06, Lemma 8.11].) Here we use a Markov bound to turn this into a standard robust soundness property, but only with soundness error very close to 1. Since the robust soundness error reduction (Section 5.1.5) preserves smoothness but does not seem to preserve RNL, we do not apply it here.

Let $\delta_{\mathsf{proof}}(\mathsf{seed})$ (resp. $\delta_{\mathsf{input}}(\mathsf{seed})$) be the fraction of bits of Π_{proof} (resp. Π_{input}) read by the verifier that we need to flip to make the verifier accept given the randomness seed. Let $\hat{\delta}(\mathsf{seed})$ be the fraction of bits (of both Π_{proof} and Π_{input}) read by the verifier that we need to flip to make the verifier accept given the randomness seed. By [BGH+06, Lemma 8.11],[5] there is a constant $\rho_0 \in (0,1)$ such that for every constant $\delta \in (0,1)$, if Π_{input} is δ-far from being in L, then for any proof oracle Π_{proof}, either $\mathbb{E}_{\mathsf{seed}}[\delta_{\mathsf{proof}}(\mathsf{seed})] \geq \rho_0$ or $\mathbb{E}_{\mathsf{seed}}[\delta_{\mathsf{input}}(\mathsf{seed})] \geq \delta/3$.

Recall that the verifier makes $|\mathbb{F}|$ queries to Π_{input} and $3\ell\cdot|\mathbb{F}|$ queries to Π_{proof}. We repeat each query to the input oracle for $9(\rho_0/\delta)\ell$ times. If Π_{input} is δ-far from being in L, the fraction of bits read by the verifier that we need to flip on expectation to make the verifier accept is

$$\mathop{\mathbb{E}}_{\mathsf{seed}}\left[\hat{\delta}(\mathsf{seed})\right] = \frac{\min\{\rho_0\cdot 3\ell\cdot|\mathbb{F}|, (\delta/3)\cdot 9(\rho_0/\delta)\ell\cdot|\mathbb{F}|\}}{3\ell\cdot|\mathbb{F}| + 9(\rho_0/\delta)\ell\cdot|\mathbb{F}|} \geq \frac{3\ell\rho_0}{3\ell + 9(\rho_0/\delta)\ell} \geq \frac{\rho_0}{1+3\rho_0/\delta}.$$

Let $\rho := \frac{\rho_0}{2+6\rho_0/\delta}$. By a Markov bound,

$$\Pr_{\mathsf{seed}}\left[\hat{\delta}(\mathsf{seed}) \leq \rho\right] \leq 1-\rho.$$

Thus, the PCPP verifier has robust soundness error $1-\rho$ with robustness parameter ρ and proximity parameter δ.

Proof of Lemma 5.1.4

Finally, we prove Lemma 5.1.4. For completeness, we define λ-biased sets:

Definition 5.1.8. For two strings $x, y \in \{0,1\}^n$, denote their inner product as $\mathrm{IP}(x,y) := (-1)^{\sum_{i=1}^n x_i y_i}$.

[5] Recall that we assume $m \leq (\log T(n))^{0.1}$ and set $\lambda = \min\{1/(ct), 1/m^{2cm}\}$, so that $m^m \leq T(n)^{1/m^2}$ and $\lambda \leq \min\{1/(ct), 1/m^{cm}\}$, which satisfies the technical requirement of [BGH+06, Lemma 8.11].

Let $\lambda > 0$. A set $S \subseteq \{0,1\}^n$ is a *λ-biased set* if for every string $y \in \{0,1\}^n$, if y is not the all-zero string, then

$$\left|\sum_{x \in S} \mathrm{IP}(x,y)\right| \leq \lambda |S|$$

Note that when $\mathbb{F} = \mathrm{GF}(2^q)$, the definition of λ-biased subsets of $\mathbb{F}^m$ in [BSVW03, Section 2] coincides with the above definition (of λ-biased subsets of $\{0,1\}^{mq}$).

We use the explicit construction of a λ-biased subset $S_\lambda \subseteq \mathbb{F}^m$ of size $O((qm/\lambda)^2)$ in [AGHP92]. Recall that we also need the first coordinate of every element in S_λ to be nonzero. Therefore, we first construct a $(\lambda/4)$-biased set $S_{\lambda/4} \subseteq \mathbb{F}^m$ using [AGHP92], and then remove every element $\vec{y} \in S_{\lambda/4}$ with $y_1 = 0$ to obtain the λ-biased set S_λ. In what follows, we prove that this remaining set is indeed a λ-biased set.

Lemma 5.1.4. *Let $\lambda < 0.1$, q, m be integers such that $q \geq \log \frac{4}{\lambda}$, and let $\mathbb{F} = \mathrm{GF}(2^q)$. There is a deterministic polynomial-time algorithm that on input $(1^m, 1^q, 1^{\lceil 1/\lambda \rceil})$, outputs a λ-biased set $S_\lambda \subseteq (\mathbb{F} \setminus \{0\}) \times \mathbb{F}^{m-1}$ of size $O((qm/\lambda)^2)$.*

Proof. We first use [AGHP92] to construct a $\lambda/4$-biased set $S_{\lambda/4} \subseteq \mathbb{F}^m$ of size $O((qm/\lambda)^2)$. Let $S_0 := \{y \in S_{\lambda/4} : y_1 = 0\}$ and $S_\lambda := S_{\lambda/4} \setminus S_0$, we will show that S_λ is a λ-biased set.

First, we show that $|S_0| \leq (\lambda/2)|S_{\lambda/4}|$, i.e., we only removed a small fraction of elements. Let $\mathcal{Y} := \{0,1\}^q \times \{0^{(m-1)q}\}$ be the set of length-(mq) strings that is zero on all but the first q input bits; each $y \in \mathcal{Y}$ corresponds to a linear test $\mathrm{IP}(\cdot, y)$ that *only depends on the first q input bits*. Abusing notation, we also use $\mathcal{Y}$ to denote the uniform distribution over $\mathcal{Y}$. For every $x \in S_{\lambda/4}$, if $x \in S_0$ then $\mathrm{IP}(x,y) = 1$ for every $y \in \mathcal{Y}$; otherwise the expectation of $\mathrm{IP}(x,y)$ over a random $y \leftarrow \mathcal{Y}$ is zero. It follows that

$$|S_0|/|S_{\lambda/4}| = \mathop{\mathbb{E}}_{x \leftarrow S_{\lambda/4}, y \leftarrow \mathcal{Y}}[\mathrm{IP}(x,y)].$$

On the other hand, if $y = 0^{mq}$ then $\mathrm{IP}(x,y) = 1$ for every possible x, while if $y \neq 0^{mq}$ then the expectation of $\mathrm{IP}(x,y)$ over a random $x \leftarrow S_{\lambda/4}$ is between $-\lambda/4$ and $\lambda/4$. Therefore

$$\mathop{\mathbb{E}}_{x \leftarrow S_{\lambda/4}, y \leftarrow \mathcal{Y}}[\mathrm{IP}(x,y)] \leq \frac{1}{2^q} + \lambda/4 \leq \lambda/2.$$

Now we show that S_λ is a λ-biased set. Fix any binary string $y \in \{0,1\}^{mq} \setminus \{0^{mq}\}$, we have

$$\begin{aligned}\left|\sum_{x \in S_\lambda} \mathrm{IP}(x,y)\right| &\leq \left|\sum_{x \in S_{\lambda/4}} \mathrm{IP}(x,y)\right| + \left|\sum_{x \in S_0} \mathrm{IP}(x,y)\right| \\ &\leq (\lambda/4)|S_{\lambda/4}| + |S_0| \\ &\leq (3\lambda/4)|S_{\lambda/4}|.\end{aligned}$$

On the other hand, we have $|S_\lambda| \geq (1 - \lambda/2)|S_{\lambda/4}| \geq 0.95|S_{\lambda/4}|$. Therefore

$$\left|\mathop{\mathbb{E}}_{x \leftarrow S_\lambda}[\mathrm{IP}(x,y)]\right| \leq \frac{3\lambda}{4 \cdot 0.95} \leq \lambda.$$

□

5.1.3 RNL-Preserving Composition Theorem

The PCPP verifier in Section 5.1.2 requires roughly $T(n)^{1/m}$ query complexity. In this subsection, we compose it with an efficient inner PCPP to reduce its query complexity to $O(1)$ while preserving the RNL property. We assume familiarity with the composition theorem of PCPs: a good reference for the basic composition theorem is [BGH+06, Section 2.4], and a composition theorem that preserves RNL is proved in [BHPT24, Section 7].

Let Circuit-Eval$^{\perp}$ denote the circuit value problem where the input alphabet of the circuit is $\{0, 1, \perp\}$. Note that the input alphabet of the decision circuit of the PCPP constructed in Section 5.1.2 is $\{0, 1, \perp\}$ instead of $\{0, 1\}$. Therefore, we should use a PCPP for Circuit-Eval$^{\perp}$ instead of Circuit-Eval for the inner PCPP.

The main result of this subsection is the following composition theorem.

Theorem 5.1.9. *Let $n \le T(n) \le 2^{\mathrm{poly}(n)}$. Suppose that $\mathsf{NTIME}[T(n)]$ has a robust and rectangular PCPP verifier V^{out} and Circuit-Eval$^{\perp}$ has a (not necessarily rectangular) PCPP verifier V^{in} with parameters specified in Table 5.2. Moreover, assume that $q^{\mathsf{in}} = O(1)$, $\rho^{\mathsf{out}} \ge \delta^{\mathsf{in}}$, $\ell^{\mathsf{in}} = 2^{r^{\mathsf{in}}}$,[6] $r^{\mathsf{out}}_{\mathsf{col}} \le \log W^{\mathsf{out}}_{\mathsf{proof}} \le r^{\mathsf{out}}_{\mathsf{col}} + r^{\mathsf{out}}_{\mathsf{shared}}$, $H^{\mathsf{out}}_{\mathsf{proof}} \cdot W^{\mathsf{out}}_{\mathsf{proof}} \le 2^{r^{\mathsf{out}}+r^{\mathsf{in}}}$, and $H^{\mathsf{out}}_{\mathsf{proof}}$, $W^{\mathsf{out}}_{\mathsf{proof}}$ are powers of 2. Then $\mathsf{NTIME}[T(n)]$ has a rectangular PCPP verifier V^{comp} with parameters specified in Table 5.2. Moreover:*

- *If the query indices of V^{out} are computable by projections, then the query indices of V^{comp} are computable by projections as well.*
- *If V^{out} has $t^{\mathsf{out}}_{\mathsf{RNL}}(n)$-time rectangular neighbour listing property, then V^{comp} has $t_{\mathsf{RNL}}(n)$-time rectangular neighbour listing property, where $t_{\mathsf{RNL}}(n) := \mathrm{poly}(t^{\mathsf{out}}_{\mathsf{RNL}}(n), \ell^{\mathsf{in}}, d^{\mathsf{in}})$. Furthermore, if the RNL for V^{out} can be computed by projections, then the RNL for V^{comp} can be computed by projections as well.*

The composed PCPP verifier. Assume that we have a robust and rectangular PCPP verifier V^{out} for $L \in \mathsf{NTIME}[T(n)]$ and a PCPP verifier V^{in} for Circuit-Eval$^{\perp}$. We now describe the composed PCPP verifier V^{comp} for L. In a nutshell, we reduce the verification of the outer PCPP V^{out} to Circuit-Eval$^{\perp}$, where the circuit represents the decision predicate of V^{out} and the input consists of the input of L and the proofs for the outer PCPP. As in [BHPT24], we need to carefully arrange the proof matrix to maintain rectangularity.

Assume that $\Pi^{\mathsf{out}}_{\mathsf{input}}$, $\Pi^{\mathsf{out}}_{\mathsf{proof}}$, and $\mathsf{seed}^{\mathsf{out}}$ are the input matrix, the proof matrix, and the random seed of V^{out}, respectively. The input matrix of V^{comp} is exactly the input matrix of V^{out}, denoted as Π_{input}. The proof of V^{comp} is the concatenation of $\Pi^{\mathsf{out}}_{\mathsf{proof}}$ and $\Pi^{\mathsf{in}}_{\mathsf{proof}}(\mathsf{seed}^{\mathsf{out}})$ for every $\mathsf{seed}^{\mathsf{out}} \in \{0,1\}^{r^{\mathsf{out}}}$, where each $\Pi^{\mathsf{in}}_{\mathsf{proof}}(\mathsf{seed}^{\mathsf{out}})$ is a PCPP proof for "V^{in} accepts $\mathsf{seed}^{\mathsf{out}}$." (In fact, to make the query indices of V^{comp} computable by projections, we need to arrange the inner PCPP proofs *before* the outer PCPP proof.) The random seed is $\mathsf{seed} := \mathsf{seed}^{\mathsf{out}} \circ \mathsf{seed}^{\mathsf{in}}$. The verifier V^{comp} works as follows:

[6] This is without loss of generality, because $\ell^{\mathsf{in}} \le 2^{r^{\mathsf{in}}} \cdot q^{\mathsf{in}}$, and in our case q^{in} will be a constant. We could always add $O(\log q^{\mathsf{in}}) = O(1)$ dummy bits to the inner verifier's randomness and pad the inner verifier's proof oracle to length $2^{r^{\mathsf{in}}}$.

Verifier	V^{out}	V^{in}	V^{comp}
Soundness error	$1-\varepsilon^{\mathsf{out}}$	$1-\varepsilon^{\mathsf{in}}$	$1-\varepsilon^{\mathsf{out}}\cdot\varepsilon^{\mathsf{in}}$
Proximity parameter	δ^{out}	δ^{in}	δ^{out}
Robustness parameter	ρ^{out}	-	-
Row randomness	$r^{\mathsf{out}}_{\mathsf{row}}$	-	$r^{\mathsf{out}}_{\mathsf{row}}$
Column randomness	$r^{\mathsf{out}}_{\mathsf{col}}$	-	$r^{\mathsf{out}}_{\mathsf{col}}$
Shared randomness	$r^{\mathsf{out}}_{\mathsf{shared}}$	r^{in}	$r^{\mathsf{out}}_{\mathsf{shared}}+r^{\mathsf{in}}$
Proof matrix height	$H^{\mathsf{out}}_{\mathsf{proof}}$	-	$H^{\mathsf{out}}_{\mathsf{proof}}+2^{r^{\mathsf{out}}+r^{\mathsf{in}}}/W^{\mathsf{out}}_{\mathsf{proof}}$
Proof matrix width	$W^{\mathsf{out}}_{\mathsf{proof}}$	-	$W^{\mathsf{out}}_{\mathsf{proof}}$
Query complexity	q^{out}	q^{in}	q^{in}
Parity check complexity	-	-	q^{in}
Decision complexity	d^{out}	d^{in}	d^{in}
Proof length	ℓ^{out}	ℓ^{in}	$H^{\mathsf{out}}_{\mathsf{proof}}\cdot W^{\mathsf{out}}_{\mathsf{proof}}+2^{r^{\mathsf{out}}+r^{\mathsf{in}}}$

Table 5.2: The parameters of the PCPPs in the composition theorem. Here $r^{\mathsf{out}} := r^{\mathsf{out}}_{\mathsf{row}}+r^{\mathsf{out}}_{\mathsf{col}}+r^{\mathsf{out}}_{\mathsf{shared}}$. Note that the input length of the inner PCPP is $d^{\mathsf{out}}=d^{\mathsf{out}}(n)$, e.g., r^{in} in the table actually refers to $r^{\mathsf{in}}(d^{\mathsf{out}}(n))$.

1. Obtain the decision circuit $\mathsf{Dec}^{\mathsf{out}}$ and the list of query indices $I^{\mathsf{out}} \leftarrow V^{\mathsf{out}}(\mathsf{seed}^{\mathsf{out}})$ of V^{out}.

2. Use the inner PCPP to verify the following Circuit-Eval$^{\perp}$ instance $(C, \Pi^{\mathsf{in}}_{\mathsf{input}}(\mathsf{seed}^{\mathsf{out}}))$: the (explicit) circuit $C : \{0,1,\perp\}^{q^{\mathsf{out}}(n)} \times \{0,1\}^{\hat{r}^{\mathsf{out}}} \to \{0,1,\perp\}$ and the (implicit) input $\Pi^{\mathsf{in}}_{\mathsf{input}}(\mathsf{seed}^{\mathsf{out}}) \in \{0,1,\perp\}^{q^{\mathsf{out}}(n)} \times \{0,1\}^{\hat{r}^{\mathsf{out}}}$ are defined as follows:

$$C(u,v) := \mathsf{Dec}^{\mathsf{out}}(u, \mathsf{Decode}(v)),$$
$$\Pi^{\mathsf{in}}_{\mathsf{input}}(\mathsf{seed}^{\mathsf{out}}) := ((\Pi_{\mathsf{input}} \circ \Pi^{\mathsf{out}}_{\mathsf{proof}})|_{I^{\mathsf{out}}}, \mathsf{Encode}(\mathsf{seed}^{\mathsf{out}})),$$

   where $(\mathsf{Encode}, \mathsf{Decode})$ is a linear-time encodable and decodable error-correcting code such that $\mathsf{Encode} : \{0,1\}^{r^{\mathsf{out}}} \to \{0,1\}^{\hat{r}^{\mathsf{out}}}$ is linear over GF(2).[7] Specifically:

   (a) The decision circuit $\mathsf{Dec}^{\mathsf{comp}}$ of V^{comp} is the same as the decision circuit $\mathsf{Dec}^{\mathsf{in}}$ of V^{in}.

   (b) The queries are sampled using $V^{\mathsf{in}}(\mathsf{seed}^{\mathsf{in}})$ for the Circuit-Eval instance defined above with the proof $\Pi^{\mathsf{in}}_{\mathsf{proof}}(\mathsf{seed}^{\mathsf{out}})$, i.e., we sample the queries $I^{\mathsf{in}} \leftarrow V^{\mathsf{in}}(\mathsf{seed}^{\mathsf{in}})$ and "redirect" them to the input oracle and the proof of the composed PCPP to obtain I^{comp}.[8]

Rectangularity of V^{comp}. We now verify the rectangularity of the composed PCPP verifier. Recall that: the proof $\Pi^{\mathsf{out}}_{\mathsf{proof}} \in \{0,1\}^{\ell^{\mathsf{out}}}$ of V^{out} is arranged as an $H^{\mathsf{out}}_{\mathsf{proof}} \times W^{\mathsf{out}}_{\mathsf{proof}}$ matrix, where the i-th row and the j-th column $\Pi^{\mathsf{out}}_{\mathsf{proof}}[i,j] := \Pi^{\mathsf{out}}_{\mathsf{proof}}[(i-1)\cdot W^{\mathsf{out}}_{\mathsf{proof}} + j]$; the inner proofs $\Pi^{\mathsf{in}}_{\mathsf{proof}}(\mathsf{seed}^{\mathsf{out}})$ are of length ℓ^{in} for every $\mathsf{seed}^{\mathsf{out}} \in \{0,1\}^{r^{\mathsf{out}}}$.

[7] We apply an error-correcting code on the randomness because we want $\mathsf{Dec}^{\mathsf{out}}$ to have *robust* soundness. Let $(\Pi', \mathsf{Encode}(\mathsf{seed}^{\mathsf{out}}))$ be the input of $\mathsf{Dec}^{\mathsf{out}}$, if given $\mathsf{seed}^{\mathsf{out}}$, Π' is far from being accepted by $\mathsf{Dec}^{\mathsf{out}}$, then $(\Pi', \mathsf{Encode}(\mathsf{seed}^{\mathsf{out}}))$ is also far from being accepted by $\mathsf{Dec}^{\mathsf{out}}$. This is not true if we do not encode $\mathsf{seed}^{\mathsf{out}}$.

[8] In fact, there are three kinds of queries: the queries to $(\Pi_{\mathsf{input}} \circ \Pi^{\mathsf{out}}_{\mathsf{proof}})|_{I^{\mathsf{out}}}$, $\mathsf{Enc}(\mathsf{seed}^{\mathsf{out}})$, and $\Pi^{\mathsf{in}}_{\mathsf{proof}}(\mathsf{seed}^{\mathsf{out}})$. The queries of the first and the third kinds will be redirected as queries, and the second kind will be treated as a parity-check bit, since Enc is a linear function over GF(2). More details can be found in the verification of the rectangularity later.

Let $W^{\mathsf{in}}_{\mathsf{proof}} := \min\{W^{\mathsf{out}}_{\mathsf{proof}}/2^{r^{\mathsf{out}}_{\mathsf{col}}}, \ell^{\mathsf{in}}\}$ and $H^{\mathsf{in}}_{\mathsf{proof}} := \ell^{\mathsf{in}}/W^{\mathsf{in}}_{\mathsf{proof}}$. Note that both $W^{\mathsf{in}}_{\mathsf{proof}}$ and $H^{\mathsf{in}}_{\mathsf{proof}}$ are powers of 2 and $W^{\mathsf{out}}_{\mathsf{proof}} \geq 2^{r^{\mathsf{out}}_{\mathsf{col}}}$. We arrange the proof matrix $\Pi^{\mathsf{comp}}_{\mathsf{proof}}$ of V^{comp} as follows: First, the proof matrix contains a proof of the inner PCPP $\Pi^{\mathsf{in}}_{\mathsf{proof}}(\mathsf{seed}^{\mathsf{out}})$ for each $\mathsf{seed}^{\mathsf{out}} \in \{0,1\}^{\ell^{\mathsf{in}}}$, sorted in lexicographic order of $\mathsf{seed}^{\mathsf{out}}$. Each such inner proof is a rectangle of size $W^{\mathsf{in}}_{\mathsf{proof}} \times H^{\mathsf{in}}_{\mathsf{proof}}$. Then we append the $H^{\mathsf{out}}_{\mathsf{proof}} \times W^{\mathsf{out}}_{\mathsf{proof}}$ proof matrix of the outer PCPP.

Clearly, the proof matrix height of the composed PCPP verifier is

$$\ell^{\mathsf{in}} \cdot 2^{r^{\mathsf{out}}}/W^{\mathsf{out}}_{\mathsf{proof}} + H^{\mathsf{out}}_{\mathsf{proof}} = 2^{r^{\mathsf{out}}+r^{\mathsf{in}}}/W^{\mathsf{out}}_{\mathsf{proof}} + H^{\mathsf{out}}_{\mathsf{proof}}.$$

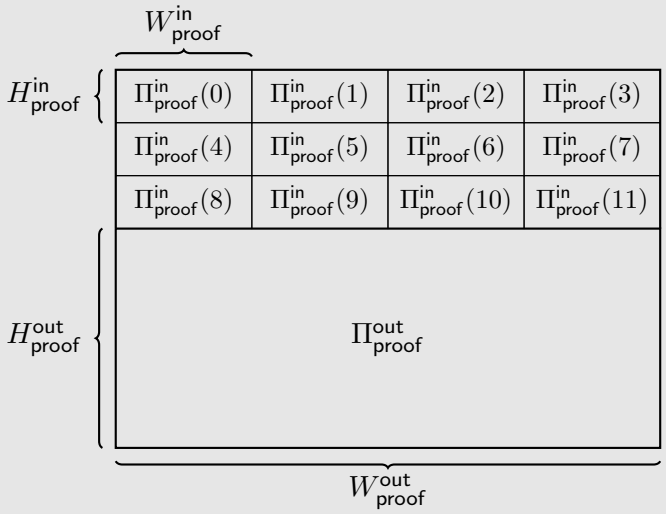


Figure 5.5: The layout of the proof matrix Π^{comp} of the composed PCPP.

Recall that the seed of the composed PCPP verifier is $\mathsf{seed} := (\mathsf{seed}^{\mathsf{out}}, \mathsf{seed}^{\mathsf{in}})$. Assume that the partition of random bits of V^{out} is $\mathsf{seed}^{\mathsf{out}} = (\mathsf{seed}^{\mathsf{out}}.\mathsf{row}, \mathsf{seed}^{\mathsf{out}}.\mathsf{col}, \mathsf{seed}^{\mathsf{out}}.\mathsf{shared})$. We partition the random bits as follows:

$$\begin{aligned}
\mathsf{seed}.\mathsf{shared} &:= (\mathsf{seed}^{\mathsf{out}}.\mathsf{shared}, \mathsf{seed}^{\mathsf{in}}).\\
\mathsf{seed}.\mathsf{row} &:= \mathsf{seed}^{\mathsf{out}}.\mathsf{row}.\\
\mathsf{seed}.\mathsf{col} &:= \mathsf{seed}^{\mathsf{out}}.\mathsf{col}.
\end{aligned}$$

Now we describe the type predicate V_{type} and rectangular verifiers $V^{\mathsf{comp}}_{\mathsf{row}}$ and $V^{\mathsf{comp}}_{\mathsf{col}}$ of the composed PCPP.

The type predicate and the row/column verifier firstly obtain $\mathsf{seed}^{\mathsf{in}}$ in $\mathsf{seed}.\mathsf{shared}$ and compute the queries $I^{\mathsf{in}} \leftarrow V^{\mathsf{in}}(\mathsf{seed}^{\mathsf{in}})$ of the inner PCPP for Circuit-Eval$^{\perp}$. There can be three cases for each query in I^{in}:

1. It probes the i-th cell of $\Pi^{\mathsf{in}}_{\mathsf{input}}(\mathsf{seed}^{\mathsf{out}})$ and $i \leq q^{\mathsf{out}}(n)$, i.e., it queries $(\Pi_{\mathsf{input}} \circ \Pi^{\mathsf{out}}_{\mathsf{proof}})|_{I^{\mathsf{out}}}$.

   - The type predicate invokes the type predicate of V^{out} to compute the type of the query, since it has $\mathsf{seed}^{\mathsf{out}}.\mathsf{shared}$ and the index of the query in hand.
   - The row/column verifier of the composed PCPP runs the row/column verifier of the outer PCPP to obtain the row/column index of the query in $(\Pi_{\mathsf{input}}, \Pi^{\mathsf{out}}_{\mathsf{proof}})$. If it is

a query to the input matrix, then the row/column index of the composed verifier is equal to the row/column index of the outer verifier. If it is a query to the $(i^{\mathsf{out}}_{\mathsf{row}}, i^{\mathsf{out}}_{\mathsf{col}})$-th entry to the proof matrix Π^{out}, then it is the $(i^{\mathsf{out}}_{\mathsf{row}} + 2^{r^{\mathsf{out}}+r^{\mathsf{in}}}/W^{\mathsf{out}}_{\mathsf{proof}}, i^{\mathsf{out}}_{\mathsf{col}})$-th entry to Π^{comp}. Clearly, after knowing $\mathsf{seed.shared}$, $V^{\mathsf{comp}}_{\mathsf{row}}$ (resp. $V^{\mathsf{comp}}_{\mathsf{col}}$) can compute the row index (resp. column index) of the query to Π^{comp} given $\mathsf{seed.row}$ (resp. $\mathsf{seed.col}$).

- Furthermore, since $2^{r^{\mathsf{out}}+r^{\mathsf{in}}}/W^{\mathsf{out}}_{\mathsf{proof}} \geq H^{\mathsf{out}}_{\mathsf{proof}} > i^{\mathsf{out}}_{\mathsf{row}}$ and $2^{r^{\mathsf{out}}+r^{\mathsf{in}}}/W^{\mathsf{out}}_{\mathsf{proof}}$ is a power of 2, it follows that the new row index $(i^{\mathsf{out}}_{\mathsf{row}} + 2^{r^{\mathsf{out}}+r^{\mathsf{in}}}/W^{\mathsf{out}}_{\mathsf{proof}})$ is a projection over the old row index $(i^{\mathsf{out}}_{\mathsf{row}})$. The new column index is equal to the old column index. Hence, if the query indices of V^{out} are computable by projections, then so are the new row/column indices in this case.

2. It probes the i-th cell of $\Pi^{\mathsf{in}}_{\mathsf{input}}(\mathsf{seed}^{\mathsf{out}})$ and $i > q^{\mathsf{out}}(n)$, i.e., it queries $\mathsf{Enc}(\mathsf{seed}^{\mathsf{out}})$. Instead of making a query, we fix this input of $\mathsf{Dec}^{\mathsf{comp}}$ to be $\mathsf{Enc}(\mathsf{seed}^{\mathsf{out}})[i - q^{\mathsf{out}}(n)]$. This bit will be considered as a parity-check bit.

3. It probes the i-th cell of $\Pi^{\mathsf{in}}_{\mathsf{proof}}(\mathsf{seed}^{\mathsf{out}})$. This is a query to the proof, so the type predicate always outputs proof. Recall that $\Pi^{\mathsf{in}}_{\mathsf{proof}}(\mathsf{seed}^{\mathsf{out}})$ is placed in some $W^{\mathsf{in}}_{\mathsf{proof}} \times H^{\mathsf{in}}_{\mathsf{proof}}$ size block in the proof matrix. Let $N_{\mathsf{proof}} := W^{\mathsf{out}}_{\mathsf{proof}}/W^{\mathsf{in}}_{\mathsf{proof}}$, i.e., there are N_{proof} blocks of inner PCPP proofs in a row of the proof matrix. Then we probe the cell at the i_{row}-th row and i_{col}-th column, where

$$i_{\mathsf{row}} = \lfloor \mathsf{seed}^{\mathsf{out}}/N_{\mathsf{proof}} \rfloor \cdot H^{\mathsf{in}}_{\mathsf{proof}} + \lfloor i/W^{\mathsf{in}}_{\mathsf{proof}} \rfloor \qquad \text{and}$$
$$i_{\mathsf{col}} = (\mathsf{seed}^{\mathsf{out}} \bmod N_{\mathsf{proof}}) \cdot W^{\mathsf{in}}_{\mathsf{proof}} + (i \bmod W^{\mathsf{in}}_{\mathsf{proof}}).$$

It is easy to see that the column (resp. row) index of the query depends on $\mathsf{seed}^{\mathsf{in}}$ and the lowest $\log N_{\mathsf{proof}}$ bits (resp. the highest $r^{\mathsf{out}} - \log N_{\mathsf{proof}}$ bits) of $\mathsf{seed}^{\mathsf{out}}$. Note that since

$$\log N_{\mathsf{proof}} \geq \log\left(W^{\mathsf{out}}_{\mathsf{proof}}/(W^{\mathsf{out}}_{\mathsf{proof}}/2^{r^{\mathsf{out}}_{\mathsf{col}}})\right) \geq |\mathsf{seed}^{\mathsf{out}}.\mathsf{col}|,$$
$$\log N_{\mathsf{proof}} \leq \log W^{\mathsf{out}}_{\mathsf{proof}} \leq |\mathsf{seed}^{\mathsf{out}}.\mathsf{col}| + |\mathsf{seed}^{\mathsf{out}}.\mathsf{shared}|,$$

we can arrange

$$\mathsf{seed}^{\mathsf{out}} := \mathsf{seed}^{\mathsf{out}}.\mathsf{col} \circ \mathsf{seed}^{\mathsf{out}}.\mathsf{shared} \circ \mathsf{seed}^{\mathsf{out}}.\mathsf{row}$$

such that the lowest $\log N_{\mathsf{proof}}$ bits (resp. the highest $r^{\mathsf{out}} - \log N_{\mathsf{proof}}$ bits) can be computed by a projection over $\mathsf{seed}^{\mathsf{out}}.\mathsf{col}$ (resp. $\mathsf{seed}^{\mathsf{out}}.\mathsf{row}$) given $\mathsf{seed}^{\mathsf{out}}.\mathsf{shared}$.

Parity-check complexity. How does the decision predicate of V^{comp} depend on its randomness? Note that $\mathsf{Dec}^{\mathsf{comp}}$ is equal to $\mathsf{Dec}^{\mathsf{in}}$ except that in Item 2 above, we fix a certain input bit of $\mathsf{Dec}^{\mathsf{comp}}$ to be a certain bit in $\mathsf{Encode}(\mathsf{seed}^{\mathsf{out}})$. Since Encode is a $\mathrm{GF}(2)$-*linear* error correcting code (Theorem 2.4.1), each bit of $\mathsf{Encode}(\mathsf{seed}^{\mathsf{out}})$ is the XOR of a subset of indices in $\mathsf{seed}^{\mathsf{out}}$. (This is the reason that we need *parity-check* bits in ROP.) Also, $\mathsf{Dec}^{\mathsf{in}}$ only depends on $\mathsf{seed.shared}$, therefore $\mathsf{Dec}^{\mathsf{comp}}$ only depends on $\mathsf{seed.shared}$ and the q^{in} parity-check bits over $\mathsf{seed.row}$ and $\mathsf{seed.col}$.

Rectangular neighbour listing of V^{comp}. Now we verify that the composed PCPP verifier V^{comp} has the rectangular neighbour listing property with $t_{\mathsf{RNL}}(n) := \mathrm{poly}(t^{\mathsf{out}}_{\mathsf{RNL}}(n), \ell^{\mathsf{in}}, d^{\mathsf{in}})$. Moreover, we show that if the RNL for V^{out} can be computed by projections, then the RNL for V^{comp} can be computed by projections as well.

Let (seed, k) be a configuration of V^{comp}, where

$$\mathsf{seed} = (\mathsf{seed.row} := \mathsf{seed}^{\mathsf{out}}.\mathsf{row}, \mathsf{seed.col} := \mathsf{seed}^{\mathsf{out}}.\mathsf{col}, \mathsf{seed.shared} := (\mathsf{seed}^{\mathsf{out}}.\mathsf{shared}, \mathsf{seed}^{\mathsf{in}}))$$

and $k \in [q^{\mathsf{comp}}]$. Assume that the verifier probes the proof matrix $\Pi^{\mathsf{comp}}_{\mathsf{proof}}$ on the k-th query given the randomness seed. By the discussion above, we know that the k-th query of the composed PCPP verifier can be one of the following two cases: a query to $\Pi^{\mathsf{out}}_{\mathsf{proof}}|_{I_{\mathsf{out}}}$ for $I_{\mathsf{out}} \leftarrow V^{\mathsf{out}}(\mathsf{seed}^{\mathsf{out}})$, or a query to $\Pi^{\mathsf{in}}_{\mathsf{proof}}(\mathsf{seed}^{\mathsf{out}})$.

Assume that the rectangular neighbour listing algorithm for V^{out} partitions $\mathsf{seed}^{\mathsf{out}}.\mathsf{shared}$ into $(\mathsf{seed}^{\mathsf{out}}.\mathsf{shared.row}, \mathsf{seed}^{\mathsf{out}}.\mathsf{shared.col})$. We now partition $\mathsf{seed.shared}$ as follows:

$$\mathsf{seed.shared.row} := (\mathsf{seed}^{\mathsf{out}}.\mathsf{shared.row}, \mathsf{low}(\mathsf{seed}^{\mathsf{in}})),$$
$$\mathsf{seed.shared.col} := (\mathsf{seed}^{\mathsf{out}}.\mathsf{shared.col}, \mathsf{high}(\mathsf{seed}^{\mathsf{in}})).$$

The algorithms A_{shared}, A_{row}, and A_{col} for the RNL of V^{comp} work as follows.

Case 1. Given the configuration (seed, k), the verifier V^{comp} probes the i-th bit of $\Pi^{\mathsf{out}}_{\mathsf{proof}}|_{I_{\mathsf{out}}}$, where the index i depends on the $\mathsf{seed}^{\mathsf{in}}$. In other words, the composed PCPP verifier probes the answer of the i-th query made by the outer PCPP verifier when it is "simulating" the outer verifier using the inner PCPP verifier. A neighbour $(\mathsf{seed}' = (\mathsf{seed}'^{\mathsf{out}}, \mathsf{seed}'^{\mathsf{in}}), k')$ of (seed, k) must be a query of the same type, i.e., it is a query to the i'-th bit of $\Pi^{\mathsf{out}}_{\mathsf{proof}}|_{I'_{\mathsf{out}}}$ where the index i' depends on $\mathsf{seed}'^{\mathsf{in}}$. Furthermore, the i-th query index in I_{out} must be the same as the i'-th query index in I'_{out}. In such case, A_{shared} generate the following list:

1. We first run the inner PCPP verifier using $\mathsf{seed}^{\mathsf{in}}$ to obtain the index i defined above.
2. Then, we use $A^{\mathsf{out}}_{\mathsf{shared}}$ to obtain the list $\mathsf{NList}^{\mathsf{out}}_{\mathsf{shared}}$. Let $\ell := |\mathsf{NList}^{\mathsf{out}}_{\mathsf{shared}}|$, for each $j \in [\ell]$, the j-th element i_j in this list indicates that the j-th neighbour of (seed, k) is the i_j-th query made by V^{out} on some $(\mathsf{seed}^{\mathsf{out}})'$.
3. For every i_j, we enumerate $(\mathsf{seed}'^{\mathsf{in}}, k') \in \{0,1\}^{r^{\mathsf{in}}} \times [q^{\mathsf{in}}]$ in lexicographic order. If the k'-th query of V^{in} given $\mathsf{seed}'^{\mathsf{in}}$ as the seed probes exactly the i_j-th bit of $(\Pi_{\mathsf{input}} \circ \Pi^{\mathsf{out}}_{\mathsf{proof}})|_{I'_{\mathsf{out}}}$, then we append k' into $\mathsf{NList}_{\mathsf{shared}}$.

Similarly, A_{row} also performs the above three steps, but each time, instead of appending k' into $\mathsf{NList}_{\mathsf{shared}}$, it appends

$$(\mathsf{seed}^{\mathsf{out}}_j.\mathsf{row}, \mathsf{seed}^{\mathsf{out}}_j.\mathsf{shared.row}, \mathsf{low}(\mathsf{seed}'^{\mathsf{in}}))$$

into $\mathsf{NList}_{\mathsf{row}}$, where $(\mathsf{seed}^{\mathsf{out}}_j.\mathsf{row}, \mathsf{seed}^{\mathsf{out}}_j.\mathsf{shared.row})$ is the corresponding row-part randomness in $\mathsf{NList}^{\mathsf{out}}_{\mathsf{row}}$. The behaviour of A_{col} is similar. It is easy to see that for fixed $(\mathsf{seed}^{\mathsf{out}}.\mathsf{shared}, \mathsf{seed}^{\mathsf{in}})$, $\mathsf{NList}_{\mathsf{row}}$ and $\mathsf{NList}_{\mathsf{col}}$ can be computed by a projection over $\mathsf{NList}^{\mathsf{out}}_{\mathsf{row}}$ and $\mathsf{NList}^{\mathsf{out}}_{\mathsf{col}}$ respectively.

Finally, the index of (seed, k) in NList (i.e., self) can be computed as follows: Whenever we are considering the $\mathsf{self}^{\mathsf{out}}$-th element in $\mathsf{NList}^{\mathsf{out}}$ (where $\mathsf{self}^{\mathsf{out}}$ is the self computed by the RNL of V^{out}) and we encounter $\mathsf{seed}'^{\mathsf{in}} = \mathsf{seed}^{\mathsf{in}}$ and $k' = k$, the current element in NList is self. Clearly, this can be computed from $\mathsf{self}^{\mathsf{out}}$ and $\mathsf{seed.shared}$.

Case 2. Given the configuration (seed, k), the verifier V^{comp} probes the i-th bit of $\Pi^{\mathsf{in}}_{\mathsf{proof}}(\mathsf{seed}^{\mathsf{out}})$. Recall that for every $\mathsf{seed}^{\mathsf{out}} \in \{0,1\}^{r^{\mathsf{out}}}$, $\Pi^{\mathsf{in}}_{\mathsf{proof}}(\mathsf{seed}^{\mathsf{out}})$ is arranged in a block of size $H^{\mathsf{in}}_{\mathsf{proof}} \times W^{\mathsf{in}}_{\mathsf{proof}}$ in the proof matrix. The neighbours of (seed, k) need to query the same block, therefore the neighbours must have the same random seed for the outer PCPP verifier. Hence, the RNL algorithms work as follows:

1. We first compute the list $\mathcal{L}^{\mathsf{in}} := \{(\mathsf{seed}^{\mathsf{in}}_j, k_j) \in \{0,1\}^{r^{\mathsf{in}}(d^{\mathsf{out}}(n))} \times [q^{\mathsf{comp}}]\}$ sorted in lexicographic order such that the inner PCPP will query the i-th bit of the inner proof on the k_j-th query given $\mathsf{seed}^{\mathsf{in}}_j$ as randomness. This can be done in $\mathrm{poly}(\ell^{\mathsf{in}}, d^{\mathsf{in}})$-time by enumerating all possible $(\mathsf{seed}^{\mathsf{in}}_j, k_j)$ and running the inner PCPP verifier.
2. We define the final list of neighbours as

$$\mathcal{L} := \left\{ \left(\mathsf{seed}_j := (\mathsf{seed}^{\mathsf{out}}, \mathsf{seed}^{\mathsf{in}}_j, k_j) \right) : (\mathsf{seed}^{\mathsf{in}}_j, k_j) \in \mathcal{L}^{\mathsf{in}} \right\}.$$

It is easy to check that the list satisfies the promises of the rectangular neighbour listing property. To compute self, we only need to find the position of $(\mathsf{seed}^{\mathsf{in}}, k)$ in $\mathcal{L}^{\mathsf{in}}$, which can be computed from $\mathsf{seed.shared}$. Moreover, the row (column) part of $\mathcal{L}$ can be computed by projections over $\mathsf{seed.row}$ ($\mathsf{seed.col}$) given $\mathsf{seed.shared}$.

Other properties. The soundness error and proximity parameter can be found in [BGH+06, Section 2.4]. Also, V^{comp} inherits the query complexity and decision complexity of V^{in}. We can see that the proof matrix of the composed PCPP verifier has width $W^{\mathsf{comp}}_{\mathsf{proof}} = W^{\mathsf{out}}_{\mathsf{proof}}$ and height $H^{\mathsf{comp}}_{\mathsf{proof}} = W^{\mathsf{out}}_{\mathsf{proof}} + 2^{r^{\mathsf{out}}(n)+r^{\mathsf{in}}(d^{\mathsf{out}}(n))}/W^{\mathsf{out}}_{\mathsf{proof}}$ (recall that $\ell^{\mathsf{in}} = 2^{r^{\mathsf{in}}}$). By the definitions of the random seeds, we can see that: The row and column randomness complexity of V^{comp} is the same as the row and column randomness complexity of V^{out}, respectively; the shared randomness complexity of V^{comp} is the sum of the shared randomness complexity of V^{out} and the randomness complexity of V^{in}.

Remark 5.1.10. The composed PCPP verifier V^{comp} will use the inner PCPP verifier V^{in} to simulate the outer PCPP verifier V^{out}. This means that the total number of queries and parity-check functions is at most the query complexity of the inner PCPP verifier. Moreover, the decision predicate of V^{comp} (after fixing the random seed) is the decision predicate of V^{in}, where the input bits of the decision circuit of V^{comp} are the parity-check bits and the answers to the queries. For instance, if the decision predicate of V^{in} given $\mathsf{seed}^{\mathsf{in}}$ is an OR of the answers or their negations, then the decision predicate of V^{comp} given $\mathsf{seed} = (\mathsf{seed}^{\mathsf{in}}, \mathsf{seed}^{\mathsf{out}})$ is also the same OR of its input bits (i.e. the answers to the queries and the parity-check bits).

5.1.4 Smoothening a PCPP with RNL

By slightly generalising the technique of [BHPT24], we can smoothen a rectangular PCPP with the rectangular neighbour listing property.

Verifier	V^{old}	V^{new}
Soundness error	s	$s + \mu$
Proximity parameter	δ	δ
Row randomness	r_{row}	r_{row}
Column randomness	r_{col}	r_{col}
Shared randomness	r_{shared}	r_{shared}
Proof matrix height	H_{proof}	$q \cdot 2^{r_{\mathsf{row}}+r_{\mathsf{shared}}/2}$
Proof matrix width	W_{proof}	$2^{r_{\mathsf{col}}+r_{\mathsf{shared}}/2}$
Query complexity	q	$\mathrm{poly}(q/\mu)$
Parity check complexity	p	p
Decision complexity	d	$\mathrm{poly}(d, q/\mu, t_{\mathsf{RNL}}(n))$

Table 5.3: The parameters of the "smoothened" PCPP V^{new}.

Theorem 5.1.11. *Suppose that L has a rectangular PCPP verifier V^{old} with ROP and $t_{\mathsf{RNL}}(n)$-time RNL property whose parameters are specified in Table 5.3. Then for every $\mu \in (0,1)$, L has a smooth and rectangular PCPP verifier V^{new} with ROP and the parameters specified in Table 5.3. Moreover, if the RNL for V^{old} can be computed by projections, then the query indices of V^{new} can be computed by projections as well.*

Proof. Let $\Pi^{\mathsf{old}} : \{0,1\}^{\ell} \to \{0,1\}$ be the proof oracle of V^{old}. Assume that $V^{\mathsf{old}}(\mathsf{seed}, i)$ outputs the index of the i-th query of V^{old} given the randomness $\mathsf{seed} \in \{0,1\}^r$. The "smoothened" verifier V^{new} expects the proof $\Pi^{\mathsf{new}} : \{0,1\}^{2^r} \times [q] \to \{0,1\}$ to be

$$\Pi^{\mathsf{new}}(\mathsf{seed}, i) := \Pi^{\mathsf{old}}[V^{\mathsf{old}}(\mathsf{seed}, i)].$$

Concretely, V^{new} works as follows: First, it checks that Π^{new} is (close to being) defined as above, i.e., there is a proof matrix Π^{old} such that $\Pi^{\mathsf{new}}(\mathsf{seed}, i)$ and $\Pi^{\mathsf{old}}[V^{\mathsf{old}}(\mathsf{seed}, i)]$ are sufficiently close. Then, it runs V^{old} using Π^{new} as the proof oracle, i.e., the verifier randomly chooses a $\mathsf{seed} \in \{0,1\}^r$, queries $\Pi^{\mathsf{new}}(\mathsf{seed}, 1), \Pi^{\mathsf{new}}(\mathsf{seed}, 2), \ldots, \Pi^{\mathsf{new}}(\mathsf{seed}, q)$, and decides whether to accept using the decision predicate of V^{old}. In fact, as in [BHPT24, Section 4.1], the first step can be combined into the second step: we only need to check the consistency of Π^{new} on the fly during the simulation of V^{old}.

The verifier V^{new}. For $\alpha \in (0,1)$, we say a graph $G = (V, E)$ is an *α-sampler* if for every $S \subseteq V$,

$$\Pr_{v \leftarrow V}\left[\left|\frac{|S|}{|V|} - \frac{\Gamma(v) \cap S}{\Gamma(v)}\right| > \alpha\right] < \alpha,$$

where $\Gamma(v)$ is the set of neighbours of v in G. By [Gol11c], there is a $\mathrm{poly}(n)$-time algorithm that given n and $\alpha \in (0,1)$, constructs a $(4/\alpha^4)$-regular graph on n vertices that is an α-sampler.

Let $\alpha := \mu/(10q)$, $\Delta := (4/\alpha^4) + 1$, then there is an explicit construction of $(\Delta - 1)$-regular α-sampler. Our new PCPP verifier works as follows:

- Let $\mathsf{seed} \in \{0,1\}^r$ be the random bits and $i \in [q]$ be the index of a query. If $V^{\mathsf{old}}(\mathsf{seed}, i)$ makes a query to the input oracle, it firstly makes the same query to the input oracle, and then probes $\Pi^{\mathsf{new}}(\mathsf{seed}, i)$ for Δ times. The last Δ queries to $\Pi^{\mathsf{new}}(\mathsf{seed}, i)$ are for the smoothness property.

- Now we assume that $V^{\mathsf{old}}(\mathsf{seed}, i)$ makes a query to the proof oracle. Let $\mathsf{NList} := \mathsf{NList}(\mathsf{seed}, i)$ be the ordered list of neighbours of (seed, i) from the rectangular neighbour listing property, $m := |\mathsf{NList}|$, and $\mathsf{self} \in [m]$ be the index of (seed, i) in the list. Let G^{NList} be an explicit $(\Delta - 1)$-regular α-sampler with m nodes from [Gol11c]. Let $j_2, j_3, \ldots, j_\Delta \in [m]$ be the neighbours of self in G^{NList}, and $j_1 := \mathsf{self}$. The verifier probes

$$\Pi^{\mathsf{new}}(\mathsf{NList}[j_1]), \Pi^{\mathsf{new}}(\mathsf{NList}[j_2]), \ldots, \Pi^{\mathsf{new}}(\mathsf{NList}[j_\Delta]),$$

  and rejects if the answers are not the same. Otherwise, the verifier treats this consistent answer as the answer to the i-th query of V^{old} and simulates V^{old}.

Rectangularity. Recall that the proof oracle is $\Pi^{\mathsf{new}} : \{0,1\}^{2^r} \times [q] \to \{0,1\}$, where $r = r_{\mathsf{row}} + r_{\mathsf{col}} + r_{\mathsf{shared}}$ is the randomness complexity. By the rectangular neighbour listing property of V^{old}, we know that the shared randomness can be partitioned into $(\mathsf{seed.shared.row}, \mathsf{seed.shared.col}) \in \{0,1\}^{r_{\mathsf{shared}}/2} \times \{0,1\}^{r_{\mathsf{shared}}/2}$. We define $W^{\mathsf{new}} := 2^{r_{\mathsf{col}} + r_{\mathsf{shared}}/2}$, $H^{\mathsf{new}} := q \cdot 2^{r_{\mathsf{row}} + r_{\mathsf{shared}}/2}$, and the $H^{\mathsf{new}} \times W^{\mathsf{new}}$ proof matrix

$$\begin{aligned}
\Pi^{\mathsf{new}}[u, v] &:= \Pi^{\mathsf{new}}(\mathsf{seed}, i); \\
\text{where} \quad u &:= (\mathsf{seed.row}, \mathsf{seed.shared.row}, i) \in \{0,1\}^{r_{\mathsf{row}} + r_{\mathsf{shared}}/2 + \log q}, \\
v &:= (\mathsf{seed.col}, \mathsf{seed.shared.col}) \in \{0,1\}^{r_{\mathsf{col}} + r_{\mathsf{shared}}/2}, \\
\mathsf{seed} &:= (\mathsf{seed.row}, \mathsf{seed.col}, \mathsf{seed.shared} := (\mathsf{seed.shared.row}, \mathsf{seed.shared.col})).
\end{aligned}$$

Now it suffices to construct the type predicate $V^{\mathsf{new}}_{\mathsf{type}}$ and the row and column verifiers $V^{\mathsf{new}}_{\mathsf{row}}$ and $V^{\mathsf{new}}_{\mathsf{col}}$. Recall that the new verifier V^{new} simulates V^{old} as follows: If V^{old} makes a query to the proof oracle, it makes Δ queries to the proof oracle using the RNL property; otherwise, it makes the same query to the input oracle and Δ queries to the same bit of the proof oracle.

- The type predicate $V^{\mathsf{new}}_{\mathsf{type}}$ (given the shared randomness) calls the type predicate $V^{\mathsf{old}}_{\mathsf{type}}$ of the old PCPP verifier, obtains the list of types of the queries, replaces each "proof" by Δ continuous "proof" and replaces each "input" by an "input" and Δ continuous "proof".

- For a query of V^{old} to the proof oracle, the row verifier $V^{\mathsf{new}}_{\mathsf{row}}$ (resp. the column verifier $V^{\mathsf{new}}_{\mathsf{col}}$) calls the row verifier $V^{\mathsf{old}}_{\mathsf{row}}$ (resp. the column verifier $V^{\mathsf{old}}_{\mathsf{col}}$) of the old PCPP. By the rectangular neighbour listing property, it can list the "row-part" (resp. the "column-part") of the neighbour list NList and also knows the index self in the list. It then constructs the sampler, finds the Δ selected neighbours of self (including itself), and outputs the "row-parts" (resp. the "column-parts") of them.

  This also means that if the RNL for V^{old} can be computed by projections, then the query indices of V^{new} can be computed by projections as well. This is because the row (column) parts of the query indices can be computed by projections over $\mathsf{NList}_{\mathsf{row}}$ ($\mathsf{NList}_{\mathsf{col}}$), which (by our hypothesis on V^{old}) can be computed by projections over $\mathsf{seed.row}$ ($\mathsf{seed.col}$).

- For a query of V^{old} to the input oracle, the row verifier $V^{\mathsf{new}}_{\mathsf{row}}$ (resp. the column verifier $V^{\mathsf{new}}_{\mathsf{col}}$) calls the row verifier $V^{\mathsf{old}}_{\mathsf{row}}$ (resp. the column verifier $V^{\mathsf{old}}_{\mathsf{col}}$) of the old PCPP to obtain the query to the input oracle rectangularly. It is easy to see that the remaining Δ queries to the proof oracle can be done rectangularly.

Smoothness. We need to show that for uniformly random $\mathsf{seed} \leftarrow \{0,1\}^r$ and $i \leftarrow [q \cdot \Delta]$, each bit of the proof oracle Π^{new} is equally likely to be probed given randomness seed on the i-th query. Let $i_1 := \lfloor (i-1)/\Delta \rfloor + 1$ and $i_2 := (i-1) \bmod \Delta + 1$. Let $\hat{G} = (\hat{V}, \hat{E})$ be the "union" of all G^{NList}, that is:

- $\hat{V} := \{0,1\}^{2^r} \times [q]$.
- Let $(\mathsf{seed}, i_1), (\mathsf{seed}', i_1')$ be two configurations given which V^{old} will probe the proof oracle. Then $((\mathsf{seed}, i_1), (\mathsf{seed}', i_1')) \in \hat{E}$ if and only if the configurations (seed, i_1) and (seed', i_1') are neighbours, and there is an edge between them in G^{NList}, where NList is the neighbourhood containing these two configurations. We also add a self-loop on every node (seed, i_1) on which V^{old} probes the proof oracle.
- For each (seed, i_1) such that $V^{\mathsf{old}}(\mathsf{seed}, i_1)$ probes the input oracle, we add Δ self-loops on the node $(\mathsf{seed}, i_1) \in \hat{V}$.

Assume that $\mathsf{seed} \in \{0,1\}^r$ and $i \in [q \cdot \Delta]$ are uniformly chosen. The query pattern of V^{new} to the proof oracle is as follows: It firstly selects a node $(\mathsf{seed}, i_1) \in \hat{V}$ uniformly, and then chooses a uniform neighbour of it. It is easy to see that each bit of the proof oracle is probed with probability

$$\frac{\Delta}{2^r \cdot q \cdot \Delta} = \frac{1}{2^r \cdot q}.$$

Soundness. The soundness of V^{old} follows from [BHPT24, Appendix A.1] (which is for PCP instead of PCPP); for completeness, we present a self-contained proof here. Assume that x is δ-far from being in L and $\Pi^{\mathsf{new}} : \{0,1\}^{2^r} \times [q]$ is a proof, we need to show that the verifier accepts with probability at most $s + \mu$. Let $\Pi^{\mathsf{old}} : \{0,1\}^{\ell} \to \{0,1\}$ be defined as follows:

$$\Pi^{\mathsf{old}}[j] := \underset{(\mathsf{seed}, i) \in \{0,1\}^r \times [q]}{\mathsf{Majority}} \left\{ \Pi^{\mathsf{new}}(\mathsf{seed}, i) : V^{\mathsf{old}}(\mathsf{seed}, i) = j \right\},$$

By the soundness of V^{old}, we know that V^{old} will accept (x, Π^{old}) with probability at most s.

Let $\mathsf{idx}^i(\mathsf{seed}) \in [\ell]$ be the i-th query of V^{old} given randomness seed. An index $j \in [\ell]$ is said to be β-*consistent* if for at least β fraction of (seed, i) such that $\mathsf{idx}^i(\mathsf{seed}) = j$, $\Pi^{\mathsf{new}}(\mathsf{seed}, i) = \Pi^{\mathsf{old}}[j]$. We define the following events over the random variable seed:

- H is the event that for every $i \in [q]$, $\mathsf{idx}^i(\mathsf{seed})$ is $(1 - 2\alpha)$-consistent (recall that $\alpha := \mu/(10q)$ is the parameter of the sampler).
- M is the event that for every $i \in [q]$, $\Pi^{\mathsf{new}}(\mathsf{seed}, i) = \Pi^{\mathsf{old}}(\mathsf{idx}^i(\mathsf{seed}))$.
- A is the event that V^{new} accepts (x, Π^{new}) on the randomness seed.
- C_i is the event that the Δ queries made by V^{new} corresponding to the i-th query of V^{old} returns the same answer (i.e. the "consistency check" passes on the simulation of the i-th query of V^{old}).

Claim 5.1.12. $\Pr[A \wedge \overline{H}] \leq q\alpha$.

Claim 5.1.13. $\Pr[\overline{M} \wedge H] \leq 2q\alpha$.

Claim 5.1.14. $\Pr[A \wedge H] \leq 2q\alpha + s$.

From the claims above, we can see that

$$\begin{aligned}\Pr_{\mathsf{seed}\leftarrow\{0,1\}^r}[A] &= \Pr_{\mathsf{seed}\leftarrow\{0,1\}^r}[A\wedge\overline{H}] + \Pr_{\mathsf{seed}\leftarrow\{0,1\}^r}[A\wedge H]\\ &\le s + q\alpha + 2q\alpha\\ &\le s+\mu.\end{aligned}$$

Proof of Claim 5.1.12. Let (seed, i) be a configuration, $\mathsf{NList} := \mathsf{NList}(\mathsf{seed}, i)$ be the list of all its neighbours, $G^{\mathsf{NList}} = (V, E)$ be the explicit sampler graph corresponding to the neighbourhood NList (i.e. V contains the configurations in NList), and self be the node corresponding to (seed, i). We say that a configuration (seed, i) is an *error configuration* if

$$\left|\frac{|S|}{|V|} - \frac{|\Gamma(\mathsf{self})\cap S|}{|\Gamma(\mathsf{self})|}\right| > \alpha,$$

where $S := \{(\mathsf{seed}', i') \in V \mid \Pi^{\mathsf{new}}(\mathsf{seed}', i') \neq \Pi^{\mathsf{new}}(\mathsf{seed}, i)\}$. Since G^{NList} is an α-sampler, there are at most α fraction of error configurations in each neighbourhood. Suppose $\overline{H}$ happens, then there is some i such that $\mathsf{idx}^i(\mathsf{seed})$ is not $(1-2\alpha)$-consistent, which means that $|S|/|V| \ge 2\alpha$. Suppose in addition that C_i happens (i.e., the "consistency check" on the i-th query passes), then $\Gamma(\mathsf{self})\cap S = \varnothing$, which means that (seed, i) is an error configuration.

Let Err be the set of error configurations. Then

$$\begin{aligned}\Pr_{\mathsf{seed}\leftarrow\{0,1\}^r}[A\wedge\overline{H}] &\le \Pr_{\mathsf{seed}\leftarrow\{0,1\}^r}\left[C_1\wedge\cdots\wedge C_q\wedge\overline{H}\right]\\ &\le \Pr_{\mathsf{seed}\leftarrow\{0,1\}^r}\left[\exists i\in[q]\ (\mathsf{seed},i)\in\mathsf{Err}\wedge\overline{H}\right]\\ &\le q\cdot \Pr_{\substack{\mathsf{seed}\leftarrow\{0,1\}^r\\ i\leftarrow[q]}}[(\mathsf{seed},i)\in\mathsf{Err}\wedge\overline{H}]\\ &\le q\cdot\alpha.\end{aligned}$$

◇

Proof of Claim 5.1.13. For every $j\in[q]$, we denote by H_j the event that $\mathsf{idx}^j(\mathsf{seed})$ is $(1-2\alpha)$-consistent (and thus $H = \bigwedge_{j\in[q]} H_j$). Let $H^\star$ be the event that $\mathsf{idx}^i(\mathsf{seed})$ is $(1-2\alpha)$-consistent over the random variable $(\mathsf{seed}, i)\in\{0,1\}^r\times[q]$. We can see that:

$$\begin{aligned}\Pr_{\mathsf{seed}\leftarrow\{0,1\}^r}\left[\overline{M}\wedge H\right] &\le q\cdot \Pr_{(\mathsf{seed},i)\leftarrow\{0,1\}^r\times[q]}\left[\Pi^{\mathsf{new}}(\mathsf{seed},i)\neq\Pi^{\mathsf{old}}(\mathsf{idx}^i(\mathsf{seed}))\ \wedge\ \bigwedge_{j\in[q]}H_j\right]\\ &\le q\cdot \Pr_{(\mathsf{seed},i)\leftarrow\{0,1\}^r\times[q]}\left[\Pi^{\mathsf{new}}(\mathsf{seed},i)\neq\Pi^{\mathsf{old}}(\mathsf{idx}^i(\mathsf{seed}))\ \wedge\ H_i\right]\\ &= q\cdot \Pr_{(\mathsf{seed},i)\leftarrow\{0,1\}^r\times[q]}\left[\Pi^{\mathsf{new}}(\mathsf{seed},i)\neq\Pi^{\mathsf{old}}(\mathsf{idx}^i(\mathsf{seed}))\ \wedge\ H^\star\right].\end{aligned}\tag{5.10}$$

Let N be the set of all neighbourhoods that contain a configuration $(\mathsf{seed}, i)\in H^\star$ (i.e. $\mathsf{idx}^i(\mathsf{seed})$ is $(1-2\alpha)$-consistent). By the definition of $H^\star$ and the neighbours of configurations, we can see that for each $h\in N$, all the configurations in h are also in $H^\star$. Thus, the uniform distribution over $H^\star$ is identical to the following distribution: we first sample a neighbourhood $h\in N$ (with probability proportional to the size of h), then uniformly sample a configuration $(\mathsf{seed}, i)\in h$.

Thus, we have:

$$\begin{aligned}
(5.10) &\leq q \cdot \Pr_{(\mathsf{seed},i)\leftarrow\{0,1\}^r\times[q]}\left[\Pi^{\mathsf{new}}(\mathsf{seed},i) \neq \Pi^{\mathsf{old}}(\mathsf{idx}^i(\mathsf{seed})) \;\middle|\; H^\star\right] \\
&= q \cdot \mathop{\mathbb{E}}_{h\leftarrow\mathcal{N}}\left[\Pr_{(\mathsf{seed},i)\leftarrow h}\left[\Pi^{\mathsf{new}}(\mathsf{seed},i) \neq \Pi^{\mathsf{old}}(\mathsf{idx}^i(\mathsf{seed}))\right]\right] \\
&\leq q \cdot \mathop{\mathbb{E}}_{h\leftarrow\mathcal{N}}[2\alpha], \qquad (5.11) \\
&\leq 2q\alpha,
\end{aligned}$$

where $\mathcal{N}$ is some distribution over N, and (5.11) holds by the definition of $(1-2\alpha)$-consistency. ◇

Proof of Claim 5.1.14. We can see that:

$$\begin{aligned}
\Pr_{\mathsf{seed}\leftarrow\{0,1\}^r}[A \wedge H] &\leq \Pr_{\mathsf{seed}\leftarrow\{0,1\}^r}[\overline{M} \wedge H] + \Pr_{\mathsf{seed}\leftarrow\{0,1\}^r}[A \wedge H \wedge M] \\
&\leq 2q\alpha + \Pr_{\mathsf{seed}\leftarrow\{0,1\}^r}\left[V^{\mathsf{old}} \text{ accepts } (x,\Pi^{\mathsf{old}}) \wedge H \wedge M\right] \\
&\leq 2q\alpha + \Pr_{\mathsf{seed}\leftarrow\{0,1\}^r}\left[V^{\mathsf{old}} \text{ accepts } (x,\Pi^{\mathsf{old}})\right] \\
&\leq 2q\alpha + s.
\end{aligned}$$

Note that the first inequality follows from the definition of V^{new} and Π^{old}. ◇

Other Properties. The query complexity, parity-check complexity, and decision complexity can be easily checked by definition. □

5.1.5 Soundness Error Reduction

Recall that the above PCPPs we considered only satisfy robust soundness with a very large soundness error $1-\varepsilon$. In this subsection, we reduce the soundness error to an arbitrarily small value using expander walks. Take a constant-degree expander graph $G=(V,E)$ where $V=\{0,1\}^{r(n)}$. Use $r(n)+O(1)$ random bits to sample a random walk of length $O(1)$ over G. Then, for every vertex $v\in\{0,1\}^{r(n)}$ in the random walk, run the old PCPP verifier with v as randomness, reject if the old PCPP verifier rejects. If every invocation of the old PCPP verifier accepts, then our new PCPP verifier also accepts.

However, we need to be careful when implementing this approach: To preserve the rectangularity of the verifier, we take the *tensor product* of three expanders (one for row randomness, one for column randomness, and one for shared randomness). To ensure that V_{row} and V_{col} are (still) projections, we use the following family of 1-*local* expanders:

Lemma 5.1.15 ([VW18])**.** *For every $\lambda\in(0,1)$, there is some $d=\mathrm{poly}(\lambda^{-1})$ such that the following holds. For every n, there is an expander graph $G_n=(V_n,E_n)$ with second largest eigenvalue at most λ, where $V_n:=\{0,1\}^n$. Moreover, there are d explicit projections (i.e., NC^0_1 circuits) $C_1,C_2,\ldots,C_d:\{0,1\}^n\to\{0,1\}^n$ such that for every $x\in V_n$, the d neighbours of x are $C_1(x),C_2(x),\ldots,C_d(x)$.*

Verifier	V^{old}	V^{new}
Soundness error	$1-\varepsilon$	μ
Proximity parameter	δ	δ
Row randomness	r_{row}	r_{row}
Column randomness	r_{col}	r_{col}
Shared randomness	r_{shared}	$r_{\mathsf{shared}} + O(\ell \log(1/\varepsilon))$
Proof matrix height	H_{proof}	H_{proof}
Proof matrix width	W_{proof}	W_{proof}
Query complexity	q	$O(q\ell)$
Parity check complexity	p	$O(p\ell)$
Decision complexity	d	$O(d\ell + \mathrm{poly}(r_{\mathsf{shared}}, r_{\mathsf{row}}, r_{\mathsf{col}}))$

Table 5.4: The parameters of the soundness error reduction (where $\ell := (1/\varepsilon^2)\log(1/\mu)$ and $O(\cdot)$ hides absolute constants).

Lemma 5.1.16 (Expander Walk, [AB09, Theorem 21.12])**.** *Let $G = (V, E)$ be a d-regular graph with second largest eigenvalue λ. For every $S \subseteq V$ such that $|S| \leq \beta \cdot |V|$ for some $\beta \in (0,1)$, let $(X_1, X_2, \ldots, X_\ell)$ be a random walk in G with random starting point, then*

$$\Pr[\forall i \in [\ell], X_i \in S] \leq \Big((1-\lambda)\sqrt{\beta} + \lambda\Big)^{\ell-1}.$$

Lemma 5.1.17 (Expander Chernoff Bound, [Vad12, Theorem 4.22])**.** *Let $G = (V, E)$ be a d-regular graph with second largest eigenvalue λ, $B \subseteq V$ be a set of size $|B| = \beta|V|$. Let $X_1, X_2, \ldots, X_\ell$ be random variables denoting a length-ℓ random walk from a random starting point. For every $i \in [\ell]$, we define $B_i = 1$ if $X_i \in B$ and $B_i = 0$ otherwise. Then:*

$$\Pr\left[\left|\frac{1}{\ell}\sum_{i=1}^{\ell} B_i - \beta\right| \geq 2\lambda\right] < 2\exp\big(-\Omega(\lambda^2\ell)\big).$$

Theorem 5.1.18. *Suppose that L has a rectangular PCPP verifier V^{old} (resp. a rectangular PCPP verifier V^{old} with ROP), where the parameters are specified in Table 5.4. For every $\mu \in (0,1)$, letting $\ell := (1/\varepsilon^2)\log(1/\mu)$, then L has a rectangular PCPP verifier V^{new} (resp. a rectangular PCPP verifier with ROP), whose parameters are specified in Table 5.4. Moreover:*

- *If V^{old} has robust soundness error (instead of plain soundness error) $1-\varepsilon$ with robustness parameter ρ, then V^{old} has robustness soundness error μ with robustness parameter $\varepsilon\rho/3$.*
- *If the query indices of V^{old} can be computed by projections, then the query indices of V^{new} can be computed by projections as well.*
- *If V^{old} is smooth, then V^{new} is also smooth.*

Proof. Let $\lambda := \varepsilon/3$ and $r := r_{\mathsf{row}} + r_{\mathsf{col}} + r_{\mathsf{shared}}$. For $d = \mathrm{poly}(1/\lambda)$, we construct the following d-regular expander graphs with second largest eigenvalue λ by Lemma 5.1.15: $G_{\mathsf{row}} = (V_{\mathsf{row}}, E_{\mathsf{row}})$ with $V_{\mathsf{row}} := \{0,1\}^{r_{\mathsf{row}}}$, $G_{\mathsf{col}} = (V_{\mathsf{col}}, E_{\mathsf{col}})$ with $V_{\mathsf{col}} := \{0,1\}^{r_{\mathsf{col}}}$, and $G_{\mathsf{shared}} = (V_{\mathsf{shared}}, E_{\mathsf{shared}})$ with $V_{\mathsf{shared}} := \{0,1\}^{r_{\mathsf{shared}}}$. Let $G = (V, E)$ be the tensor product of these expanders:

$$V := V_{\mathsf{row}} \times V_{\mathsf{col}} \times V_{\mathsf{shared}} = \{0,1\}^{r_{\mathsf{row}}} \times \{0,1\}^{r_{\mathsf{col}}} \times \{0,1\}^{r_{\mathsf{shared}}};$$

$$E := \{((u,v,w),(u',v',w')) : (u,u') \in E_{\mathsf{row}}, (v,v') \in E_{\mathsf{col}}, (w,w') \in E_{\mathsf{shared}}\}.$$

Note that G is a d^3-regular graph with second largest eigenvalue λ (see [AB09, Lemma 21.17]).

The Construction of V^{new}. The new verifier has the same proof matrix, row randomness, and column randomness as the old verifier V^{old}. The shared random seed of the new verifier V^{new} consists of the shared random seed $\mathsf{seed.shared}$ of V^{old} and $\mathsf{seed.walk}$, which is used to sample a random walk in G of length $\ell := O(\lambda^{-2}\log(\mu^{-1}))$. Concretely:

- The random seed $\mathsf{seed.walk}$ will be used to sample $\sigma_1, \sigma_2, \ldots, \sigma_{3(\ell-1)} \in [d]$. We can see that
$$|\mathsf{seed.walk}| = O(\ell \cdot \log d) = O(\lambda^{-2}\log(\mu^{-1})\log(\lambda^{-1})).$$
- Let $u_1 := \mathsf{seed.row}$, $v_1 := \mathsf{seed.col}$, and $w_1 := \mathsf{seed.shared}$. We use $\sigma_1, \sigma_2, \ldots, \sigma_{\ell-1}$ to specify a length-ℓ random walk $(u_1, u_2, \ldots, u_\ell)$ in G_{row}. In particular, let $C_1, C_2, \ldots, C_d$ be the projections in Lemma 5.1.15 for G_{row}. For every $j \in \{1, 2, \ldots, \ell-1\}$, we define $u_{j+1} := C_{\sigma_j}(u_j)$. Similarly, we can use the remaining $2(\ell-1)$ bits to specify a random walk $(v_1, v_2, \ldots, v_\ell)$ in G_{col} and a random walk $(w_1, w_2, \ldots, w_\ell)$ in G_{shared}.

The verifier V^{old} will run the verifier V^{new} for ℓ times with the seeds:

$$(u_1, v_1, w_1), (u_2, v_2, w_2), \ldots, (u_\ell, v_\ell, w_\ell),$$

and will accept the proof if V^{old} accepts given all these ℓ seeds. Since $\mathsf{seed.walk}$ is in the shared randomness of V^{new} and G is obtained from the tensor product of G_{row}, G_{col}, and G_{shared}, it is easy to see that V^{new} is still a rectangular PCPP verifier. The query complexity (and parity-check complexity when V^{old} has ROP) increases by an $\ell = O(\varepsilon^{-2}\log(\mu^{-1}))$ multiplicative factor. Finally, it follows from Lemma 5.1.15 that each u_i (resp. v_i) can be computed by a projection over $\mathsf{seed.row}$ (resp. $\mathsf{seed.col}$), hence the query indices of V^{new} can be computed by projections if the query indices of V^{old} can.

Smoothness. Let $\mathsf{idx} \in [H_{\mathsf{proof}} \cdot W_{\mathsf{proof}}]$ be an index in the proof. Denote as $V^{\mathsf{new}}(\mathsf{seed}, i)$ (resp. $V^{\mathsf{old}}(\mathsf{seed}, i)$) the index in the proof probed by V^{new} (resp. V^{old}) for the i-th query.

$$\begin{aligned}
&\Pr_{\mathsf{seed},\mathsf{seed.walk},i\in[q\ell]}[V^{\mathsf{new}}(\mathsf{seed} \circ \mathsf{seed.walk}, i) = \mathsf{idx}] \\
&= \mathop{\mathbb{E}}_{j\in[\ell]}\left[\Pr_{\mathsf{seed},\mathsf{seed.walk},i\in[q]}[V^{\mathsf{new}}(\mathsf{seed} \circ \mathsf{seed.walk}, (j-1)\ell + i) = \mathsf{idx}]\right]. \qquad (5.12)
\end{aligned}$$

Fix $j \in [\ell]$. By the definition of V^{new}, we know that $V^{\mathsf{new}}(\mathsf{seed} \circ \mathsf{seed.walk}, (j-1)\ell + i)$ works as follows: Let $(u, v, w) := (\mathsf{seed.row}, \mathsf{seed.col}, \mathsf{seed.shared})$, and $\sigma_1, \sigma_2, \ldots, \sigma_{3\ell}$ be defined as above; V^{new} will choose the j-th node in the random walk on G seeded by $\mathsf{seed.walk}$ starting from $(u_1 := u, v_1 := v, w_1 := w)$ as the seed for V^{old}, and probe the proof according to the i-th query of V^{old}. Since the expander graph is regular, each $\mathsf{seed} \in \{0,1\}^r$ is equally likely to be selected

from a random walk with a random starting point. Hence

$$\begin{aligned}
&\Pr_{\mathsf{seed},\mathsf{seed.walk},i\in[q]}[V^{\mathsf{new}}(\mathsf{seed}\circ\mathsf{seed.walk},(j-1)\ell+i)=\mathsf{idx}] \\
&=\Pr_{\mathsf{seed},i\in[q]}\left[V^{\mathsf{old}}(\mathsf{seed},i)=\mathsf{idx}\right] \\
&=\frac{1}{H_{\mathsf{proof}}\cdot W_{\mathsf{proof}}}.
\end{aligned}$$

This means that (5.12) $=\frac{1}{H_{\mathsf{proof}}\cdot W_{\mathsf{proof}}}$, i.e., every bit in the proof is equally likely to be probed.

Soundness. Assume that $x\in\{0,1\}^n$ is δ-far from being in L and let Π be an arbitrary proof. We say a node (u,v,w) in the expander graph $G=(V,E)$ is *bad* if V^{old} accepts (x,Π) with the random seed $\mathsf{seed.row}:=u$, $\mathsf{seed.col}:=v$, and $\mathsf{seed.shared}:=w$. Let B be the set of all bad nodes, then $|B|\le(1-\varepsilon)\cdot|V|$. Note that the new verifier accepts (x,Π) if and only if a length-ℓ random walk on G from a random starting point only accesses bad nodes. By Lemma 5.1.16, we can see that

$$\Pr\left[V^{\mathsf{new}}\text{ accepts }(x,\Pi)\right]\le\left((1-\lambda)\sqrt{1-\varepsilon}+\lambda\right)^{\ell-1}\le\left(1-\frac{\varepsilon}{5}\right)^{\ell-1}\le\exp\left(-\frac{\varepsilon(\ell-1)}{5}\right)\le\mu,$$

when $\ell\ge10\cdot\varepsilon^{-1}\ln(\mu^{-1})$.

Robust Soundness. Assume that the original PCPP verifier V^{old} has robust soundness error $1-\varepsilon$ with robustness parameter ρ, we show that V^{new} has robust soundness error μ. Let $x\in\{0,1\}^n$ be δ-far from L and Π be an arbitrary proof. We say a node (u,v,w) in the expander graph $G=(V,E)$ is *bad* if given the randomness $\mathsf{seed}=(\mathsf{seed.row},\mathsf{seed.col},\mathsf{seed.shared})=(u,v,w)$, the fraction of bits read by the old PCPP verifier that we need to change to make V^{old} accept (x,Π) is at most ρ.

Let B be the set of bad nodes and $X_1,X_2,\ldots,X_\ell$ be the random variables denoting a random walk from a random starting point (equivalently, denoting the randomness V^{new} used to simulate V^{old}). It follows from the robustness soundness of V^{old} that $|B|\le(1-\varepsilon)\cdot|V|$. Let $B_i=1$ when $X_i\in B$ and 0 otherwise. By Lemma 5.1.17, we can see that

$$\Pr\left[\frac{1}{\ell}\sum_{i=1}^{\ell}B_i\ge1-\varepsilon+2\lambda\right]\le2\exp(-\Omega(\lambda^2\ell))\le\mu$$

when $\ell=O(\lambda^{-2}\log(\mu^{-1}))$. As a result, with probability as least $1-\mu$, the fraction of bits read by V^{new} that we need to change to make V^{new} accepts (x,Π) is at least

$$(1-(1-\varepsilon+2\lambda))\rho\ge\varepsilon\rho/3.$$

This satisfies the requirement of robust soundness μ with robustness parameter $\varepsilon\rho/3$. □

Soundness error	s
Proximity parameter	δ
Row randomness	$r_{\mathsf{row}} := h_{\mathsf{proof}} - (5/m)\log T(n)$
Column randomness	$r_{\mathsf{col}} := w_{\mathsf{proof}} - (5/m)\log T(n)$
Shared randomness	$r_{\mathsf{shared}} := (10/m)\log T(n) + O(\log\log T(n) + m\log m)$
Query complexity	q
Parity check complexity	q
Decision complexity	$\mathrm{poly}(T(n)^{1/m})$

Table 5.5: Parameters of the PCPP constructed in Theorem 2.5.11.

5.1.6 Final Construction

Theorem 5.1.19 ([Mie09])**.** *Let L be a pair language in $\mathsf{NTIME}[T(n)]$ for some non-decreasing function $T : \mathbb{Z}^+ \to \mathbb{Z}^+$. For all constants $s, \delta > 0$, L has a PCPP verifier with randomness complexity $\log T(n) + O(\log\log T(n))$, soundness error s, proximity parameter δ, query complexity $O(1)$, and decision complexity $\mathrm{polylog}(T(n))$.*

Theorem 2.5.11 (Smooth and Rectangular PCPP)**.** *For all constants $\delta \in (0,1)$ and $s \in (0,1)$, there is a constant $q \geq 1$ such that the following holds. Let $m = m(n)$, $T(n)$, $w_{\mathsf{proof}}(n)$, $w_{\mathsf{input}}(n)$ be good functions such that $1 \leq m(n) \leq (\log T(n))^{0.1}$, $n \leq T(n) \leq 2^{\mathrm{poly}(n)}$, $w_{\mathsf{proof}}(n) \leq \log T(n)$, and $w_{\mathsf{input}}(n) \leq \log n$. Then there are good functions $h_{\mathsf{proof}}(n)$ and $h_{\mathsf{input}}(n)$ satisfying*

$$h_{\mathsf{proof}}(n) := \log T(n) + \Theta(m\log\log T(n)) - w_{\mathsf{proof}}(n), \qquad \text{and}$$
$$h_{\mathsf{input}}(n) := \lceil \log n \rceil - w_{\mathsf{input}}(n),$$

such that the following holds.

Suppose that $w_{\mathsf{proof}}, h_{\mathsf{proof}} \geq (5/m)\log T(n)$, and that for some absolute constant $C \geq 1$,

$$\frac{w_{\mathsf{input}}(n)}{w_{\mathsf{proof}}(n)}, \frac{h_{\mathsf{input}}(n)}{h_{\mathsf{proof}}(n)} \leq 1 - \frac{Cm^2\log\log T(n)}{\log T(n)}.$$

Let $W_{\mathsf{proof}}(n) := 2^{w_{\mathsf{proof}}(n)}$, $H_{\mathsf{proof}}(n) := 2^{h_{\mathsf{proof}}(n)}$, $W_{\mathsf{input}}(n) := 2^{w_{\mathsf{input}}(n)}$, and $H_{\mathsf{input}}(n) := 2^{h_{\mathsf{input}}(n)}$. Then $\mathsf{NTIME}[T(n)]$ has a smooth and rectangular PCP of proximity with an $H_{\mathsf{input}}(n) \times W_{\mathsf{input}}(n)$ input matrix and an $H_{\mathsf{proof}}(n) \times W_{\mathsf{proof}}(n)$ proof matrix, with query indices computable by projections, and whose other parameters are specified in Table 2.2.

Proof. The high-level roadmap of the proof is as follows.

1. From Theorem 5.1.3, we obtain a robust and rectangular PCPP verifier V^{out} with RNL property for $t_{\mathsf{RNL}} = \mathrm{poly}(\log T(n), m^m)$ and query complexity $T(n)^{1/m} \cdot \mathrm{polylog}(T(n))$.
2. Let V^{in} be the PCPP verifier for Circuit-Eval$^{\perp}$ with constant query complexity in Theorem 5.1.19. We compose V^{out} and V^{in} by Theorem 5.1.9 to obtain a rectangular PCPP verifier V^{comp} with RNL property.
3. We smoothen V^{comp} by Theorem 5.1.11 to obtain a smooth and rectangular PCPP V^{smth} with constant query complexity, whose soundness error is some constant $s^{\mathsf{smth}} \in (0,1)$.

4. By Theorem 5.1.18, we reduce the soundness error to s while still maintaining the query complexity to be a (larger) constant.

Robust and Rectangular PCPP. Let $\delta \in (0,1)$ and $s \in (0,1)$ be some constants; q be a large constant to be determined later that only depends on δ and s; C be a large constant; $m = m(n)$, $T(n)$, $w_{\mathsf{input}}(n)$, $h_{\mathsf{input}}(n)$, $w_{\mathsf{proof}}(n)$, and $h_{\mathsf{proof}}(n)$ be defined as above. Let $w^{\mathsf{out}}_{\mathsf{proof}}(n) := w_{\mathsf{proof}}(n) - O(\log\log T(n) + m\log m)$ where the concrete value will be determined later. We will set $h_{\mathsf{proof}}(n) = h^{\mathsf{out}}_{\mathsf{proof}}(n) + O(\log\log T(n) + m\log m)$ for some good function $h^{\mathsf{out}}_{\mathsf{proof}}(n)$ (which is actually the proof height parameter of the outer PCPP). We check the technical conditions of Theorem 5.1.3 holds; in particular, we need to ensure that

Claim 5.1.20. *For some constant C' that could be made large enough (depending on C),*

$$\frac{w_{\mathsf{input}}}{w^{\mathsf{out}}_{\mathsf{proof}}}, \frac{h_{\mathsf{input}}}{h^{\mathsf{out}}_{\mathsf{proof}}} \le 1 - \frac{C' m \log\log T(n)}{\log T(n)}.$$

Proof. Since $w_{\mathsf{proof}} \ge (5/m)\log T(n)$, and $w_{\mathsf{proof}} - w^{\mathsf{out}}_{\mathsf{proof}} \le \alpha_1(\log\log T(n) + m\log m)$ for some constant α_1, it follows that

$$\begin{aligned}
\frac{w_{\mathsf{input}}}{w^{\mathsf{out}}_{\mathsf{proof}}} &\le \frac{w_{\mathsf{input}}}{w_{\mathsf{proof}}}\left(1 + \frac{\alpha_1(\log\log T(n) + m\log m)}{w_{\mathsf{proof}} - \alpha_1(\log\log T(n) + m\log m)}\right) \\
&\le 1 - \frac{Cm^2 \log\log T(n)}{\log T(n)} + \frac{\Theta(m)\cdot(\log\log T(n) + m\log m)}{\log T(n)} \\
&\le 1 - \frac{C' m \log\log T(n)}{\log T(n)}.
\end{aligned}$$

The same argument works for $\frac{h_{\mathsf{input}}}{h^{\mathsf{out}}_{\mathsf{proof}}}$. ◇

We also note that $w^{\mathsf{out}}_{\mathsf{proof}}, h^{\mathsf{out}}_{\mathsf{proof}} \ge (4/m)\log T(n)$. Hence, we can use Theorem 5.1.3 to obtain a robust and rectangular PCPP verifier V^{out} for L with RNL property and other parameters as follows:

- Proximity parameter $\delta^{\mathsf{out}} := \delta$.
- Robust soundness error $s^{\mathsf{out}} := 1 - \rho^{\mathsf{out}}$ with robustness parameter ρ^{out}, where $\rho^{\mathsf{out}} \in (0,1)$ is some constant depending on δ.
- Proof matrix size $H^{\mathsf{out}}_{\mathsf{proof}}(n) \times W^{\mathsf{out}}_{\mathsf{proof}}(n)$, where $H^{\mathsf{out}}_{\mathsf{proof}} = 2^{h^{\mathsf{out}}_{\mathsf{proof}}}$, $W^{\mathsf{out}}_{\mathsf{proof}} = 2^{w^{\mathsf{out}}_{\mathsf{proof}}}$. The proof height parameter $h^{\mathsf{out}}_{\mathsf{proof}}$, which is given by Theorem 5.1.3, satisfies

$$h^{\mathsf{out}}_{\mathsf{proof}} = \log T(n) + \Theta(m\log\log T(n)) - w^{\mathsf{out}}_{\mathsf{proof}}(n).$$

- Row randomness complexity $r^{\mathsf{out}}_{\mathsf{row}} = h^{\mathsf{out}}_{\mathsf{proof}} - (4/m)\log T(n)$.
- Column randomness complexity $r^{\mathsf{out}}_{\mathsf{col}} = w^{\mathsf{out}}_{\mathsf{proof}} - (4/m)\log T(n)$.
- Shared randomness complexity $r^{\mathsf{out}}_{\mathsf{shared}} = (7/m)\log T(n) + O(\log\log T(n) + m\log m)$.
- Query complexity $q^{\mathsf{out}}(n) = T(n)^{1/m}\cdot \mathrm{polylog}(T(n))$.
- Decision complexity $d^{\mathsf{out}}(n) = T(n)^{1/m}\cdot \mathrm{polylog}(T(n))$.
- RNL time complexity $t^{\mathsf{out}}_{\mathsf{RNL}}(n) = \mathrm{poly}(\log T(n), m^m)$.

Finally, the RNL for V^{out} can be computed by projections.

Reducing the Query Complexity. By Theorem 5.1.19, we can construct a PCPP verifier V^{in} for Circuit-Eval$^{\perp}$ with input length $d^{\mathsf{out}}(n)$ and other parameters specified as follows.

- Randomness complexity $r^{\mathsf{in}}(n) = \log d^{\mathsf{out}}(n) + O(\log\log d^{\mathsf{out}}(n)) = \frac{1}{m}\log T(n) + O(\log\log T(n))$.
- Soundness error $s^{\mathsf{in}} := \rho^{\mathsf{out}}/2$.
- Proximity parameter $\delta^{\mathsf{in}} := \rho^{\mathsf{out}}/2$.
- Query complexity $q^{\mathsf{in}} = O(1)$ is a constant depends on s^{in} and δ^{in}, which further means that it only depends on δ.
- Decision complexity $d^{\mathsf{in}}(d^{\mathsf{out}}(n)) = \mathrm{polylog}(T(n))$.

Without loss of generality, we assume that the proof length $\ell^{\mathsf{in}}(n) = 2^{r^{\mathsf{in}}(n)}$.

We now construct V^{comp} by composing V^{out} and V^{in} by Theorem 5.1.9. We first check the requirements of the composition theorem.

- $q^{\mathsf{in}} = O(1)$, $\rho^{\mathsf{out}} \geq \delta^{\mathsf{in}} = \rho^{\mathsf{out}}/2$, $\ell^{\mathsf{in}} = 2^{r^{\mathsf{in}}}$.
- $\log W^{\mathsf{out}}_{\mathsf{proof}} = r^{\mathsf{out}}_{\mathsf{col}} + (4/m)\log T(n) \geq r^{\mathsf{out}}_{\mathsf{col}}$.
- $\log W^{\mathsf{out}}_{\mathsf{proof}} = r^{\mathsf{out}}_{\mathsf{col}} + (4/m)\log T(n) \leq r^{\mathsf{out}}_{\mathsf{col}} + r^{\mathsf{out}}_{\mathsf{shared}}$.
- Finally, since $r^{\mathsf{out}} + r^{\mathsf{in}} = r^{\mathsf{out}}_{\mathsf{row}} + r^{\mathsf{out}}_{\mathsf{col}} + r^{\mathsf{out}}_{\mathsf{shared}} + r^{\mathsf{in}} = h^{\mathsf{out}}_{\mathsf{proof}} + w^{\mathsf{out}}_{\mathsf{proof}} + O(\log\log T(n) + m\log m)$, it follows that $H^{\mathsf{out}}_{\mathsf{proof}} \cdot W^{\mathsf{out}}_{\mathsf{proof}} \leq 2^{r^{\mathsf{out}} + r^{\mathsf{in}}}$.

Hence, we can obtain a rectangular PCPP V^{comp} with ROP that has the RNL property. The parameters of the composed PCPP are as follows.

- Soundness error $s^{\mathsf{comp}} := 1 - (1 - s^{\mathsf{out}}) \cdot (1 - s^{\mathsf{in}}) < 1$ that only depends on δ.
- Proximity parameter $\delta^{\mathsf{comp}} := \delta^{\mathsf{out}} = \delta$.
- Row randomness complexity $r^{\mathsf{comp}}_{\mathsf{row}} = r^{\mathsf{out}}_{\mathsf{row}} = h^{\mathsf{out}}_{\mathsf{proof}} - (4/m)\log T(n)$.
- Column randomness complexity $r^{\mathsf{comp}}_{\mathsf{col}} = r^{\mathsf{out}}_{\mathsf{col}} = w^{\mathsf{out}}_{\mathsf{proof}} - (4/m)\log T(n)$.
- Shared randomness complexity $r^{\mathsf{comp}}_{\mathsf{shared}} = r^{\mathsf{out}}_{\mathsf{shared}} + r^{\mathsf{in}} = (7/m)\log T(n) + O(\log\log T(n) + m\log m) + (1/m)\log T(n) + O(\log\log T(n)) = (8/m)\log T(n) + O(\log\log T(n) + m\log m)$.
- Proof matrix height $H^{\mathsf{comp}}_{\mathsf{proof}} = H^{\mathsf{out}}_{\mathsf{proof}} + 2^{r^{\mathsf{out}} + r^{\mathsf{in}}}/W^{\mathsf{out}}_{\mathsf{proof}}$.
- Proof matrix width $W^{\mathsf{comp}}_{\mathsf{proof}} = W^{\mathsf{out}}_{\mathsf{proof}}$.
- Query complexity $q^{\mathsf{comp}} = q^{\mathsf{in}} = O(1)$ that only depends on δ.
- ROP parity check complexity $p^{\mathsf{comp}} = q^{\mathsf{in}} = O(1)$.
- Decision complexity $d^{\mathsf{comp}}(n) = d^{\mathsf{in}}(d^{\mathsf{out}}(n)) = \mathrm{polylog}(T(n))$.
- RNL time complexity $t^{\mathsf{comp}}_{\mathsf{RNL}}(n) = \mathrm{poly}(t^{\mathsf{out}}_{\mathsf{RNL}}(n), \ell^{\mathsf{in}}, d^{\mathsf{in}}) = \mathrm{poly}\big(T(n)^{1/m}\big)$, where $\mathrm{poly}(\cdot)$ hides some absolute constant on the exponent. Note that $t^{\mathsf{out}}_{\mathsf{RNL}}(n) = \mathrm{poly}(\log T(n), m^m) \leq \mathrm{poly}(T(n)^{1/m})$, since $m \leq (\log T(n))^{0.1}$.

Since the RNL for V^{out} can be computed by projections, the RNL for V^{comp} can also be computed by projections.

Smoothening via RNL. Now we apply Theorem 5.1.11 to obtain a smooth and rectangular PCPP V^{smth} with $\mu := (1 - s^{\mathsf{comp}})/2$ and other parameters as follows.

- Soundness error $s^{\mathsf{smth}} := s^{\mathsf{comp}} + \mu < 1$ that only depends on δ.
- Proximity parameter $\delta^{\mathsf{smth}} := \delta^{\mathsf{comp}} = \delta$.
- Row randomness complexity $r^{\mathsf{smth}}_{\mathsf{row}} := r^{\mathsf{comp}}_{\mathsf{row}} = h^{\mathsf{out}}_{\mathsf{proof}} - (4/m)\log T(n)$.
- Column randomness complexity $r^{\mathsf{smth}}_{\mathsf{col}} := r^{\mathsf{comp}}_{\mathsf{col}} = w^{\mathsf{out}}_{\mathsf{proof}} - (4/m)\log T(n)$.
- Shared randomness complexity $r^{\mathsf{smth}}_{\mathsf{shared}} = r^{\mathsf{comp}}_{\mathsf{shared}} = (8/m)\log T(n) + O(\log\log T(n) + m\log m)$.
- Proof matrix width $W^{\mathsf{smth}}_{\mathsf{proof}} = 2^{r^{\mathsf{comp}}_{\mathsf{col}} + r^{\mathsf{comp}}_{\mathsf{shared}}/2} = 2^{w^{\mathsf{out}}_{\mathsf{proof}}} \cdot \mathrm{poly}(\log T(n), m^m)$. Note that here we

set $w^{\mathsf{out}}_{\mathsf{proof}}$ carefully so that $r^{\mathsf{comp}}_{\mathsf{col}} + r^{\mathsf{comp}}_{\mathsf{shared}}/2 = w^{\mathsf{out}}_{\mathsf{proof}} + O(\log\log T(n) + m\log m) = w_{\mathsf{proof}}$. This means that the proof matrix width is exactly $2^{w_{\mathsf{proof}}}$.

- Proof matrix height $H^{\mathsf{smth}}_{\mathsf{proof}} = q^{\mathsf{comp}} \cdot 2^{r^{\mathsf{comp}}_{\mathsf{row}} + r^{\mathsf{comp}}_{\mathsf{shared}}/2} = 2^{h^{\mathsf{out}}_{\mathsf{proof}}} \cdot \mathrm{poly}(\log T(n), m^m)$. Since $q^{\mathsf{comp}} = O(1)$, we can add $O(1)$ dummy queries to the composed PCPP V^{comp} so that q^{comp} becomes a power of two. We then set

$$\begin{aligned} h_{\mathsf{proof}} &= \log H^{\mathsf{smth}}_{\mathsf{proof}} \\ &= h^{\mathsf{out}}_{\mathsf{proof}} + O(\log\log T(n) + m\log m) \\ &= \log T(n) + \Theta(m\log\log T(n)) + O(\log\log T(n) + m\log m) \\ &\quad - (w_{\mathsf{proof}}(n) - O(\log\log T(n) + m\log m)) \\ &= \log T(n) + \Theta(m\log\log T(n)) - w_{\mathsf{proof}}(n). \end{aligned}$$

- Query complexity $q^{\mathsf{smth}} = \mathrm{poly}(q^{\mathsf{comp}}/\mu) = O(1)$ that only depends on δ.
- ROP parity check complexity $p^{\mathsf{smth}} = p^{\mathsf{comp}} = O(1)$.
- Decision complexity $d^{\mathsf{smth}}(n) = \mathrm{poly}(d^{\mathsf{comp}}(n), q^{\mathsf{comp}}/\mu, t^{\mathsf{comp}}_{\mathsf{RNL}}(n)) = \mathrm{poly}\big(T(n)^{1/m}\big)$, where $\mathrm{poly}(\cdot)$ hides some absolute constant on the exponent.

Since the RNL for V^{comp} can be computed by projections, the query indices of V^{smth} can be computed by projections as well.

Reducing the Soundness Error. Finally, we reduce the soundness error of V^{smth} to be s by Theorem 5.1.18, to obtain a smooth and rectangular PCPP with parameters specified as follows.

- Soundness error s.
- Proximity parameter δ.
- Row randomness complexity $h^{\mathsf{out}}_{\mathsf{proof}} - (4/m)\log T(n) \geq h_{\mathsf{proof}} - (5/m)\log T(n)$.
- Column randomness complexity $w^{\mathsf{out}}_{\mathsf{proof}} - (4/m)\log T(n) \geq w_{\mathsf{proof}} - (5/m)\log T(n)$.
- Shared randomness complexity $(8/m)\log T(n) + O(\log\log T(n) + m\log m)$.
- Proof matrix height $2^{h_{\mathsf{proof}}}$ and proof matrix width $2^{w_{\mathsf{proof}}}$.
- Query complexity $q = \mathrm{poly}(q^{\mathsf{smth}}) = O(1)$ that depends on δ and s.
- ROP parity check complexity $\mathrm{poly}(p^{\mathsf{smth}}) = O(1)$ that depends on δ and s.
- Decision complexity $\mathrm{poly}(d^{\mathsf{smth}}(n)) = \mathrm{poly}(T(n)^{1/m})$.

We can move some bits from the row and column randomness to the shared randomness, so that the row and column randomness complexity become exactly $h_{\mathsf{proof}} - (5/m)\log T(n)$ and $w_{\mathsf{proof}} - (5/m)\log T(n)$, respectively, and the shared randomness complexity becomes $(10/m)\log T(n) + O(\log\log T(n) + m\log m)$. Finally, since the query indices of V^{smth} can be computed by projections, the query indices of our final PCPP can also be computed by projections. This completes the construction. □

5.2 Construction of Rectangular PCPPs with Low Query Complexity

Recall that in our framework for solving Range Avoidance and finding hard partial truth tables, the query complexity of the PCPPs will affect the circuit class for which we need to con-

struct satisfying-pair algorithms. Hence, we want (rectangular) PCPPs with query complexity as small as possible. In this section, we construct a rectangular (but not necessarily smooth) PCPP with query complexity 3 and perfect completeness. Furthermore, if we are willing to sacrifice perfect completeness, we can construct a 2-query PCPP with a constant gap between the completeness and soundness parameters.

5.2.1 A 3-Query PCPP

Theorem 5.2.1 ([CW19b], Lemma 24)**.** *For every constant $\delta > 0$, there is a constant $s \in (0,1)$ and a PCP of proximity for* Circuit-Eval *with proximity δ, soundness error s, randomness complexity $O(\log n)$, query complexity $q = 3$, and decision complexity* $\mathrm{polylog}(n)$*. Moreover, the decision predicate is an* OR *of the 3 answers to the queries or their negations.*

We need the following standard composition theorem for PCP of Proximity from [BGH+06] to construct 3-query PCPPs for any pair language in $\mathsf{NTIME}[T(n)]$.

Theorem 5.2.2 ([BGH+06])**.** *Let $r^{\mathsf{out}}, r^{\mathsf{in}}, d^{\mathsf{out}}, d^{\mathsf{in}}, q^{\mathsf{in}} : \mathbb{N} \to \mathbb{N}$ and $\varepsilon^{\mathsf{out}}, \varepsilon^{\mathsf{in}}, \rho^{\mathsf{out}}, \delta^{\mathsf{in}}, \delta^{\mathsf{out}} : \mathbb{N} \to [0,1]$. Suppose that:*

- *Language L has a robust PCPP verifier V^{out} with randomness complexity $r^{\mathsf{out}}(n)$, decision complexity $d^{\mathsf{out}}(n)$, robust soundness error $1 - \varepsilon^{\mathsf{out}}(n)$, robustness parameter $\rho^{\mathsf{out}}(n)$, and proximity parameter $\delta^{\mathsf{out}}(n)$.*
- Circuit-Eval *has a PCPP verifier V^{in} with randomness complexity $r^{\mathsf{in}}(n)$, query complexity $q^{\mathsf{in}}(n)$, decision complexity $d^{\mathsf{in}}(n)$, soundness error $1 - \varepsilon^{\mathsf{in}}(n)$, and proximity parameter $\delta^{\mathsf{in}}(n)$.*
- *$\delta^{\mathsf{in}}(d^{\mathsf{out}}(n)) \leq \rho^{\mathsf{out}}(n)$ for every n.*

Then L has a PCPP Verifier V^{comp} with randomness complexity $r^{\mathsf{out}}(n) + r^{\mathsf{in}}(d^{\mathsf{out}}(n))$, query complexity $q^{\mathsf{in}}(d^{\mathsf{out}}(n))$, decision complexity $d^{\mathsf{in}}(d^{\mathsf{out}}(n))$, soundness error $1 - \varepsilon^{\mathsf{out}}(n) \cdot \varepsilon^{\mathsf{in}}(d^{\mathsf{out}}(n))$, and proximity parameter $\delta^{\mathsf{out}}(n)$. Moreover, the decision predicate of V^{comp} is the same as the decision predicate of V^{in} up to projections on the inputs.

Theorem 5.2.3. *Let L be a pair language in $\mathsf{NTIME}[T(n)]$ for some non-decreasing function $T : \mathbb{Z}^+ \to \mathbb{Z}^+$. For every constant δ, there is a constant $s \in (0,1)$ and a PCP of proximity for L with randomness complexity $\log T(n) + O(\log \log T(n))$, decision complexity $\mathrm{poly}(\log \log T(n))$, soundness error s, proximity parameter δ, and query complexity $q = 3$. Moreover, the decision predicate is an* OR *of the 3 answers to the queries or their negations.*

Proof. Let L be a pair language in $\mathsf{NTIME}[T(n)]$ and $\delta > 0$. We will compose the following two PCPP verifiers using Theorem 5.2.2:

- By Theorem 5.2.1, for every $\delta^{\mathsf{in}} > 0$, there is a constant $s^{\mathsf{in}} \in (0,1)$ and a PCPP verifier V^{in} for Circuit-Eval with randomness complexity $r^{\mathsf{in}} = O(\log n)$, soundness error s^{in}, proximity parameter δ^{in}, query complexity $q^{\mathsf{in}} = 3$, and decision complexity $d^{\mathsf{in}} = \mathrm{polylog}(n)$.

- By Theorem 5.1.19, for all constants $\delta^{\mathsf{out}}, s^{\mathsf{out}} > 0$, there is a constant q^{out} and a PCPP V^{out} for L with randomness complexity $r^{\mathsf{out}} = \log T(n) + O(\log\log T(n))$, soundness error s^{out}, proximity parameter δ^{out}, query complexity q^{out}, and decision complexity $d^{\mathsf{out}} = \mathrm{polylog}(T(n))$. Since $q^{\mathsf{out}} = O(1)$, V^{out} is trivially a robust PCPP with robustness parameter $\rho^{\mathsf{out}} = 1/q^{\mathsf{out}}$.

Fix $\delta^{\mathsf{out}} = \delta$, $s^{\mathsf{out}} = 0.5$, $\delta^{\mathsf{in}} = 1/(2q^{\mathsf{out}})$, and $s = 1 - 0.5 \cdot (1 - s^{\mathsf{in}})$. It is clear that $\delta^{\mathsf{in}}(d^{\mathsf{out}}(n)) \leq \rho^{\mathsf{out}}(n)$. By Theorem 5.2.2, we can obtain a PCPP verifier V^{comp} for L with the following parameters:

- Randomness complexity $r^{\mathsf{out}} + r^{\mathsf{in}}(d^{\mathsf{out}}(n)) = \log T(n) + O(\log\log T(n))$.
- Decision complexity $d^{\mathsf{in}}(d^{\mathsf{out}}(n)) = \mathrm{poly}(\log\log T(n))$.
- Soundness error $1 - (1 - s^{\mathsf{out}}(n)) \cdot (1 - s^{\mathsf{in}}(d^{\mathsf{out}}(n))) = s$.
- Proximity parameter $\delta^{\mathsf{out}}(n) = \delta$.
- Query complexity $q^{\mathsf{in}}(d^{\mathsf{out}}(n)) = 3$.

This satisfies our requirements. □

5.2.2 A 3-Query Rectangular PCPP

Now we construct a 3-query rectangular PCPP by composing the PCPP constructions in Theorem 5.1.3 and Theorem 5.2.3 using the composition theorem (see Theorem 5.1.9).

Theorem 5.2.4 (3-Query Rectangular PCPP)**.** *For every constant $\delta \in (0,1)$, there is a constant $s \in (0,1)$ such that the following holds. Let $m = m(n)$, $T(n)$, $w_{\mathsf{proof}}(n)$, $w_{\mathsf{input}}(n)$ be good functions such that $1 \leq m \leq (\log T(n))^{0.1}$, $n \leq T(n) \leq 2^{\mathrm{poly}(n)}$, $w_{\mathsf{proof}}(n) \leq \log T(n)$, and $w_{\mathsf{input}}(n) \leq \log n$. Then there are good functions $h_{\mathsf{proof}}(n)$ and $h_{\mathsf{input}}(n)$ satisfying*

$$h_{\mathsf{proof}}(n) = \log T(n) + \Theta(m \log\log T(n)) - w_{\mathsf{proof}}(n), \qquad \text{and}$$
$$h_{\mathsf{input}}(n) = \lceil \log n \rceil - w_{\mathsf{input}}(n).$$

such that the following holds.

Suppose that $w_{\mathsf{proof}}, h_{\mathsf{proof}} \geq (5/m)\log T(n)$, and that for some absolute constant $C \geq 1$,

$$\frac{w_{\mathsf{input}}(n)}{w_{\mathsf{proof}}(n)}, \frac{h_{\mathsf{input}}(n)}{h_{\mathsf{proof}}(n)} \leq 1 - \frac{Cm\log\log T(n)}{\log T(n)}.$$

Let $W_{\mathsf{proof}}(n) := 2^{w_{\mathsf{proof}}(n)}$, $H_{\mathsf{proof}}(n) := 2^{h_{\mathsf{proof}}(n)}$, $W_{\mathsf{input}}(n) := 2^{w_{\mathsf{input}}(n)}$, and $H_{\mathsf{input}}(n) := 2^{h_{\mathsf{input}}(n)}$. Then $\mathsf{NTIME}[T(n)]$ has a rectangular PCP of proximity with an $H_{\mathsf{proof}}(n) \times W_{\mathsf{proof}}(n)$ proof matrix and an $H_{\mathsf{input}}(n) \times W_{\mathsf{input}}(n)$ input matrix, whose other parameters are specified in Table 5.6.

Moreover, the total number of queries and parity-check bits is at most 3; and for every $\mathsf{seed.shared}$, the decision predicate $\mathsf{VDec} \leftarrow V_{\mathsf{dec}}(\mathsf{seed.shared})$ of the rectangular PCPP verifier is an OR of its 3 input bits or their negations, where each input is either a query answer or a parity-check bit. Also, the query indices of this PCPP can be computed by projections (in the sense of Definition 2.5.6).

Soundness error	s
Proximity parameter	δ
Row randomness	$h_{\mathsf{proof}} - (5/m)\log T(n)$
Column randomness	$w_{\mathsf{proof}} - (5/m)\log T(n)$
Shared randomness	$(10/m)\log T(n) + O(\log\log T(n) + m\log m)$
Query complexity	3
Parity check complexity	
Decision complexity	$\mathrm{poly}(\log\log T(n))$

Table 5.6: Parameters of the PCPP constructed in Theorem 5.2.4.

Proof. Let $L \in \mathsf{NTIME}[T(n)]$ and $\delta > 0$ be a constant; $m(n)$, $T(n)$, $w_{\mathsf{proof}}(n)$, $h_{\mathsf{proof}}(n)$, $w_{\mathsf{input}}(n)$, $h_{\mathsf{input}}(n)$, and C be defined as above. In one sentence, we compose the robust and rectangular PCPP verifier (Theorem 5.1.3) with the 3-query PCPP verifier (Theorem 5.2.3) using the rectangularity-preserving composition theorem (Theorem 5.1.9).

Outer PCPP. Let $w^{\mathsf{out}}_{\mathsf{proof}}(n) := w_{\mathsf{proof}}(n)$. By Theorem 5.1.3, we can construct a robust and rectangular PCPP verifier V^{out} for L with parameters as follows:

- Robust soundness error $s^{\mathsf{out}} \in (0,1)$ with robustness parameter $\rho^{\mathsf{out}} := 1 - s^{\mathsf{out}}$.
- Proximity parameter $\delta^{\mathsf{out}} := \delta$.
- Proof matrix size $H^{\mathsf{out}}_{\mathsf{proof}}(n) \times W^{\mathsf{out}}_{\mathsf{proof}}(n)$, where $H^{\mathsf{out}}_{\mathsf{proof}} = 2^{h^{\mathsf{out}}_{\mathsf{proof}}}$, $W^{\mathsf{out}}_{\mathsf{proof}} = 2^{w_{\mathsf{proof}}}$, and $h^{\mathsf{out}}_{\mathsf{proof}} = \log T(n) + \Theta(m\log\log T(n)) - w_{\mathsf{proof}}(n)$.
- Row randomness complexity $r^{\mathsf{out}}_{\mathsf{row}} = h^{\mathsf{out}}_{\mathsf{proof}} - (4/m)\log T(n)$.
- Column randomness complexity $r^{\mathsf{out}}_{\mathsf{col}} = w_{\mathsf{proof}} - (4/m)\log T(n)$.
- Shared randomness complexity $r^{\mathsf{out}}_{\mathsf{shared}} = (7/m)\log T(n) + O(\log\log T(n) + m\log m)$.
- Query complexity $q^{\mathsf{out}}(n) = T(n)^{1/m} \cdot \mathrm{polylog}(T(n))$.
- Decision complexity $d^{\mathsf{out}}(n) = T(n)^{1/m} \cdot \mathrm{polylog}(T(n))$.

Moreover, the query indices of V^{out} are computable by projections.

Inner PCPP. Let $\delta^{\mathsf{in}} := \rho^{\mathsf{out}}/2$. By Theorem 5.2.3, there is a constant $s^{\mathsf{in}} \in (0,1)$ and a PCPP verifier V^{in} for Circuit-Eval$^{\perp}$ on input length $d^{\mathsf{out}}(n)$ with randomness complexity $r^{\mathsf{in}} = \log d^{\mathsf{out}}(n) + O(\log\log d^{\mathsf{out}}(n)) = (1/m)\log T(n) + O(\log\log T(n))$, soundness error s^{in}, proximity parameter δ^{in}, query complexity $q = 3$, and decision complexity $d^{\mathsf{in}} = \mathrm{poly}(\log\log d^{\mathsf{out}}(n))$. Without loss of generality, we assume that the proof length is $\ell^{\mathsf{in}} = 2^{r^{\mathsf{in}}}$.

Composition. We now compose V^{out} with the inner PCPP V^{in} by Theorem 5.1.9. We first check that the technical conditions are satisfied.

- $q^{\mathsf{in}} = 3 = O(1)$, $\rho^{\mathsf{out}} \geq \delta^{\mathsf{in}}$, $\ell^{\mathsf{in}} = 2^{r^{\mathsf{in}}}$.
- Since $r^{\mathsf{out}}_{\mathsf{col}} = w^{\mathsf{out}}_{\mathsf{proof}} - (4/m)\log T(n)$ and $r^{\mathsf{out}}_{\mathsf{shared}} \geq (7/m)\log T(n)$, we have that $r^{\mathsf{out}}_{\mathsf{col}} \leq w^{\mathsf{out}}_{\mathsf{proof}} \leq r^{\mathsf{out}}_{\mathsf{col}} + r^{\mathsf{out}}_{\mathsf{shared}}$.
- Since $r^{\mathsf{out}} + r^{\mathsf{in}} = r^{\mathsf{out}}_{\mathsf{row}} + r^{\mathsf{out}}_{\mathsf{col}} + r^{\mathsf{out}}_{\mathsf{shared}} + r^{\mathsf{in}} = h^{\mathsf{out}}_{\mathsf{proof}} + w^{\mathsf{out}}_{\mathsf{proof}} + O(\log\log T(n) + m\log m)$, we have that $H^{\mathsf{out}}_{\mathsf{proof}} \cdot W^{\mathsf{out}}_{\mathsf{proof}} \leq 2^{r^{\mathsf{out}} + r^{\mathsf{in}}}$.

By Theorem 5.1.9, we can obtain a rectangular PCPP V^{comp} with ROP, whose parameters are as follows:

- Soundness error $s^{\mathsf{comp}} = 1 - (1 - s^{\mathsf{in}}) \cdot (1 - s^{\mathsf{out}}) < 1$.

- Proximity parameter $\delta^{\mathsf{comp}} = \delta^{\mathsf{out}} = \delta$.
- Query complexity and ROP parity checking complexity $q^{\mathsf{comp}} = q^{\mathsf{in}} = 3$.
- Proof matrix width $W_{\mathsf{proof}} = W^{\mathsf{out}}_{\mathsf{proof}} = 2^{w_{\mathsf{proof}}}$.
- Proof matrix height $H_{\mathsf{proof}} = H^{\mathsf{out}}_{\mathsf{proof}} + 2^{r^{\mathsf{out}}+r^{\mathsf{in}}}/W^{\mathsf{out}}_{\mathsf{proof}}$. Note that

$$H^{\mathsf{out}}_{\mathsf{proof}} = T(n)\log^{\Theta(m)} T(n)/W^{\mathsf{out}}_{\mathsf{proof}},$$

  hence

$$\begin{aligned} H_{\mathsf{proof}} &= H^{\mathsf{out}}_{\mathsf{proof}} + 2^{h^{\mathsf{out}}_{\mathsf{proof}}+O(\log\log T(n)+m\log m)} \\ &\le H^{\mathsf{out}}_{\mathsf{proof}} \cdot (1+\mathrm{poly}(\log T(n), m^m)) \\ &\le T(n)\log^{\Theta(m)} T(n)/W^{\mathsf{out}}_{\mathsf{proof}}. \end{aligned}$$

  Without loss of generality, we assume that H_{proof} is a power of two. We then define

$$h_{\mathsf{proof}} := \log H_{\mathsf{proof}} = \log T(n) + \Theta(m\log\log T(n)) - w_{\mathsf{proof}}.$$

- Decision complexity $d^{\mathsf{in}}(d^{\mathsf{out}}(n)) = \mathrm{poly}(\log\log T(n))$.

Now we determine the randomness complexity of the composed PCPP verifier. Note that

$$\begin{aligned} r_{\mathsf{shared}} &= r^{\mathsf{out}}_{\mathsf{shared}} + r^{\mathsf{in}} \\ &= (7/m)\log T(n) + O(\log\log T(n) + m\log m) + (1/m)\log T(n) + O(\log\log T(n)) \\ &= (8/m)\log T(n) + O(\log\log T(n) + m\log m), \\ r_{\mathsf{row}} &= r^{\mathsf{out}}_{\mathsf{row}} = h_{\mathsf{proof}} - (4/m)\log T(n) - \Theta(m\log\log T(n)) \ge h_{\mathsf{proof}} - (5/m)\log T(n), \\ r_{\mathsf{col}} &= r^{\mathsf{out}}_{\mathsf{col}} = w_{\mathsf{proof}} - (4/m)\log T(n). \end{aligned}$$

Since we can always move some portion of $\mathsf{seed.row}$ or $\mathsf{seed.col}$ into $\mathsf{seed.shared}$, we can simply assume that $r_{\mathsf{row}} = h_{\mathsf{proof}} - (5/m)\log T(n)$, $r_{\mathsf{col}} = w_{\mathsf{proof}} - (5/m)\log T(n)$, and $r_{\mathsf{shared}} = (10/m)\log T(n) + O(\log\log T(n) + m\log m)$.

By Remark 5.1.10 and the fact that the decision predicate of V^{in} is an OR of the answers or their negations (see Theorem 5.2.3), we know that the total number of queries and parity-check bits of V^{comp} is at most 3, and that for every $\mathsf{seed.shared}$, the decision predicate $\mathsf{VDec} \leftarrow V^{\mathsf{comp}}_{\mathsf{dec}}(\mathsf{seed.shared})$ of V^{comp} is an OR of its input bits (i.e., query answers and parity-check bits) or their negations. Moreover, the query indices of V^{comp} are computable by projections. □

5.2.3 A 2-Query Rectangular PCPP with Imperfect Completeness

Following the construction in [CW19b, Appendix A], we can also construct a 2-query rectangular PCPP with a constant gap between the completeness and soundness parameters, using the following classical gadget due to [GJS76].

Lemma 5.2.5. *Let $x_1, x_2, x_3 \in \{0,1\}$ be Boolean variables. If $x_1 \vee x_2 \vee x_3$, then there is a $y \in \{0,1\}$ such that at least 7 of the following 10 constraints are satisfied:*

$$x_1, x_2, x_3, \overline{x_1} \vee \overline{x_2}, \overline{x_1} \vee \overline{x_3}, \overline{x_2} \vee \overline{x_3}, y, x_1 \vee \overline{y}, x_2 \vee \overline{y}, x_3 \vee \overline{y}. \tag{5.13}$$

Otherwise, at most 6 *of the constraints in* (5.13) *are satisfied for any* $y \in \{0,1\}$. *Moreover, every* $x_1, x_2, x_3, y \in \{0,1\}$ *satisfies at most* 7 *of the above* 10 *constraints.*

Theorem 5.2.6 (2-Query Rectangular PCPP)**.** *For every constant* $\delta \in (0,1)$*, there are constants* $0 < s < c < 1$ *such that the following holds. Let* $m = m(n)$*,* $T(n)$*,* $w_{\mathsf{proof}}(n)$*,* $w_{\mathsf{input}}(n)$ *be good functions such that* $1 \le m \le (\log T(n))^{0.1}$*,* $n \le T(n) \le 2^{\mathrm{poly}(n)}$*,* $w_{\mathsf{proof}}(n) \le \log T(n)$*, and* $w_{\mathsf{input}}(n) \le \log n$*. Then there are good functions* $h_{\mathsf{proof}}(n)$ *and* $h_{\mathsf{input}}(n)$ *satisfying*

$$h_{\mathsf{proof}}(n) = \log T(n) + \Theta(m \log\log T(n)) - w_{\mathsf{proof}}(n), \qquad \textit{and}$$
$$h_{\mathsf{input}}(n) = \lceil \log n \rceil - w_{\mathsf{input}}(n),$$

such that the following holds.

Suppose that $w_{\mathsf{proof}}, h_{\mathsf{proof}} \ge (5/m)\log T(n)$*, and that for some absolute constant* $C \ge 1$*,*

$$\frac{w_{\mathsf{input}}(n)}{w_{\mathsf{proof}}(n)}, \frac{h_{\mathsf{input}}(n)}{h_{\mathsf{proof}}(n)} \le 1 - \frac{Cm \log\log T(n)}{\log T(n)}.$$

Let $W_{\mathsf{proof}}(n) := 2^{w_{\mathsf{proof}}(n)}$*,* $H_{\mathsf{proof}}(n) := 2^{h_{\mathsf{proof}}(n)}$*,* $W_{\mathsf{input}}(n) := 2^{w_{\mathsf{input}}(n)}$*, and* $H_{\mathsf{input}}(n) := 2^{h_{\mathsf{input}}(n)}$*. Then* $\mathsf{NTIME}[T(n)]$ *has a rectangular PCP of proximity with an* $H_{\mathsf{proof}}(n) \times W_{\mathsf{proof}}(n)$ *proof matrix and an* $H_{\mathsf{input}}(n) \times W_{\mathsf{input}}(n)$ *input matrix, whose other parameters are specified in Table 5.7.*

Moreover, given the randomness $\mathsf{seed} \in \{0,1\}^r$*, the total number of queries and parity-check bits is at most* 2*, and the decision predicate* $\mathsf{VDec} \leftarrow V_{\mathsf{dec}}(\mathsf{seed.shared})$ *of the rectangular PCPP verifier is an* OR *of the* 2 *input bits (including queries and parity-check bits) or their negations for every* seed.shared*. Also, the query indices of this PCPP can be computed by projections.*

Completeness error	$1 - c$
Soundness error	s
Proximity parameter	δ
Row randomness	$h_{\mathsf{proof}} - (5/m)\log T(n)$
Column randomness	$w_{\mathsf{proof}} - (5/m)\log T(n)$
Shared randomness	$(10/m)\log T(n) + O(\log\log T(n) + m\log m)$
Query complexity	2
Parity check complexity	
Decision complexity	$\mathrm{poly}(\log\log T(n))$

Table 5.7: Parameters of the PCPP constructed in Theorem 5.2.6.

Proof. Let $\delta \in (0,1)$ be arbitrary. By Theorem 5.2.4, there is a rectangular PCPP verifier V^{3q} with perfect completeness and parameters:

- Soundness error $s^{\mathsf{3q}} \in (0,1)$.
- Proximity parameter δ.
- Query complexity and parity-check complexity 3.
- Proof matrix size $H^{\mathsf{3q}}_{\mathsf{proof}} \times W^{\mathsf{3q}}_{\mathsf{proof}}$, where $W^{\mathsf{3q}}_{\mathsf{proof}} = 2^{w^{\mathsf{3q}}_{\mathsf{proof}}}$, $H^{\mathsf{3q}}_{\mathsf{proof}} = 2^{h^{\mathsf{3q}}_{\mathsf{proof}}}$, $w^{\mathsf{3q}}_{\mathsf{proof}} = w_{\mathsf{proof}}$, and $h^{\mathsf{3q}}_{\mathsf{proof}} = \log T(n) + \Theta(m \log\log T(n)) - w_{\mathsf{proof}}(n)$.[9]

[9]Note that the final matrix height is $h_{\mathsf{proof}} \le h^{\mathsf{3q}}_{\mathsf{proof}} + O(\log\log T(n))$, hence the technical requirement

- Shared randomness complexity $r_{\mathsf{shared}} = (10/m)\log T(n) + O(\log\log T(n) + m\log m)$.
- Row randomness complexity $r_{\mathsf{row}} = h^{\mathsf{3q}}_{\mathsf{proof}} - (5/m)\log T(n)$.
- Column randomness complexity $r_{\mathsf{col}} = w^{\mathsf{3q}}_{\mathsf{proof}} - (5/m)\log T(n)$.
- Decision complexity $\mathrm{poly}(\log\log T(n))$.

Let $r^{\mathsf{3q}} := r_{\mathsf{row}} + r_{\mathsf{col}} + r_{\mathsf{shared}}$ be the length of total randomness. Moreover, we know that the total number of queries and parity-check bits is at most 3, and that the decision circuit of V is an OR of its input bits (i.e. the answers to the queries and parity-check bits) or their negations after fixing the random seed. We will now combine V^{3q} and the gadget in Lemma 5.2.5 to construct a 2-query PCPP.

Suppose, for the simplicity of presentation, that the PCPP verifier V always probes 2 bits of the input and proof oracles, and has 1 parity-check bit. (The other cases can be considered similarly and we omit the details.) Then the decision predicate $\mathsf{VDec} \leftarrow V_{\mathsf{dec}}(\mathsf{seed.shared})$ for every fixed $\mathsf{seed.shared} \in \{0,1\}^{r_{\mathsf{shared}}}$ is a function of the form

$$\mathsf{VDec}(\mathsf{ans}_1, \mathsf{ans}_2, pc_1(\mathsf{seed})) := (\mathsf{ans}_1 \oplus b_1) \vee (\mathsf{ans}_2 \oplus b_2) \vee (pc_1(\mathsf{seed}) \oplus b_3).$$

where $b_1, b_2, b_3 \in \{0,1\}$. The new PCPP verifier V is defined as follows.

- The proof of the new PCPP verifier V is the concatenation of the proof for V^{3q} and an $y : \{0,1\}^{r^{\mathsf{3q}}} \to \{0,1\}$ of length $2^{r^{\mathsf{3q}}}$ used as the additional variable y in Lemma 5.2.5.
- The randomness of V is the concatenation of the randomness seed for V^{3q} and a $j \in [10]$.

Queries and parity-check bits. Assume that $(\mathsf{seed}, j) \in \{0,1\}^{r^{\mathsf{3q}}} \times [10]$ is given as the randomness. The verifier V first generates the indices i_1, i_2 of the queries to the input and proof oracles (denoted by a single oracle Π for simplicity) and the parity-check function pc_1. Instead of making all these queries and doing the parity-check, we identify $\mathsf{ans}_1 \oplus b_1, \mathsf{ans}_2 \oplus b_2, pc_1(\mathsf{seed}) \oplus b_3, y(\mathsf{seed})$ with x_1, x_2, x_3, y in the gadget given by Lemma 5.2.5, respectively, and query the j-th gadget. (For instance, if $j = 5$, the corresponding constraint is $\overline{x_1} \vee \overline{x_3}$, so that we will query the i_1-th of Π and compute the parity-check pc_1; if $j = 8$, the constraint is $x_1 \vee \overline{y}$, so that we will query the i_1-th bit of Π and the entry $y(\mathsf{seed})$.) The decision predicate will accept if and only if either the j-th constraint is satisfied when identifying $\mathsf{ans}_1 \oplus b_1, \mathsf{ans}_2 \oplus b_2, pc_1(\mathsf{seed}) \oplus b_3, y(\mathsf{seed})$ with x_1, x_2, x_3, y, respectively.

Completeness. For every input $x \in L$, by the completeness of V^{3q}, there is a proof oracle $\Pi^{\mathsf{3q}}_{\mathsf{proof}}$ such that V^{3q} accepts given the oracle $x \circ \Pi^{\mathsf{3q}}_{\mathsf{proof}}$ with probability 1, which means that for every $\mathsf{seed} \in \{0,1\}^{r^{\mathsf{3q}}}$, the answers $\mathsf{ans}_1, \mathsf{ans}_2$ to the queries and the parity-check bits $pc_1(\mathsf{seed})$ satisfies

$$\mathsf{VDec}(\mathsf{ans}_1, \mathsf{ans}_2, pc_1(\mathsf{seed})) = (\mathsf{ans}_1 \oplus b_1) \vee (\mathsf{ans}_2 \oplus b_2) \vee (pc_1(\mathsf{seed}) \oplus b_3) = 1.$$

By Lemma 5.2.5, there is an y_{seed} such that at least 7 of the 10 constraints in the gadgets are satisfied. This means that given the proof oracle $\Pi^{\mathsf{3q}}_{\mathsf{proof}} \circ y_{\mathsf{seed}}$, the verifier will accept with probability at least $1 - c$ where $c := 3/10$.

$h_{\mathsf{input}}(n)/h^{\mathsf{3q}}_{\mathsf{proof}} \le 1 - C' \log\log T(n)/\log T(n)$ for large C' holds, given the assumption that $h_{\mathsf{input}}(n)/h_{\mathsf{proof}} \le 1 - C\log\log T(n)/\log T(n)$ for large C, $h_{\mathsf{proof}} \ge (5/m)\log T(n)$, and $m \le (\log T(n))^{0.1}$, as shown in Claim 5.1.20.

Soundness. Assume that $x \in \{0,1\}^n$ that is δ-far from being in L, and $\Pi_{\mathsf{proof}} = \Pi^{\mathsf{3q}}_{\mathsf{proof}} \circ y$ is any proof, where $\Pi^{\mathsf{3q}}_{\mathsf{proof}}$ is a proof for V^{3q} and $y : \{0,1\}^{r^{\mathsf{3q}}} \to \{0,1\}$. By the soundness of V^{3q}, we know that for at least $1 - s^{\mathsf{3q}}$ fraction of $\mathsf{seed} \in \{0,1\}^n$,

$$\mathsf{VDec}(\mathsf{ans}_1, \mathsf{ans}_2, pc_1(\mathsf{seed})) = (\mathsf{ans}_1 \oplus b_1) \vee (\mathsf{ans}_2 \oplus b_2) \vee (pc_1(\mathsf{seed}) \oplus b_3) = 0.$$

By Lemma 5.2.5, we can see that for these seed, the accept probability of V is at most $6/10$, whereas in other cases the accept probability of V is at most $7/10$. Thus the accept probability of V is at most $s := (7/10) \cdot s^{\mathsf{3q}} + (6/10) \cdot (1 - s^{\mathsf{3q}}) < c$.

Rectangularity. Since we only need to introduce $O(1)$ bits of randomness representing $j \leftarrow [10]$, we can put it into the shared randomness. We only need to show that the new proof $\Pi^{\mathsf{3q}}_{\mathsf{proof}} \circ y$ can be arranged as a matrix so that the queries can be done rectangularly.

Note that $r^{\mathsf{3q}} = h^{\mathsf{3q}}_{\mathsf{proof}} + w^{\mathsf{3q}}_{\mathsf{proof}} + O(\log\log T + m \log m)$. Let $h_{\mathsf{proof}} := r^{\mathsf{3q}} - W^{\mathsf{3q}}_{\mathsf{proof}} + 1$ and $H_{\mathsf{proof}} := 2^{h_{\mathsf{proof}}}$. The final proof matrix will be the concatenation of two matrices of size $(2^{r^{\mathsf{3q}}}/W^{\mathsf{3q}}_{\mathsf{proof}}) \times W^{\mathsf{3q}}_{\mathsf{proof}}$ each. The first matrix contains the original $H^{\mathsf{3q}}_{\mathsf{proof}} \times W^{\mathsf{3q}}_{\mathsf{proof}}$ proof of V^{3q} (that is, only the first $H^{\mathsf{3q}}_{\mathsf{proof}}$ rows of this matrix will be queried; note that $H^{\mathsf{3q}}_{\mathsf{proof}} \le (2^{r^{\mathsf{3q}}}/W^{\mathsf{3q}}_{\mathsf{proof}})$). The second matrix contains the truth table of $y : \{0,1\}^{r^{\mathsf{3q}}} \to \{0,1\}$. Recall that there are two kinds of queries to the proof oracle.

1. If the query is to the proof oracle $\Pi^{\mathsf{3q}}_{\mathsf{proof}}$ of V^{3q} or to the input oracle, we can use the row and column verifier of V^{3q} to generate the queries rectangularly.

2. Otherwise, the query is to the proof $y(\mathsf{seed})$ for the randomness $\mathsf{seed} \in \{0,1\}^r$ of V^{3q}. Then the column (resp. row) index of this query only depends on the lowest w_{proof} bits (resp. the highest $r - w_{\mathsf{proof}}$ bits) of the random seed of V. Recall that the random seed of V is the concatenation of seed and a $j \in [10]$. If we arrange the randomness as

   $$\mathsf{seed.col} \circ \mathsf{seed.shared} \circ j \circ \mathsf{seed.row},$$

   then the lowest w_{proof} bits (resp. the highest $r - w_{\mathsf{proof}}$ bits) of the random seed only depends on the $(\mathsf{seed.col}, \mathsf{seed.shared})$ (resp. $(\mathsf{seed.shared}, j, \mathsf{seed.row})$), since

   $$\begin{aligned} r_{\mathsf{col}} &= w_{\mathsf{proof}} - (5/m)\log T(n) \le w_{\mathsf{proof}} \\ r_{\mathsf{col}} + r_{\mathsf{shared}} &= w_{\mathsf{proof}} + (5/m)\log T(n) + O(\log\log n + m \log m) \ge w_{\mathsf{proof}}. \end{aligned}$$

As a result, the queries can be done rectangularly. Clearly, in both cases, the query indices can be computed by a projection. □

Chapter 6

Polynomial-Time Pseudodeterministic Constructions

6.1 Introduction

How hard is it to construct an n-bit prime[1]? This is a fundamental problem in number theory and in complexity theory. Under reasonable assumptions, the problem is solvable in deterministic polynomial time. In more detail, Cramér's conjecture [Cra36] in number theory asserts that the largest prime gap in any consecutive sequence of n-bit numbers is $O(n^2)$. Assuming this conjecture, we can solve the prime construction problem efficiently by testing the first $O(n^2)$ integers greater than 2^{n-1} for primality and outputting the first one, where the primality tests are done efficiently using the algorithm of Agrawal, Kayal, and Saxena [AKS04]. An independent source of evidence for the efficiency of prime construction is the complexity-theoretic conjecture that $\mathsf{DTIME}(2^{O(n)})$ requires Boolean circuits of exponential size on almost all input lengths. Under this conjecture, we can use the Impagliazzo–Wigderson pseudorandom generator [IW97] to *derandomise* the simple randomised algorithm that outputs a random n-bit number, using the facts that primality testing is in polynomial time and that an $\Omega(1/n)$ fraction of n-bit numbers are prime.

However, we seem very far from either settling Cramér's conjecture or proving strong complexity lower bounds. The best upper bound we can prove on the gap between consecutive n-bit primes is $2^{(0.525+o(1))n}$ [BHP01], and no super-linear circuit lower bounds are known for $\mathsf{DTIME}(2^{O(n)})$ [LY22]. Indeed, the best unconditional result we have so far is that deterministic prime construction can be done in time $2^{(0.5+o(1))n}$ [LO87], which is very far from the polynomial-time bound we seek. The Polymath 4 project (see [TCH12]) sought to improve this upper bound using number-theoretic techniques but did not achieve an unconditional improvement.

In contrast to the situation with deterministic prime construction, it is easy to generate an n-bit prime *randomly*, as mentioned above: simply generate a random n-bit number, test it for primality in polynomial time, and output it if it is a prime. This algorithm has success probability $\Omega(1/n)$ by the Prime Number Theorem, and the success probability can be amplified to be exponentially close to 1 by repeating the process $\mathrm{poly}(n)$ times independently, and outputting the first of these $\mathrm{poly}(n)$ numbers that is verified to be prime, assuming that there is at least

[1] Recall that a positive integer q is an n-bit prime if q is a prime number and $2^{n-1} \leq q \leq 2^n - 1$.

one.

Gat and Goldwasser [GG11] asked whether it is possible to generate primes efficiently by a randomised process, such that the output is essentially *independent* of the randomness of the algorithm. In other words, is there a polynomial-time randomised algorithm, which on input 1^n, constructs a *canonical* prime of length n with high probability? They call such an algorithm a *pseudodeterministic* algorithm, since the output of the algorithm is (almost) deterministic even though the algorithm might use random bits in its operation. Note that the randomised algorithm for prime generation we described in the previous paragraph is very far from being pseudodeterministic, as different runs of the algorithm are unlikely to produce the same prime. It is easy to see that a pseudodeterministic construction serves as an intermediate notion between a randomised construction (which is trivial for primes) and a deterministic construction (where little progress has been made so far).

[GG11] initiated a general theory of pseudodeterminism for search problems, motivated by applications in cryptography and distributed computing. Since then, there have been a number of papers on pseudodeterminism, in various contexts, such as query complexity [GGR13, GIPS21, CDM23], streaming algorithms [GGMW20, BKKS23], parallel computation [GG17, GG21], learning algorithms [OS18], Kolmogorov complexity [Oli19, LOS21], space-bounded computation [GL19], proof systems [GGH18, GGH19], number theory and computational algebra [Gro15, OS17b], approximation algorithms [DPV18], and many other settings (see, *e.g.*, [BB18, Gol25, DPV21, DPWV22, WDP+22, CPW23]).

Despite all this progress, the main problem about pseudodeterminism posed in [GG11] has remained open: Is there a pseudodeterministic polynomial-time algorithm for prime construction? They describe this problem as "the most intriguing" and "perhaps the most compelling challenge for finding a unique output".

Unlike in the case of deterministic construction, number-theoretic techniques have so far not proven useful for the pseudodeterministic construction problem for primes. Using complexity-theoretic techniques, Oliveira and Santhanam [OS17b] (see also [LOS21]) showed that for any $\varepsilon > 0$, there is an algorithm that runs in time 2^{n^ε} and succeeds on infinitely many input lengths.

6.1.1 Our Results

We design a significantly faster algorithm and provide an affirmative answer to the question posed by Gat and Goldwasser in the infinitely-often regime. Our main result can be stated in full generality as follows.

Theorem 6.1.1 (Infinitely-Often Polynomial-Time Pseudodeterministic Constructions)**.** *Let $Q \subseteq \{0,1\}^*$ be a language with the following properties:*

(Density.) *there is a constant $\rho \geq 1$ such that for every $n \in \mathbb{N}_{\geq 1}$, $Q_n := Q \cap \{0,1\}^n$ satisfies $|Q_n| \geq n^{-\rho} \cdot 2^n$; and*

(Easiness.) *there is a deterministic polynomial-time algorithm A_Q that decides whether an input $x \in \{0,1\}^*$ belongs to Q.*

Then there exist a probabilistic polynomial-time algorithm B and a sequence $\{x_n\}_{n \in \mathbb{N}_{\geq 1}}$ of n-bit strings in Q such that the following conditions hold:

1. *On every input length* $n \in \mathbb{N}_{\geq 1}$, $\Pr_B[B(1^n) \notin \{x_n, \perp\}] \leq 2^{-n}$.

2. *On infinitely many input lengths* $n \in \mathbb{N}_{\geq 1}$, $\Pr_B[B(1^n) = x_n] \geq 1 - 2^{-n}$.

Interestingly, our construction is "non-black-box", in the sense that changing the *code* of the algorithm A_Q deciding property Q affects the canonical output of the corresponding algorithm B. We will revisit this point when we discuss our techniques (see the remark at the end of Section 6.1.3).

Letting Q be the set of prime numbers and noticing that Q is both dense (by the Prime Number Theorem) and easy (by the AKS primality test [AKS04]), we immediately obtain the following corollary of Theorem 6.1.1.

Corollary 6.1.2 (Infinitely-Often Polynomial-Time Pseudodeterministic Construction of Primes)**.** *There is a randomised polynomial-time algorithm B such that, for infinitely many values of n, $B(1^n)$ outputs a canonical n-bit prime p_n with high probability.*

Corollary 6.1.2 improves upon the subexponential-time infinitely-often pseudodeterministic construction of primes from [OS17b] mentioned above. Note that the result for prime construction is a corollary of a far more general result about properties that are dense and easy. This is evidence of the surprising power of complexity theory when applied to a problem which seems to be about number theory (but where number-theoretic techniques have not so far been effective). The famous efficient primality testing algorithm of [AKS04] similarly applied complexity-theoretic derandomisation ideas to solve a longstanding open problem in computational number theory, though their argument does require more information about primes.

For a string $w \in \{0,1\}^*$ and $t\colon \mathbb{N} \to \mathbb{N}$, we let $\mathsf{rK}^t(w)$ denote the length of the smallest randomised program that runs for at most $t(|w|)$ steps and outputs w with probability at least $2/3$. (We refer to [LO22] for a formal definition and for an introduction to probabilistic notions of time-bounded Kolmogorov complexity.) By encoding the (constant-size) randomised polynomial-time algorithm B and each good input length n using $O(1) + \log n$ bits in total, the following result holds.

Corollary 6.1.3 (Infinitely Many Primes with Efficient Succinct Descriptions)**.** *There is a constant $c \geq 1$ such that, for $t(n) = n^c$, the following holds. For every $m \geq 1$, there is $n > m$ and an n-bit prime p_n such that $\mathsf{rK}^t(p_n) \leq \log(n) + O(1)$.*

In other words, there are infinitely many primes that admit very short efficient descriptions. The bound in Corollary 6.1.3 improves upon the sub-polynomial bound on $\mathsf{rK}^{\mathrm{poly}}(p_n)$ from [LOS21].

In the next subsection, we describe at a high level the ideas in the proof of Theorem 6.1.1 and how they relate to previous work.

6.1.2 Proof Ideas

The proof of Theorem 6.1.1 relies on *uniform hardness-randomness trade-offs* [IW01, TV07]. For concreteness, assume that $Q = \{Q_n\}_{n \in \mathbb{N}_{\geq 1}}$, with each $Q_n \subseteq \{0,1\}^n$ consisting of the set of n-bit prime numbers. Let A_Q be a deterministic polynomial-time algorithm that decides Q (*e.g.*, A_Q is the AKS primality test algorithm [AKS04]). Before we present our algorithm and the main

ideas underlying our result, it is instructive to discuss the approach of [OS17b], which provides a subexponential-time pseudodeterministic construction that succeeds on infinitely many input lengths.

Subexponential-time constructions of [OS17b]. We first recall how uniform hardness-randomness trade-offs work. Given a presumed hard language L, a uniform hardness-randomness trade-off for L states that either L is easy for probabilistic polynomial-time algorithms, or else we can build a *pseudorandom set* $G_n \subseteq \{0,1\}^n$ computable in subexponential time (thus also has subexponential size), which fools probabilistic polynomial-time algorithms on inputs of length n (for infinitely many n). In particular, Trevisan and Vadhan [TV07] give a uniform hardness-randomness trade-off for a PSPACE-complete language L_{TV} they construct, which has certain special properties tailored to uniform hardness-randomness trade-offs.[2]

The subexponential-time construction in [OS17b] uses a *win-win* argument to derive an *unconditional* pseudodeterministic algorithm from the uniform hardness-randomness trade-off of [TV07]. There are two cases: either $L_{\mathsf{TV}} \in \mathsf{BPP}$, or it is not. If the former is the case, then $\mathsf{PSPACE} \subseteq \mathsf{BPP}$ by the PSPACE-completeness of L_{TV}. Now, since we can in *polynomial space* test all n-bit numbers using A_Q until we find the lexicographic first prime number, we can also do it in *randomised polynomial time*, *i.e.*, there is a randomised algorithm $B(1^n)$ that runs in polynomial time and outputs the lexicographically first n-bit prime with high probability. Thus, in this case, the lexicographically first n-bit prime is the "canonical" output of the pseudodeterministic algorithm, and the algorithm works on *every* input length n.

Suppose, on the other hand, that $L_{\mathsf{TV}} \notin \mathsf{BPP}$. Using the uniform hardness-randomness trade-off of [TV07], we have that for each $\varepsilon > 0$, there is a pseudorandom set $G = \{G_n\}$, where each $G_n \subseteq \{0,1\}^n$ is of size at most 2^{n^ε}, such that for infinitely many n, G_n fools the algorithm A_Q on inputs of length n. Since A_Q accepts an $\Omega(1/n)$ fraction of strings of length n by the Prime Number Theorem, we have that the fraction of strings in G_n that are prime is $\Omega(1/n)$ (by choosing the error parameter of the uniform hardness-randomness trade-off to be small enough). In particular, there must exist an element of G_n that is prime. Since G_n is computable in subexponential time, we can define a subexponential time *deterministic* algorithm that enumerates elements of G_n and tests each one for primality until it finds and outputs one that is prime. This algorithm is deterministic, but it runs in subexponential time, and is only guaranteed to be correct for infinitely many n.

Thus, in either case, we have a pseudodeterministic algorithm for constructing primes that runs in subexponential time and works infinitely often. Note that we do not know a priori which of the two cases above holds, and therefore the argument is somewhat non-constructive. By exploiting further properties of the uniform hardness-randomness trade-off, [OS17b] managed to give an explicit construction algorithm that runs in subexponential time infinitely often.

Win-win arguments. The above argument gives a subexponential-time construction, but the win-win structure of the argument seems incapable of giving an optimal polynomial-time construction. Indeed, this is the case for many win-win arguments used in complexity theory:

[2] For the pseudorandomness experts, these special properties are *downward self-reducibility* and *random self-reducibility*.

- A win-win argument based on the Karp–Lipton theorem [KL80] gives that $\Sigma_2\mathsf{EXP}$ requires super-polynomial size Boolean circuits [Kan82], but seems incapable of giving truly exponential ($2^{\Omega(n)}$) Boolean circuit lower bounds.[3]

- A win-win argument based on uniform hardness-randomness trade-offs gives that either $\mathsf{E} \subseteq \mathsf{BPP}$ or BPP can be simulated infinitely often in deterministic subexponential time on average [IW01], but it remains unknown if such a trade-off holds at the "high end", *i.e.*, whether it is the case that either E is in probabilistic subexponential-time or else BPP can be simulated infinitely often in deterministic polynomial time on average.

- A win-win argument based on the Easy Witness Lemma gives that if $\mathsf{NEXP} \subseteq \mathsf{SIZE}(\text{poly})$, then $\mathsf{NEXP} = \mathsf{MA}$ [IKW02], but it is unknown if any interesting uniform collapse follows from the simulation of NEXP by subexponential-size Boolean circuits.

In each of these cases, the win-win argument seems to have inherent limitations that prevent us from getting optimal lower bounds or trade-offs. Indeed, a paper by Miltersen, Vinodchandran, and Watanabe [MVW99] studies the "fractional exponential" lower bounds that seem to be the best provable using win-win arguments in the context of Boolean circuit lower bounds for exponential-time classes.[4]

Thus, in order to obtain a polynomial-time pseudodeterministic algorithm for primality, it seems that we need to go beyond win-win arguments. One natural idea is to apply uniform hardness-randomness trade-offs *recursively*. However, this seems hard to do with the uniform hardness-randomness trade-off of [TV07]. Their trade-off applies only to the special language L_{TV}. If we argue based on the hardness or other properties of L_{TV}, then in the case where $L_{\mathsf{TV}} \in \mathsf{BPP}$, we get a pseudodeterministic polynomial-time algorithm for constructing primes, but in the case where $L_{\mathsf{TV}} \notin \mathsf{BPP}$, we get a subexponential-time constructible pseudorandom set, and it is unclear how to apply the uniform hardness-randomness trade-off to the algorithm for constructing this set.

Recursive application of uniform hardness-randomness trade-offs. One of our main ideas is to exploit very recent work on uniform hardness-randomness trade-offs [CT21a] which applies to *generic* computations, as long as they satisfy certain mild properties. These trade-offs yield *hitting sets* rather than pseudorandom sets based on hardness — a hitting set $\mathsf{H} \subseteq \{0,1\}^M$ is a set that has non-empty intersection with every $Q_M \subseteq \{0,1\}^M$ that is dense (*i.e.*, accepts at least a $1/\text{poly}(M)$ fraction of strings) and is efficiently computable. It turns out that for our application to pseudodeterministic algorithms, uniform hardness-randomness trade-offs that yield hitting sets are sufficient.

Specifically, Chen and Tell [CT21a] show that for any multi-output function $f\colon \{1^n\} \to \{0,1\}^n$ computed by uniform Boolean circuits of size $T = T(n)$ and depth $d = d(n)$, either there

[3] Partially inspired by our results, subsequent works [CHR24, Li24] proved near-maximum circuit lower bounds for the classes $\Sigma_2\mathsf{E}$ and $\mathsf{S}_2\mathsf{E}$.

[4] For example, a function $f : \mathbb{N} \to \mathbb{N}$ is *sub-half-exponential* if $f(f(n)^c)^c \leq O(2^n)$ for every constant c. (The exact definition of sub-half-exponential functions may be different in different papers.) Functions such as n^k and $2^{\log^k n}$ are sub-half-exponential, while $2^{\varepsilon n}$ and 2^{n^ε} are not. It is known that $\Sigma_2\mathsf{EXP}$ cannot be computed by $f(n)$-size circuits for every sub-half-exponential f, but it remains open to show that $\Sigma_2\mathsf{EXP}$ requires circuit complexity 2^{n^ε} for any constant $\varepsilon > 0$.

is a hitting set $\mathsf{H} \subseteq \{0,1\}^M$ computable in time $\mathrm{poly}(T)$, or $f(1^n)$ can be computed with high probability in time $(d+n)\cdot\mathrm{poly}(M)$ (which could be much less than T). Note that this trade-off is applicable to *any* multi-output function f given bounds on its uniform circuit complexity.

Our key idea is that this more generic uniform hardness-randomness trade-off can be applied *recursively*. Indeed, we apply it to multi-output functions which capture the very task we are trying to solve, *i.e.*, constructing a prime! In our base case, we use the function f which does a brute-force search over n-bit numbers and outputs the lexicographically first one that is prime. This function can be computed by uniform Boolean circuits of size $2^{O(n)}$ and depth $\mathrm{poly}(n)$, and hence we can apply the Chen–Tell trade-off to it. We set $M = n^\beta$ for some large enough constant $\beta > 1$ in the trade-off. If we have that $f(1^n)$ is computable with high probability in time $(d+n)\cdot\mathrm{poly}(M)$, then we are done, since this gives us a pseudodeterministic algorithm for primes at length n. If not, we have that there is a hitting set $\mathsf{H} \subseteq \{0,1\}^{n^\beta}$ computable in time $2^{O(n)}$. In particular, by iterating over the elements of H and outputting the first one that is prime, we gain over the naïve brute-force search algorithm, since we are now outputting a prime of length n^β in time $2^{O(n)}$. Now *this* new algorithm can be captured by a multi-output function with output length n^β to which we apply the Chen–Tell trade-off again. In each recursive step, either we obtain a pseudodeterministic polynomial-time construction of primes, or we obtain a significantly faster deterministic construction of primes (of a larger input length). Intuitively, analyzing this process after $O(\log n)$ steps of recursion, we can hope to show that at least one of the steps leads to a polynomial-time pseudodeterministic algorithm at the input length considered at that step.

This doesn't quite work as stated because the Chen–Tell trade-off uses the Nisan–Wigderson generator [NW94], which is not known to have optimal parameters for all levels of hardness.[5] Our recursive process explores essentially all possible levels of hardness for the uniform hardness-randomness trade-off, since each recursive step corresponds to a different level of hardness. Using the original Chen–Tell trade-off gives a *quasi-polynomial-time* pseudodeterministic construction, but in order to get a polynomial-time pseudodeterministic construction, we need to work harder.

Another crucial idea for us is to optimise the Chen–Tell trade-off by using the Shaltiel–Umans generator [SU05] rather than the Nisan–Wigderson generator. This idea comes with its own implementation challenges, since the Shaltiel–Umans generator is not known to possess a crucial learnability property required for the uniform hardness-randomness trade-off. We sidestep this issue using a further win-win analysis, together with some other tricks; see Section 6.1.3 for details. This enables us to achieve an optimal polynomial-time pseudodeterministic construction on infinitely many input lengths, and thereby establish Theorem 6.1.1.[6] We note that the subexponential-time construction of [OS17b] also only works for infinitely many input lengths, and it is still open even to get a subexponential-time construction that works on all input lengths.

The intuitive description here does not address several subtleties that arise in the proof, such as maintaining the right uniformity and depth conditions when recursively applying the uniform hardness-randomness trade-off. We refer to Section 6.1.3 for a more detailed discussion of such

[5] Informally speaking, given a "hard truth table" of length T, we want to construct a hitting set $\mathsf{H} \subseteq \{0,1\}^M$ in $\mathrm{poly}(T)$ time; however, the Nisan–Wigderson generator requires $2^{\Theta(\log^2 T/\log M)}$ time to construct.

[6] While we do not explore this direction in the current work, we believe that our improvement on the Chen-Tell trade-off can be used to improve the trade-off from [CRT22, Theorem 5.2 and Theorem 5.3], thus getting a better uniform hardness vs randomness connection in the low-end regime.

matters.

6.1.3 Technical Overview

As explained above, we consider a chain of $t = O(\log n)$ recursively defined (candidate) HSGs $\mathsf{H}_0, \mathsf{H}_1, \ldots, \mathsf{H}_t$ operating over different input lengths. These HSGs are obtained from the recent construction by Chen and Tell [CT21a], which we informally describe next. Recall that we use Q_M to denote the easy and dense property over inputs of length M.

The Chen–Tell [CT21a] targeted HSG ("ideal version"). Let $c \geq 1$ be a large enough constant, and let $f\colon \{1^n\} \to \{0,1\}^n$ be a family of unary functions computed by (uniform) Boolean circuits of size $T = T(n)$ and depth $d = d(n)$. Then, for every $\log T \leq M \leq T$ there is a set $\mathsf{H} \subseteq \{0,1\}^M$ computable in

$$\text{time } \widetilde{T} := T^c \quad \text{and} \quad \text{depth } \widetilde{d} := d \cdot \log(T) + M^c$$

such that, if $Q_M \subseteq \{0,1\}^M$ *avoids* H, (*i.e.*, Q_M is dense but $Q_M \cap \mathsf{H} = \varnothing$), then we can compute $f(1^n)$ with high probability in time $(d+n) \cdot M^c$.

In other words, if f admits *low-depth* circuits, we can construct a candidate HSG H over length-M inputs such that breaking the generator H allows us to compute $f(1^n)$ in $\mathrm{poly}(n, d, M)$ time. For $d, M \ll T$, this can be much faster than the original time T required to compute f.

The statement above differs from the results in [CT21a] (stated for unary functions) in two important ways. First, the claimed upper bound on $\widetilde{T}$ (the running time of the HSG) is not obtained by [CT21a] for all choices of M. Secondly, we have not formally specified the *uniformity* of the family of circuits computing f. While these are crucial points in [CT21a] and when proving our result, for simplicity, we will assume for now that this upper bound can be achieved and omit the discussion on uniformity.

Bootstrapping the win-win argument. We now review the idea discussed in Section 6.1.2, using notations that will be more convenient for the remainder of this technical overview. Fix an arbitrary $n \in \mathbb{N}_{\geq 1}$, and consider the corresponding property $Q_n \subseteq \{0,1\}^n$ decided by $A_Q(x)$ on inputs of length n. Our initial H_0 is trivial and set to $\{0,1\}^n$. (Intuitively, this corresponds to the first case of the [OS17b] argument sketched above where $L_{\mathsf{TV}} \in \mathsf{BPP}$.) Consider now a "brute-force" algorithm $\mathsf{BF}(1^n)$ that computes the first $x \in \mathsf{H}_0$ such that $A_Q(x) = 1$. We let $f(1^n) := \mathsf{BF}(1^n)$ in the Chen–Tell HSG. Note that $f(1^n)$ can be uniformly computed in time $T = 2^{O(n)}$ and depth $d = \mathrm{poly}(n)$, since $A_Q(x)$ runs in polynomial time and all elements of H_0 can be tested in parallel. We set $M(n) := n^\beta$, where $\beta > 1$ is a large enough constant. Let $\mathsf{H}_1 \subseteq \{0,1\}^M$ be the candidate HSG provided by Chen–Tell. Note that H_1 can be computed in time $\widetilde{T} = 2^{O(n)}$ and depth $\widetilde{d} = \mathrm{poly}(n)$.

Next, we consider a win-win argument based on whether Q_M avoids H_1. If this is the case, then Chen–Tell guarantees that we can compute $f(1^n) = \mathsf{BF}(1^n) \in Q_n$ with high probability in time $(d+n) \cdot M^c = \mathrm{poly}(n)$. In other words, we can pseudodeterministically produce a string in Q_n in polynomial time. On the other hand, if $\mathsf{H}_1 \cap Q_M \neq \varnothing$, we now have a set H_1 of strings of

length $M = n^\beta$ that contains a string in Q_M and that can be deterministically computed in time $2^{O(n)}$. That is, we are back to the former case, except that we can compute H_1 (a set containing at least one M-bit prime) in time much faster than $2^{O(M)}$. Crucially, in contrast to the approach of [OS17b], the Chen–Tell HSG does not limit us to the use of the special language L_{TV}, effectively allowing us to reapply the same argument (with a speedup) over a larger input length.

In the next subsection, we discuss the "bootstrapping" and its parameters in more detail, and explain how it yields a polynomial-time pseudodeterministic construction, assuming we have the ideal version of [CT21a] described above.

Infinitely-Often Pseudodeterministic Polynomial-Time Constructions

Let $n_0 \in \mathbb{N}$ be an "initial" input length, and $t = O(\log n_0)$ be a parameter. For each $1 \le i \le t$, we define the i-th input length to be $n_i := n_{i-1}^\beta$, for a large enough constant $\beta > 1$. Our goal is to design a pseudodeterministic algorithm for finding elements in Q that will be correct on *at least one of the input lengths* $n_0, n_1, \ldots, n_t$. On each input length n_i we will have:

1. the property Q_{n_i} that we want to hit;
2. a candidate hitting set generator $\mathsf{H}_i \subseteq \{0,1\}^{n_i}$; and
3. the brute-force algorithm $\mathsf{BF}_i : \{1^{n_i}\} \to \{0,1\}^{n_i}$, which iterates through all elements in H_i and outputs the first element that is in Q_{n_i}.

Note that BF_i is completely defined by H_i. Suppose that H_i can be computed (deterministically) in time T_i and depth d_i, then BF_i can also be computed (deterministically) in time $T_i' := T_i \cdot \mathrm{poly}(n_i)$ and depth $d_i' := d_i \cdot \mathrm{poly}(n_i)$. As discussed above, initially, $\mathsf{H}_0 := \{0,1\}^{n_0}$ is the trivial hitting set generator, $T_0 := 2^{O(n_0)}$, and $d_0 := \mathrm{poly}(n_0)$.

For each $0 \le i < t$, we let $f(1^{n_i}) := \mathsf{BF}_i, M := n_{i+1}$, and invoke the Chen–Tell HSG to obtain the HSG $\mathsf{H}_{i+1} \subseteq \{0,1\}^{n_{i+1}}$. Recall that Chen–Tell guarantees the following: Suppose that $Q_M = Q_{n_{i+1}}$ avoids the HSG H_{i+1}, then one can use $Q_{n_{i+1}}$ to compute $f(1^{n_i})$ with high probability in time $\mathrm{poly}(d_i', n_i, M) \le \mathrm{poly}(d_i, n_i)$, by our choice of parameters. Recall that if H_i indeed hits Q_{n_i}, then $f(1^{n_i})$ implements the brute-force algorithm and outputs the first element in $\mathsf{H}_i \cap Q_{n_i}$ (*i.e.*, a *canonical* element in Q_{n_i}). To reiterate, Chen–Tell gives us the following win-win condition:

- *either* $Q_{n_{i+1}}$ avoids H_{i+1}, in which case we obtain a probabilistic algorithm that outputs a canonical element in Q_{n_i} (thus a pseudodeterministic algorithm) in $\mathrm{poly}(d_i, n_i)$ time;
- *or* H_{i+1} hits $Q_{n_{i+1}}$, in which case we obtain a hitting set H_{i+1} that hits $Q_{n_{i+1}}$, thereby making progress on input length n_{i+1}.

The HSG H_{i+1} can be computed in time $T_{i+1} := (T_i')^c$ and depth $d_{i+1} := d_i' \cdot \log T_i' + n_{i+1}^c$. Crucially, although T_0 is exponential in n_0, it is possible to show by picking a large enough $\beta > 1$ that the sequence $\{n_i\}_{i \in \mathbb{N}}$ grows faster than the sequence $\{T_i\}_{i \in \mathbb{N}}$, and eventually when $i = t = O(\log n_0)$, it will be the case that $T_t \le \mathrm{poly}(n_t)$ and we can apply the brute-force algorithm to find the first element in H_t that is in Q_{n_t} in time polynomial in n_t.

A more precise treatment of the growth of the two sequences $\{n_i\}$ and $\{T_i\}$ is as follows. There is some absolute constant $\alpha \geq 1$ such that $T_0 \leq 2^{\alpha n_0}$ and

$$T_{i+1} \leq T_i^{\alpha} \text{ (for each } 0 \leq i < t).$$

We set $\beta := 2\alpha$ (recall that each $n_{i+1} = n_i^{\beta}$). It follows from induction that for each $0 \leq i \leq t$,

$$T_{i+1} \leq T_0^{\alpha^i} = 2^{\alpha^{i+1} n_0} \quad \text{and} \quad n_{i+1} = n_i^{\beta} = n_0^{\beta^{i+1}} = n_0^{(2\alpha)^{i+1}}.$$

Since

$$\frac{\log T_t}{\log n_t} \leq \frac{\alpha^t n_0}{(2\alpha)^t \log n_0} = \frac{n_0}{2^t \log n_0},$$

it follows that when $t \approx \log(n_0/\log n_0)$, T_t will be comparable to n_t (rather than 2^{n_t}). Similarly, one can show that $d_i \leq \text{poly}(n_i)$ for every $i \leq t$.

Informal description of the algorithm and correctness. To wrap up, we arrive at the following pseudodeterministic algorithm that is correct on at least one of the input lengths $n_0, n_1, \ldots, n_t$. On input length n_i, if $i = t$, then we use $\text{poly}(T_t) \leq \text{poly}(n_t)$ time to find the first string in H_i that is also in Q_{n_i} (*i.e.*, simulate BF_i); otherwise, use $Q_{n_{i+1}}$ as a distinguisher for the Chen–Tell hitting set H_i and print the output of BF_i in $\text{poly}(n_i, d_i) \leq \text{poly}(n_i)$ time. To see that our algorithm succeeds on at least one n_i, consider the following two cases:

1. Suppose that H_t indeed hits Q_{n_t}. Then, clearly, our algorithm succeeds on input length n_t.

2. On the other hand, suppose that H_t does not hit Q_{n_t}. Since our trivial HSG H_0 hits Q_{n_0}, there exists an index $0 \leq i < t$ such that H_i hits Q_{n_i} but $Q_{n_{i+1}}$ avoids H_{i+1}.

   Since $Q_{n_{i+1}}$ avoids H_{i+1}, Chen–Tell guarantees that we can speed up the computation of BF_i using $Q_{n_{i+1}}$ as an oracle. Since H_i hits Q_{n_i}, the output of BF_i is indeed a canonical element in Q_{n_i}. It follows that our algorithm succeeds on input length n_i.

This completes the sketch of the algorithm and its correctness. We note that while this exposition explains how the second bullet of Theorem 6.1.1 is achieved, it does not address the behavior of the algorithm on other input lengths (*i.e.*, the first bullet in the same statement). For simplicity, we omit this here and refer to the formal presentation in Section 6.3.[7]

While the aforementioned construction conveys the gist of our approach, there are two important issues with our presentation. Firstly, as explained before, the results of [CT21a] do not achieve the *ideal parameters* of the HSG stated above. Secondly, we have only vaguely discussed the *circuit uniformity* of the function $f(1^n)$. The uniformity of f is critical for the reconstruction procedure of [CT21a] to run in time comparable to the circuit depth of f. On the other hand, since our HSGs and functions f (corresponding to the algorithm BF) are recursively defined, the circuit uniformity of the [CT21a] generator itself becomes another critical complexity measure in the proof.

[7] Alternatively, the guarantee from the first bullet of Theorem 6.1.1 can always be achieved via a general argument. We refer to [OS17b, Proposition 2] for the details.

In the next subsection, we discuss the Chen–Tell generator in more detail and explain how to obtain an improved generator construction satisfying our requirements.

Improving the Chen–Tell Targeted Hitting Set Generator

The uniform hardness-to-randomness framework of Chen–Tell builds on two important ingredients:[8]

1. A *layered-polynomial representation* of a shallow uniform circuit.
2. A hitting set generator with a *uniform learning reconstruction* algorithm.

Layered-polynomial representation. We now discuss the first ingredient. Let $f\colon \{0,1\}^n \to \{0,1\}^n$ be a logspace-uniform circuit family of size $T(n)$ and depth $d(n)$.[9] Let $M\colon \mathbb{N} \to \mathbb{N}$ be the parameter for output length. Building on the doubly efficient interactive proof system by [GKR15] (and its subsequent simplification by [Gol17]), for any $z \in \{0,1\}^n$, [CT21a] showed that there is a sequence of polynomials $\{P_i^z\}_{i\in[d']}$ for $d' = d \cdot \mathrm{polylog}(T)$ with the following nice properties:

- (**Arithmetic setting.**) Let $\mathbb{F}$ be a finite field of size M^c for a large universal constant $c > 1$, and let m be of order $\frac{\log T}{\log M}$. All the P_i^z map $\mathbb{F}^m$ to $\mathbb{F}$ and have total degree at most M.
- (**Base case.**) There is an algorithm Base such that, given the input $z \in \{0,1\}^n$ and $\vec{w} \in \mathbb{F}^m$, computes $P_1^z(\vec{w})$ in $\mathrm{poly}(M)$ time.
- (**Downward self-reducibility.**) There is an oracle algorithm DSR that, given input $i \in \{2,\dots,d'\}$ and $\vec{w} \in \mathbb{F}^m$, together with the oracle access to $P_{i-1}^z(\cdot)$, computes $P_i^z(\vec{w})$ in $\mathrm{poly}(M)$ time.
- (**Faithful representation.**) There is an oracle algorithm OUT that, given input $i \in [n]$ and oracle access to $P_{d'}^z$, outputs $f(z)_i$ in $\mathrm{poly}(M)$ time.

Intuitively, these polynomials form an *encoded* version of the computation of f in the sense that they admit both *downward self-reducibility* and *random self-reducibility*: every P_i^z has low degree and hence admits error correction properties; downward self-reducibility follows from the definition.

We note that the proof of this result depends in a crucial way on the logspace-uniformity of the circuit family computing f. (This allows one to arithmetise a formula of bounded size that computes the direct connection language of the circuit, while also controlling the circuit uniformity of the resulting polynomials.)

[8] Below we will focus on the high-level picture of the Chen–Tell framework without diving into too many details. Our presentation is also somewhat different from the original presentation in [CT21a].

[9] Intuitively, a circuit family is logspace-uniform if each circuit in the family can be printed by a fixed machine that runs in space that is of logarithmic order in the size of the circuits. See Section 6.2.3 for the precise definition of logspace-uniform circuits.

Hitting set generators with a uniform learning reconstruction algorithm. The second ingredient of [CT21a] is the Nisan–Wigderson generator combined with Reed–Muller codes [NW94, STV01]. The most important property of this generator is that it supports a uniform learning reconstruction algorithm. In more detail, for a polynomial $P\colon \mathbb{F}^m \to \mathbb{F}$, the generator NW^P takes $s = O\left(\frac{\log^2 T}{\log M}\right)$ bits as seed, such that there is a uniform oracle algorithm R (for "reconstruction") where the following holds. Given oracle access to both P and an oracle $D\colon \{0,1\}^M \to \{0,1\}$ that distinguishes $\mathsf{NW}^P(U_s)$ from the uniform distribution, $R^{P,D}$ runs in $\mathrm{poly}(M)$ time and with high probability outputs a polynomial-size D-oracle circuit that computes P.

Now, the hitting set $H_f(z)$ is defined as

$$H_f(z) := \bigcup_{i \in [d']} \mathsf{NW}^{P_i^z} .$$

The uniform reconstruction algorithm. One key observation here is that if a distinguisher $D\colon \{0,1\}^M \to \{0,1\}$ avoids $H_f(z)$, meaning that D accepts a large fraction of inputs from $\{0,1\}^M$ but rejects all strings in $H_f(z)$, then clearly D also distinguishes all $\mathsf{NW}^{P_i^z}(U_s)$ from the uniform distribution. Following [IW01], [CT21a] then shows that there is a uniform oracle algorithm R_f that takes input $z \in \{0,1\}^n$ and any "avoider" D of $H_f(z)$ as oracle, and outputs $f(z)$ with high probability. In more detail, R_f works as follows:

1. It is given input $z \in \{0,1\}^n$ and oracle access to an avoider $D\colon \{0,1\}^M \to \{0,1\}$ of $H_f(z)$.
2. For every $i \in \{2, \dots, d'\}$:
   (a) The goal of the i-th step is to construct a $\mathrm{poly}(M)$-size D-oracle circuit C_i that computes P_i^z.
   (b) It runs the learning reconstruction algorithm $R^{P_i^z, D}$ to obtain a $\mathrm{poly}(M)$-size D-oracle circuit. To answer queries to P_i^z, we first run the algorithm DSR to convert them into queries to P_{i-1}^z. Next, when $i = 2$, we answer these queries by calling Base directly, and when $i > 2$, we answer these queries by evaluating our D-oracle circuit C_{i-1}.
3. For every $i \in [n]$, output $\mathsf{OUT}_{d'}^{C^D}(i)$.

Issue with the original Chen–Tell construction: Super-logarithmic seed length of NW. The main issue with the construction above is that $\mathsf{NW}^{P_i^z}$ has seed length $O\left(\frac{\log^2 T}{\log M}\right)$. In particular, this means that when $\log M \leq o(\log T)$, the hitting set $H_f(z)$ has super-polynomial size, and therefore cannot be computed in $\mathrm{poly}(T)$ time as in the "ideal version" of [CT21a] stated above.[10] Hence, to improve the computation time of $H_f(z)$ to $\mathrm{poly}(T)$, we need an HSG with seed length $O(\log T)$ for all possible values of M, together with a uniform learning reconstruction, when it is instantiated with polynomials. Jumping ahead, we will replace NW with the Shaltiel–Umans Hitting Set Generator [SU05], obtaining an optimised version of the Chen–Tell generator with better parameters. However, the original generator from [SU05] does

[10] Indeed, if we rely on the original Chen–Tell construction to implement the bootstrapping method described above, we would only obtain a quasi-polynomial-time pseudodeterministic construction, instead of a polynomial-time one.

not provide a uniform learning reconstruction procedure. By using the classical construction of a *cryptographic pseudorandom generator from a one-way permutation* and another idea, we manage to modify their construction to allow a uniform learning reconstruction. See the next subsection for more details.

Controlling the circuit uniformity of the optimised Chen–Tell generator. As stressed above, in order to construct a layered-polynomial representation for f with the aforementioned parameters, it is crucial that f admits a logspace-uniform circuit family. Since we will rely on multiple applications of the generator, and each new function BF on which the result is invoked contains as a subroutine the code of the previous generator, we must *upper bound the circuit uniformity* of our optimised Chen–Tell generator. This turns out to require a delicate manipulation of all circuits involved in the proof and of the Turing machines that produce them, including the components of the Shaltiel–Umans generator. For this reason, whenever we talk about a Boolean circuit in the actual proof, we also bound the description length and space complexity of its corresponding machine. Additionally, as we manipulate a super-constant number of circuits (and their corresponding machines) in our construction, we will also consider the complexity of producing the code of a machine M_2 encoding a circuit C_2 from the code of a machine M_1 encoding a circuit C_1 (see, e.g., the "Moreover" part in the statement of Theorem 6.3.1). The details are quite tedious, but they are necessary for verifying the correctness and running time of our algorithm. In order to provide some intuition for it, we notice that as we move from the HSG H_i to H_{i+1}, we also increase the corresponding input length parameter from n_i to $n_{i+1} = n_i^{\beta}$. While there is an increase in the uniformity complexity, it remains bounded relative to the new input length. (Think of a truncated geometric series whose value is dominated by the complexity over the current input length.) We omit the details in this proof overview.

Non-black-box behavior. We note that the recursive application of the Chen–Tell generator is responsible for the *fully non-black-box* behavior of our pseudodeterministic construction. Indeed, since we invoke the Chen–Tell generator on each function BF (which contains the code of the algorithm A_Q deciding property Q as a subroutine), the collection of strings in the hitting set generator depends on the layered-polynomial representation that is obtained from the *code* of BF. As a consequence, our construction has the unusual feature that the canonical outputs of the algorithm B in Theorem 6.1.1 are affected by the code of A_Q. In other words, by using a different primality test algorithm (or by making changes to the code implementing the AKS routine), one might get a different n-bit prime!

The parameters of our hitting set generator appear in Section 6.3. The proof of the result is given in Section 6.5.

Modified Shaltiel–Umans Generator with Uniform Learning Reconstruction

As explained above, in order to complete the proof of Theorem 6.1.1 we need to design a variant of the Shaltiel–Umans generator [SU05] with a *uniform learning reconstruction* procedure.

The Shaltiel–Umans generator takes as input a low-degree polynomial $P : \mathbb{F}_p^m \to \mathbb{F}_p$ (in our case, p will be a power of 2) and produces a set of binary strings (which is supposed to be a hitting set). The construction of this generator also relies on "generator matrices". A matrix $A \in \mathbb{F}_p^{m \times m}$ is a *generator matrix* if it satisfies $\{A^i \cdot \vec{1}\}_{1 \le i < p^m} = \mathbb{F}_p^m \setminus \{\vec{0}\}$. Roughly put, the matrix A can be thought of as performing multiplication with a generator of the multiplicative group of $\mathbb{F}_{p^m}$.

Recall that a generator has a uniform learning reconstruction algorithm if the following holds. Given an algorithm D that avoids the output of the generator constructed using P, as well as P itself, we can *uniformly* and *efficiently* generate (with high probability) a D-oracle circuit that computes the polynomial P. (In other words, we can query P while producing the circuit, but the circuit itself does not have access to P.)

However, the reconstruction procedure provided by the original Shaltiel–Umans generator only guarantees the following: If the generator is constructed using P and some generator matrix A, then using an algorithm D that avoids the output of the generator, and *given the matrix* A and oracle access to P, one can obtain a (D-oracle) circuit $C : [p^m - 1] \to \mathbb{F}_p^m$ such that $C(i) = P(A^i \cdot \vec{1})$.[11] (For the precise statement, see Theorem 6.4.9.) That is, this reconstruction is not a uniform learning algorithm in the following sense:

1. It needs to know the matrix A (which can be viewed as non-uniform advice).
2. Given oracle access to P, it only learns a circuit that computes the mapping $i \mapsto P(A^i \cdot \vec{1})$, instead of a circuit that computes $P(\vec{x})$ on a given $\vec{x} \in \mathbb{F}_p^m$.

We now describe how to modify the Shaltiel–Umans generator to make its reconstruction a uniform learning algorithm.

For the first issue, our idea is that, instead of using a generator matrix that is obtained by brute-force search as in the original construction (we note that the reconstruction cannot afford to perform the brute-force search due to its time constraints), we will use a generator matrix that is from a small set of matrices that can be constructed *efficiently*. More specifically, using results about finding primitive roots of finite fields (*e.g.*, [Sho92]), we show that one can efficiently and deterministically construct a set S of matrices that contains at least one generator matrix. The advantage is that the reconstruction algorithm can still afford to compute this set S. Note that although we don't know which matrix in S is a valid generator matrix (as verifying whether a matrix is a generator matrix requires too much time), we can try all the matrices from S, and one of them will be the correct one. This allows us to obtain a list of candidate circuits, one of which computes P (provided that we can also handle the second issue, which will be discussed next). Then, by selecting from the list a circuit that is sufficiently close to P (note that given oracle access to P, we can easily test whether a circuit is close to P by sampling) and by using the *self-correction* property of low-degree polynomials, we can obtain a circuit that computes P exactly.

With the above idea, we may now assume that in the reconstruction we know the generator matrix A used by the Shaltiel–Umans generator. Next, we describe how to handle the second issue. Recall that the reconstruction algorithm of the Shaltiel–Umans generator gives a circuit

[11] In fact, the circuit only computes $P(A^i \cdot \vec{v})$ for some $\vec{v}$ output by the reconstruction algorithm. We assume $\vec{v} = \vec{1}$ here for simplicity.

C such that $C(i) = P(A^i \cdot \vec{1})$, for $i \in [p^m - 1]$, and we want instead a circuit that given $\vec{x} \in \mathbb{F}_p^m$ computes $P(\vec{x})$. Now suppose given $\vec{x} \in \mathbb{F}_p^m \setminus \{\vec{0}\}$, we can also *efficiently* compute the value $i \in [p^m - 1]$ such that $A^i \cdot \vec{1} = \vec{x}$. Then we would be able to combine this with C to get a circuit E that computes P, *i.e.*, if $\vec{x} = \vec{0}$ then E outputs $P(\vec{0})$ (where the value $P(\vec{0})$ can be hardcoded); otherwise, E computes i for $\vec{x}$ as described above and then outputs $C(i)$. However, the task of finding such i given A and $\vec{x}$ is essentially the *discrete logarithm problem*, for which no efficient algorithm is known!

A classical result in cryptography is that one can construct a pseudorandom generator based on the hardness of the discrete logarithm problem (see, *e.g.*, [BM84, Yao82]). More generally, given a permutation f whose inverse admits *random self-reducibility*[12], one can construct a generator G based on f so that if there is a distinguisher D that breaks G, then it can be used to invert f via a uniform reduction. Our idea is to consider the bijection $f : [p^m - 1] \to \mathbb{F}_p^m \setminus \{\vec{0}\}$ such that for each $i \in [p^m - 1]$, $f(i) = A^i \cdot \vec{1}$ (where the random self-reducibility of f^{-1} follows easily from that of the discrete logarithm problem), and try to construct a pseudorandom generator G based on f. We then combine the output of G with that of the Shaltiel–Umans generator constructed with the polynomial P and the generator matrix A. Now if there is an algorithm D that avoids this combined generator, which means D *simultaneously* avoids both the Shaltiel–Umans generator and the generator G, then D can be used to obtain

- a circuit C such that $C(i) = P(A^i \cdot \vec{1})$ for every $i \in [p^m - 1]$, and
- a circuit C' that inverts f, *i.e.*, $C'(\vec{x})$ outputs i such that $A^i \cdot \vec{1} = \vec{x}$ for every $\vec{x} \in \mathbb{F}_p^m \setminus \{\vec{0}\}$.

Then it is easy to combine C and C' to obtain a circuit that computes P.

A careful implementation of these ideas allows us to obtain a variant of the Shaltiel–Umans generator with uniform learning reconstruction, as needed in our optimised Chen–Tell generator. We refer to Theorem 6.4.1 in Section 6.4 for more details.

This completes the sketch of the proof of Theorem 6.1.1.

Further remarks about the proof. We note that in our proof the gap between two good input lengths on which the algorithm outputs a canonical prime can be exponentially large. It would be interesting to develop techniques to reduce this gap.

Additionally, the proof assumes the existence of a deterministic polynomial-time algorithm that decides the dense property. In contrast, the subexponential time algorithm from [OS17b] also works with a dense property that is decidable by a randomised polynomial-time algorithm. This is caused by the non-black-box nature of our approach via the Chen-Tell generator, which employs the code of the algorithm A deciding the property as part of the description of the generator. Consequently, as alluded to above, changing the code of A could result in a different canonical output on a given input length. If A is randomised, fixing the randomness of A is similar to the consideration of a different algorithm that decides the property, and it is not immediately clear how to maintain the pseudodeterministic behaviour in this case.

[12] Roughly speaking, a function has random self-reducibility if computing the function on a given instance can be efficiently reduced to computing the function for uniformly random instances.

6.2 Preliminaries

For a positive integer k, we use $[k]$ to denote the set $\{1, 2, \ldots, \}$. We use $\mathbb{N}$ to denote all non-negative integers and $\mathbb{N}_{\geq 1}$ to denote all positive integers.

For $x, y \in \{0,1\}^*$, we use $x \circ y$ to denote their concatenation.[13] For a function $f\colon \{0,1\}^\ell \to \{0,1\}$ we use $\mathtt{tt}(f)$ to denote the 2^ℓ-length truth-table of f (*i.e.*, $\mathtt{tt}(f) = f(w_1) \circ f(w_2) \circ \ldots \circ f(w_{2^\ell})$, where $w_1, \ldots, w_{2^\ell}$ is the enumeration of all strings from $\{0,1\}^\ell$ in the lexicographical order).

Unless explicitly stated otherwise, we assume that all circuits are comprised of Boolean NAND gates of fan-in two. In several places, we will need the following notion, which strengthens the standard notion of a time-computable function by requiring the function to be computable in logarithmic space. The depth of a circuit is defined to be the maximum length (measured by the number of edges) of any input-to-output path.

Definition 6.2.1 (Logspace-Computable Functions)**.** We say that a function $T\colon \mathbb{N} \to \mathbb{N}$ is logspace-computable if there exists an algorithm that gets input 1^n, runs in space $O(\log(T(n)))$, and outputs $T(n)$.

For convenience, we consider circuit families indexed by a tuple of parameters. Specifically, a circuit family with k input parameters $\vec{\ell} = (\ell_1, \ell_2, \ldots, \ell_k) \in \mathbb{N}^k$ is defined as $\{C_{\vec{\ell}}\}_{\vec{\ell} \in \mathbb{N}^k}$, where each $C_{\vec{\ell}}$ is a circuit.

6.2.1 Finite Fields

Throughout this chapter, we will only consider finite fields of the form $\mathrm{GF}(2^{2 \cdot 3^\lambda})$ for some $\lambda \in \mathbb{N}$ since they enjoy simple representations that will be useful for us. We say $p = 2^r$ is a *nice power* of 2, if $r = 2 \cdot 3^\lambda$ for some $\lambda \in \mathbb{N}$.

Let $\ell \in \mathbb{N}$ and $n = 2 \cdot 3^\ell$. In the following, we use $\mathbb{F}$ to denote $\mathbb{F}_{2^n}$ for convenience. We will always represent $\mathbb{F}_{2^n}$ as $\mathbb{F}_2[\mathbf{x}]/(\mathbf{x}^n + \mathbf{x}^{n/2} + 1)$.[14] That is, we identify an element of $\mathbb{F}_{2^n}$ with an $\mathbb{F}_2[\mathbf{x}]$ polynomial with degree less than n. To avoid confusion, given a polynomial $P(\mathbf{x}) \in \mathbb{F}_2[\mathbf{x}]$ with degree less than n, we will use $(P(\mathbf{x}))_{\mathbb{F}}$ to denote the unique element in $\mathbb{F}$ identified with $P(\mathbf{x})$.

Let $\kappa^{(n)}$ be the natural bijection between $\{0,1\}^n$ and $\mathbb{F} = \mathrm{GF}(2^n)$: for every $a \in \{0,1\}^n$, $\kappa^{(n)}(a) = \left(\sum_{i \in [n]} a_i \cdot \mathbf{x}^{i-1}\right)_{\mathbb{F}}$. We always use $\kappa^{(n)}$ to encode elements from $\mathbb{F}$ by Boolean strings. That is, whenever we say that an algorithm takes an input from $\mathbb{F}$, we mean it takes a string $x \in \{0,1\}^n$ and interprets it as an element of $\mathbb{F}$ via $\kappa^{(n)}$. Similarly, whenever we say that an algorithm outputs an element from $\mathbb{F}$, we mean it outputs a string $\{0,1\}^n$ encoding that element via $\kappa^{(n)}$. For simplicity, sometimes we use $(a)_{\mathbb{F}}$ to denote $\kappa^{(n)}(a)$. Also, when we say the i-th element in $\mathbb{F}$, we mean the element in $\mathbb{F}$ encoded by the i-th lexicographically smallest Boolean string in $\{0,1\}^n$.

[13] We sometimes also use $C_1 \circ C_2$ to denote the composition of two circuits, but the meaning of the symbol $\circ$ will always be clear from the context.

[14] $\mathbf{x}^{2 \cdot 3^\ell} + \mathbf{x}^{3^\ell} + 1 \in \mathbb{F}_2[\mathbf{x}]$ is irreducible, see [VL99, Theorem 1.1.28].

6.2.2 Bounded-Space Turing Machines

Our argument is robust to specific details about the computational model, but in order to estimate the relevant bounds, we must fix a model. We use the standard model of space-bounded computation (see [Gol08, Section 5] or [AB09, Section 4]). A deterministic space-bounded Turing machine has three tapes: an input tape (that is read-only), a work tape (that is read/write), and an output tape (that is write-only and uni-directional). We assume that the machine's alphabet is $\Sigma := \{0,1\}$. The space complexity of the machine is the number of cells used on the work tape. For concreteness, we assume that the work tape contains initially only $\square$ ("blank") symbols, and that the machine writes symbols from Σ in the tape.

Throughout this chapter, we will describe a space-bounded Turing machine by fixing a universal Turing machine U that has an additional read-only *program tape* such that $\mathsf{TM}(x)$ is defined to be the output of U with the program tape initialised as TM.[15] Abusing the notation, we often use TM to denote both the Turing machine and a binary string description of the Turing machine. Without loss of generality, we also assume our description is *paddable* meaning that for every $\mathsf{TM} \in \{0,1\}^*$ and $k \in \mathbb{N}$, TM and $\mathsf{TM} \circ 0^k$ represent the same machine. To avoid certain technicalities, we will always assume that the space bound of a Turing machine TM is greater than its description size.

Configurations of space-bounded machines. On a fixed input $x \in \{0,1\}^n$, a space-s Turing machine TM has $2^{s'}$ possible configurations, where $s' = s'(s,n) = s + O(\log s) + \log n$. Each configuration can be described by s' bits. Here, s measures the space used by the universal Turing machine U that simulates TM on input x. In more detail, it can be described by the content of U's work tape, U's current state, and the location of U's heads, including the head on the input/program tape. (Note that a configuration does not include the content of the output tape, which does not affect the next step of the machine.)

We will need the following fact for determining the relationship between configurations of a Turing machine. Recall that a sequence $\{D_n\}_{n\geq 1}$ of size-$T(n)$ computational devices is *logspace-uniform* if there is a machine $M(1^n)$ that runs in space $O(\log T(n))$ and outputs D_n (or equivalently, decides the direct connection language of D_n).

Fact 6.2.2. *Given a description of Turing machine $\mathsf{TM} \in \{0,1\}^*$, a space bound $s \in \mathbb{N}$, an input $x \in \{0,1\}^n$, and two configurations $\gamma, \gamma' \in \{0,1\}^{s'}$, there is an algorithm $\mathbb{A}_{\mathsf{nxt}}$ that determines whether γ' is the next configuration obtained by running TM for one step on input x. Moreover, $\mathbb{A}_{\mathsf{nxt}}$ can be computed by a logspace-uniform $O(m^3)$-size $O(\log m)$-depth formula and by an $O(m)$-space algorithm, where m is the total number of input bits. (Here, we assume that if γ is the accepting state or the rejecting state, then the next configuration of γ is always γ itself.)*

6.2.3 Circuits Generated by Bounded-Space Turing Machines

In this chapter, we often use the following two representations of a circuit (recall that throughout this chapter, all circuits consist entirely of fan-in two `NAND` gates).

[15] The advantage of fixing a universal Turing machine is that now our Turing machine always has a constant number of states, which is helpful when bounding the number of configurations of a Turing machine of super-constant size.

- (**Adjacency relation tensor.**) A circuit C of size T is given as a tensor $T_C \in \{0,1\}^{T\times T\times T}$ such that for every tuple $(u,v,w) \in [T]^3$, $T_C(u,v,w) = 1$ if and only if the gates in C indexed by v and by w feed into the gate in C indexed by u.

- (**Layered adjacency relation tensor.**) A circuit C of width T and depth d is given as a list of d tensors $T_C^{(i)} \in \{0,1\}^{T\times T\times T}$, where $i \in [d]$, such that for every layer $i \in [d]$ and tuple $(u,v,w) \in [T]^3$, $T_C^{(i)}(u,v,w) = 1$ if and only if the gates in the $(i-1)$-th layer of C indexed by v and by w feed into the gate in the i-th layer of C indexed by u.

  Here, the input gates are on the 0-th layer, and the output gates are on the d-th layer. Without loss of generality, we can assume all layers have exactly T gates.

In both cases above, when evaluating C in a context, we will also specify two integers n_{in} and n_{out} to denote the number of input/output gates; see the definition of $\mathsf{Circuit}[T,s,n_{\mathsf{in}},n_{\mathsf{out}}](\mathsf{TM})$ given below for details.

While we will mostly use the (unlayered) adjacency relation tensor representation, the layered variant will be very convenient in Section 6.5.1.

We define next a more general notion of a space-uniform circuit family with input parameters. This will be useful in some situations where we need to compute explicit space bounds for uniformity and index circuits by a tuple of parameters.

Definition 6.2.3 (α-Space-Uniform Circuits)**.** Let $k \in \mathbb{N}$ and $\alpha, T\colon \mathbb{N}^k \to \mathbb{N}$. We say that a circuit family with k input parameters $\{C_{\vec{\ell}}\}_{\vec{\ell}\in\mathbb{N}^k}$ of size $T = T(\vec{\ell})$ is α-$\mathsf{space\text{-}uniform}$ if there exists an algorithm A such that:

1. **Decides the adjacency relation.** The algorithm gets $\vec{\ell} \in \mathbb{N}^k$ and $(u,v,w) \in \{0,1\}^{3\log(T)}$ as input and accepts if and only if the gates in $C_{\vec{\ell}}$ indexed by v and by w feed into the gate in $C_{\vec{\ell}}$ indexed by u. (That is, the algorithm computes the adjacency relation tensor of $C_{\vec{\ell}}$.)

2. **Runs in $\alpha(\vec{\ell})$ space.** For input parameters $\vec{\ell} \in \mathbb{N}^k$, the algorithm runs in space $\alpha(\vec{\ell})$.

We say $\{C_{\vec{\ell}}\}_{\vec{\ell}\in\mathbb{N}^k}$ is $\mathsf{logspace\text{-}uniform}$ if it is $\mu \log T$-space-uniform for some constant μ.

Circuit determined by a Turing machine through the adjacency relation tensor. We will also consider the circuit determined by a Turing machine in the non-asymptotic setting. More specifically, given a Turing machine $\mathsf{TM} \in \{0,1\}^*$, parameters $T, s, n_{\mathsf{in}}, n_{\mathsf{out}} \in \mathbb{N}$, we use $\mathsf{Circuit}[T,s,n_{\mathsf{in}},n_{\mathsf{out}}](\mathsf{TM})$ to denote the circuit whose adjacency relation is determined by running TM with space bound s over all triples $(u,v,w) \in \{0,1\}^{3\log T}$ with $u > v > w$. The first n_{in} out of T gates are the input gates, and the last n_{out} out of T gates are the output gates. If TM fails to halt on some triples using s bits of space, or the resulting circuit is invalid (*i.e.*, inputs are not source, or outputs are not sink), we let $\mathsf{Circuit}[T,s,n_{\mathsf{in}},n_{\mathsf{out}}](\mathsf{TM}) = \bot$.

Given two circuits $C_1\colon \{0,1\}^{n_1} \to \{0,1\}^{n_2}$ and $C_2\colon \{0,1\}^{n_2} \to \{0,1\}^{n_3}$, one can compose them into a single circuit $C_2 \circ C_1\colon \{0,1\}^{n_1} \to \{0,1\}^{n_3}$ in a natural way (*i.e.*, by identifying the outputs of C_1 with the inputs of C_2). Suppose C_1 is a circuit of size T_1 and depth d_1, and C_2 is a circuit of size T_2 and depth d_2, then $C_2 \circ C_1$ has size $T_1 + T_2$ and depth $d_1 + d_2$. Also, if

C_1, C_2 are given by two Turing machines TM_1 and TM_2, we can easily generate another Turing machine TM_3 that specifies $C_2 \circ C_1$. Formally, we will pick a universal machine such that we have the following simple fact on the description length of TM_3, whose proof we omit.

Fact 6.2.4 (Turing Machine Description of Circuit Composition)**.** *There is a universal constant $c_{\mathsf{comp}} \in \mathbb{N}$ such that the following holds. Given the descriptions of Turing machines TM_1 and TM_2, parameters*

$$\vec{\ell}_1 = (T_1, s_1, n_1, n_2), \qquad \vec{\ell}_2 = (T_2, s_2, n_2, n_3) \in \mathbb{N}^4,$$

and letting

$$C_1 = \mathsf{Circuit}[\vec{\ell}_1](\mathsf{TM}_1),\ C_2 = \mathsf{Circuit}[\vec{\ell}_2](\mathsf{TM}_2), \quad \text{and} \quad \vec{\ell}_3 = (T_1 + T_2, 2 \cdot (s_1 + s_2) + c_{\mathsf{comp}}, n_1, n_3),$$

there is a polynomial-time algorithm $\mathbb{A}_{\mathsf{comp}}$ that given $\mathsf{TM}_1, \mathsf{TM}_2, \vec{\ell}_1, \vec{\ell}_2$ as input, outputs the description of a Turing machine TM_3 such that[16]

$$(C_2 \circ C_1) = \mathsf{Circuit}[\vec{\ell}_3](\mathsf{TM}_3) \quad \text{and} \quad |\mathsf{TM}_3| \leq 2 \cdot (|\mathsf{TM}_1| + |\mathsf{TM}_2| + \log n_2) + c_{\mathsf{comp}}.$$

6.2.4 Pseudorandom Generators and Hitting Set Generators

Definition 6.2.5 (Avoiding and Distinguishing)**.** Let $m, t \in \mathbb{N}$, $D\colon \{0,1\}^m \to \{0,1\}$, and $Z = (z_i)_{i \in [t]}$ be a list of strings from $\{0,1\}^m$. Let $\varepsilon \in (0,1)$. We say that D *ε-distinguishes* Z, if

$$\left| \Pr_{r \leftarrow \{0,1\}^m}[D(r) = 1] - \Pr_{i \leftarrow [t]}[D(z_i) = 1] \right| \geq \varepsilon.$$

We say that D *ε-avoids* Z, if $\Pr_{r \leftarrow \{0,1\}^m}[D(r) = 1] \geq \varepsilon$ and $D(z_i) = 0$ for every $i \in [t]$.

6.3 Pseudodeterministic Constructions for Dense Properties

In this section, we prove our main result, restated below for convenience.

Theorem 6.1.1 (Infinitely-Often Polynomial-Time Pseudodeterministic Constructions)**.** *Let $Q \subseteq \{0,1\}^*$ be a language with the following properties:*

(Density.) *there is a constant $\rho \geq 1$ such that for every $n \in \mathbb{N}_{\geq 1}$, $Q_n := Q \cap \{0,1\}^n$ satisfies $|Q_n| \geq n^{-\rho} \cdot 2^n$; and*

(Easiness.) *there is a deterministic polynomial-time algorithm A_Q that decides whether an input $x \in \{0,1\}^*$ belongs to Q.*

Then there exist a probabilistic polynomial-time algorithm B and a sequence $\{x_n\}_{n \in \mathbb{N}_{\geq 1}}$ of n-bit strings in Q such that the following conditions hold:

1. *On every input length $n \in \mathbb{N}_{\geq 1}$, $\Pr_B[B(1^n) \notin \{x_n, \bot\}] \leq 2^{-n}$.*
2. *On infinitely many input lengths $n \in \mathbb{N}_{\geq 1}$, $\Pr_B[B(1^n) = x_n] \geq 1 - 2^{-n}$.*

[16] We note that if either $C_1 = \bot$ or $C_2 = \bot$, then there is no guarantee on $\mathbb{A}_{\mathsf{comp}}$'s behavior.

We will need the following theorem, which is obtained by combining [SU05] and [CT21a]. The proof is presented in Section 6.5.

Theorem 6.3.1 (Improved Chen–Tell Hitting Set Generator)**.** *There exists a universal $c \in \mathbb{N}_{\geq 1}$, a deterministic algorithm H^{ct}, and a probabilistic oracle algorithm R^{ct} such that the following holds. Let $\kappa, \rho \in \mathbb{N}$. Let $T, d, M, n \in \mathbb{N}$ all be sufficiently large such that $n \leq T$, $d \leq T$, and $c \cdot \log T \leq M \leq T^{1/(c\rho)}$. Denote $\vec{\ell} := (n, T, d, M, \kappa, \rho)$ as the input parameters.*

For a Turing machine TM with description size $|\mathsf{TM}| = \kappa \cdot \log T$, we let

$$C_{\mathsf{TM}} := \mathsf{Circuit}[T, \kappa \cdot \log T, n, n](\mathsf{TM}).$$

Assume the circuit $C_{\mathsf{TM}} \neq \bot$ and C_{TM} has depth at most d.

(Generator.) *The generator $\mathsf{H}^{\mathsf{ct}}_{\vec{\ell}}$ (we write $\mathsf{H}^{\mathsf{ct}}_{\vec{\ell}}$ to denote that H^{ct} takes $\vec{\ell}$ as input parameters) takes the description of a Turing machine $\mathsf{TM} \in \{0,1\}^{\kappa \log T}$ as input, and outputs a list of M-bit strings. We assume that the list has exactly $T^{(c \cdot \kappa)/2}$ entries.*

Let $\widetilde{T} := T^{c \cdot \kappa}$ and $\widetilde{d} := c \cdot (d \log T + \kappa^2 \log^2 T) + M^c$. There is a Turing machine TM_{H} with description length $c \log \widetilde{T}$ such that for

$$C_{\mathsf{H}} := \mathsf{Circuit}\left[\widetilde{T}, c \cdot \kappa \log T, n, \left(\widetilde{T}\right)^{1/2} \cdot M\right](\mathsf{TM}_{\mathsf{H}}),$$

it holds that (1) $C_{\mathsf{H}}(1^n) = \mathsf{H}^{\mathsf{ct}}_{\vec{\ell}}(\mathsf{TM})$ and (2) C_{H} has depth $\widetilde{d}$. Moreover, there is a polynomial-time[17] algorithm $\mathbb{A}^{\mathsf{ct}}$ that on inputs $\vec{\ell}$ and $\mathsf{TM} \in \{0,1\}^{\kappa \log T}$, outputs the description of TM_{H}.

(Reconstruction.) *The reconstruction algorithm R^{ct} takes the description of a Turing machine $\mathsf{TM} \in \{0,1\}^{\kappa \log T}$ as input, receives an oracle $D \colon \{0,1\}^M \to \{0,1\}$, and satisfies the following:*

- **(Soundness.)** *For every oracle $D \colon \{0,1\}^M \to \{0,1\}$, $(\mathsf{R}^{\mathsf{ct}})^D_{\vec{\ell}}(\mathsf{TM})$ runs in time $(d+n) \cdot M^{c\rho}$ and with probability at least $1 - 2^{-M}$, its output is either $C_{\mathsf{TM}}(1^n)$ or $\bot$.*
- **(Completeness.)** *If D $(1/M^{\rho})$-avoids $\mathsf{H}^{\mathsf{ct}}_{\vec{\ell}}(\mathsf{TM})$, then $(\mathsf{R}^{\mathsf{ct}})^D_{\vec{\ell}}(\mathsf{TM})$ outputs $C_{\mathsf{TM}}(1^n)$ with probability at least $1 - 2^{-M}$.*

We are now ready to prove Theorem 6.1.1.

Proof of Theorem 6.1.1. We start with some notations.

Notation. Let $n_0 \in \mathbb{N}$ be sufficiently large. We define $n_0^{(0)} = n_0$, and for every $\ell \in \mathbb{N}_{\geq 1}$,

$$n_0^{(\ell)} = 2^{2^{n_0^{(\ell-1)}}}.$$

Now, fix $\ell \in \mathbb{N}$. For simplicity of notation, in the following we will use n_i, H_i, t to denote $n_i^{(\ell)}, \mathsf{H}_i^{(\ell)}, t^{(\ell)}$, which will be defined later.

[17] In this chapter, whenever we say an algorithm $\mathbb{A}$ that generates Turing machines or other succinct descriptions *runs in polynomial time*, we mean the running time is polynomial in the total number of input bits. In this case, the time bound is polynomial in the description length of $\vec{\ell}$ and TM, *i.e.*, $\mathrm{poly}(\kappa \log T)$.

Construction of hitting sets. For some parameter t that we set later, we will define a sequence of input lengths $n_1, \dots, n_t$, with the hope that we can construct a string in Q pseudo-deterministically on at least one of the input lengths.

Let $\beta \in \mathbb{N}_{\geq 1}$ be a sufficiently large constant to be chosen later. For every $i \in [t]$, we set $n_i = (n_{i-1})^\beta$. For each $i \in \{0, \dots, t\}$, we will construct a hitting set $\mathsf{H}_i \subseteq \{0,1\}^{n_i}$, which is computable by a logspace-uniform T_i-size d_i-depth circuit. As the base case, we set H_0 as the whole set $\{0,1\}^{n_0}$. We note that there is a logspace-uniform T_0-size d_0-depth circuit that outputs all elements in H_0, where $T_0 = 2^{2n_0}$ and $d_0 = 2n_0$.

Let $\kappa \in \mathbb{N}$ be a large enough constant to be specified later. Let c be the universal constant from Theorem 6.3.1.

Informal description. We will first give a somewhat informal description of the construction of the H_i, in particular, we will omit details about the uniformity of the circuits (whose analysis is rather tedious). We hope this can help the reader gain some intuition first. Later, we will carefully analyse the uniformity of the circuits for H_i.

For each $i \in [t]$, we construct H_i as follows:

1. We define BF_{i-1} as the circuit implementing the following algorithm: Enumerate every element in $\mathsf{H}_{i-1} \subseteq \{0,1\}^{n_{i-1}}$, and output the first element that is in $Q_{n_{i-1}}$; if no such element exists, then $\mathsf{BF}_{i-1}(n)$ outputs $\perp$;

   Using the assumed polynomial-time algorithm A_Q for deciding membership in Q, BF_{i-1} can be implemented by a T'_{i-1}-size d'_{i-1}-depth circuit, where

   $$T'_{i-1} = T_{i-1} \cdot \mathrm{poly}(n_{i-1}) \quad \text{and} \quad d'_{i-1} = d_{i-1} + \mathrm{poly}(n_{i-1}).$$

2. We then set H_i as the hitting set from Theorem 6.3.1 constructed with the Turing machine describing the circuit BF_{i-1} and output length n_i.[18] By Theorem 6.3.1, H_i can be implemented by a T_i-size d_i-depth circuit, where

   $$T_i = \mathrm{poly}(T'_{i-1}) \quad \text{and} \quad d_i = O(d'_{i-1} \cdot \log T'_{i-1} + \log^2 T'_{i-1}) + \mathrm{poly}(n_i).$$

   (Here we are being informal, see below for a more precise description.)

Formal construction. Next, we carefully detail the construction. Let $\mu \in \mathbb{N}_{\geq 1}$ be a large enough constant. First, we define a Turing machine $\mathsf{TM}_{\mathsf{H}_0}$ of description size μ that describes a T_0-size d_0-depth circuit C_{H_0} for H_0 on input 1^{n_0} in $\mu \log T_0$ space. Formally

$$\mathsf{Circuit}[T_0, \mu \cdot \log T_0, n_0, \sqrt{T_0} \cdot n_0](\mathsf{TM}_{\mathsf{H}_0}) = C_{\mathsf{H}_0}.$$

Let $\tau \in \mathbb{N}$ be a large enough constant such that the running time of A_Q on n-bit inputs is bounded by $n^{\tau/3}$.

We will make sure all H_i have exactly $\sqrt{T_i}$ elements. (This is satisfied for $i = 0$ since $T_0 = 2^{2n_0}$.)

[18] We do not discuss how to construct the Turing machine here; the details can be found in the formal construction below.

Now, for each $i \in [t]$, we will define a Turing machine $\mathsf{TM}_{\mathsf{H}_i}$ such that

$$\mathsf{Circuit}[T_i, \mu \cdot \log T_i, n_i, \sqrt{T_i} \cdot n_i](\mathsf{TM}_{\mathsf{H}_i}) = C_{\mathsf{H}_i},$$

where C_{H_i} has depth at most d_i. We will also ensure the invariant that $|\mathsf{TM}_{\mathsf{H}_i}| \leq \mu \cdot \log T_i$. By our choice of μ, the above is satisfied when $i = 0$. The machine $\mathsf{TM}_{\mathsf{H}_i}$ is defined in two steps: In the first step, we define a machine $\mathsf{TM}_{\mathsf{BF}_{i-1}}$ describing the circuit BF_{i-1}, and in the second step, we plug $\mathsf{TM}_{\mathsf{BF}_{i-1}}$ into Theorem 6.3.1 to obtain the machine $\mathsf{TM}_{\mathsf{H}_i}$.

A Turing machine $\mathsf{TM}_{\mathsf{BF}_{i-1}}$ for BF_{i-1}. We first define a Turing machine $\mathsf{TM}_{\mathsf{BF}_{i-1}}$ such that $\mathsf{TM}_{\mathsf{BF}_{i-1}}(1^{n_{i-1}})$ outputs a circuit for the algorithm BF_{i-1}. Recall that BF_{i-1} works as follows: Enumerate every element in $\mathsf{H}_{i-1} \subseteq \{0,1\}^{n_{i-1}}$ and output the first element that is in $Q_{n_{i-1}}$; if no such element exists, then $\mathsf{BF}_{i-1}(n)$ outputs $\perp$;

Using the assumed polynomial-time algorithm A_Q for deciding membership in Q, we first construct a Turing machine $\mathsf{TM}_{\mathsf{test}}$ with description size μ such that

$$C_{\mathsf{test}} = \mathsf{Circuit}\Big[T_{i-1} \cdot (n_{i-1})^{\tau/2}, \mu \cdot \log T_{i-1}, \sqrt{T_{i-1}} \cdot n_{i-1}, n_{i-1}\Big](\mathsf{TM}_{\mathsf{test}})$$

has depth $(n_{i-1})^{\tau/2}$, takes a list of $(T_{i-1})^{1/2}$ strings from $\{0,1\}^{n_{i-1}}$, and outputs the lexicographically first one in $Q_{n_{i-1}}$ (if no such string exists, outputs $\perp$ instead).

Applying Fact 6.2.4 to compose $C_{\mathsf{H}_{i-1}}$ and C_{test}, we obtain the desired Turing machine $\mathsf{TM}_{\mathsf{BF}_{i-1}}$ that constructs a circuit $C_{\mathsf{BF}_{i-1}}$ computing BF_{i-1}. Noting that μ is sufficiently large, we have that $\mathsf{TM}_{\mathsf{BF}_{i-1}}$ takes

$$2 \cdot \left(\left|\mathsf{TM}_{\mathsf{H}_{i-1}}\right| + \mu + \log n_{i-1} + \log T_{i-1}\right) \leq 3\mu \cdot \log T_{i-1}$$

bits to describe and uses

$$2 \cdot (\mu \cdot \log T_{i-1} + \mu \cdot \log T_{i-1} + \log T_{i-1}) + \mu \leq 5\mu \cdot \log T_{i-1}$$

space. We now set $T'_{i-1} = T_{i-1} \cdot n_{i-1}^{\tau}$ and $d'_{i-1} = d_{i-1} + n_{i-1}^{\tau}$, and we have

$$\mathsf{Circuit}\big[T'_{i-1}, 5\mu \cdot \log T_{i-1}, n_{i-1}, n_{i-1}\big](\mathsf{TM}_{\mathsf{BF}_{i-1}}) = C_{\mathsf{BF}_{i-1}},$$

where $C_{\mathsf{BF}_{i-1}}$ has depth at most d'_{i-1}.

The Turing machine $\mathsf{TM}_{\mathsf{H}_i}$ for H_i. Recall that H_i is defined as the hitting set H^{ct} of Theorem 6.3.1 constructed with the circuit BF_{i-1} and output length n_i in the informal argument. We now formally define H_i as the hitting set

$$\mathsf{H}^{\mathsf{ct}}_{n_{i-1}, T'_{i-1}, d'_{i-1}, n_i, \kappa, \rho}(\mathsf{TM}_{\mathsf{BF}_{i-1}}).$$

To apply Theorem 6.3.1, we first need to ensure that

$$5\mu \cdot \log T_{i-1} \leq \kappa \log T'_{i-1},$$

which is satisfied by setting $\kappa \geq 5\mu$. We also need to ensure that

$$n_{i-1} \leq T'_{i-1}, \quad d'_{i-1} \leq T'_{i-1}, \quad \text{and} \quad c \cdot \log T'_{i-1} \leq n_i \leq (T'_{i-1})^{1/(c\rho)}. \tag{6.1}$$

By Theorem 6.3.1, we know that

$$\mathsf{TM}_{\mathsf{H}_i} = \mathbb{A}^{\mathsf{ct}}_{n_{i-1},T'_{i-1},d'_{i-1},n_i,\kappa,\rho}(\mathsf{TM}_{\mathsf{BF}_{i-1}})$$

describes a T_i-size, d_i-depth circuit C_{H_i} such that $C_{\mathsf{H}_i}(1^{n_{i-1}})$ computes H_i. Moreover, $\mathsf{TM}_{\mathsf{H}_i}$ takes $c \cdot \kappa \cdot \log T'_{i-1} \leq \mu \cdot \log T_i$ space and $c \cdot \log T_i$ bits to describe, where

$$T_i = (T'_{i-1})^{c\cdot\kappa} \quad \text{and} \quad d_i = c \cdot (d'_{i-1} \log T'_{i-1} + \kappa^2 \cdot \log^2 T'_{i-1}) + n_i^c.$$

Formally, we have

$$C_{\mathsf{H}_i} = \mathsf{Circuit}[T_i, \mu \cdot \log T_i, n_i, \sqrt{T_i} \cdot n_i](\mathsf{TM}_{\mathsf{H}_i})$$

as desired. Our invariant on $|\mathsf{TM}_{\mathsf{H}_i}|$ is satisfied by setting $\mu > c$.

Analysis of T_i and d_i and justification of (6.1). We set t to be the first integer such that

$$n_{t+1} > T_t^{1/(c\rho)}.$$

In the following, we first show that $t \leq \log n_0$.

We first analyse the growth of T_i and T'_i. For every $i < t$, by our choice of t, we have that $n_i < n_{i+1} \leq T_i^{1/(c\rho)} < T_i$ and hence $T'_i = T_i \cdot n_i^\tau \leq T_i^{\tau+1}$. Then, from $T_{i+1} = (T'_i)^{c\cdot\kappa}$, we have $T_{i+1} \leq T_i^{c\cdot(\tau+1)\cdot\kappa}$ and consequently $\log T_{i+1} \leq c \cdot (\tau+1) \cdot \kappa \cdot \log T_i$. Letting $\lambda = c \cdot (\tau+1) \cdot \kappa$, we have

$$\log T_i \leq \lambda^i \cdot \log T_0 = \lambda^i \cdot 2n_0$$

for every $i \leq t$.

Recall that $n_i = n_{i-1}^\beta$, we have $\log n_i = \beta^i \cdot \log n_0$. For $T_t < n_t$ to hold, we only need to ensure the following:

$$\begin{aligned} &\lambda^i \cdot 2n_0 < \beta^i \cdot \log n_0 \\ \iff &2n_0/\log n_0 < (\beta/\lambda)^i. \end{aligned}$$

Now we will set $\beta \geq 100\lambda$. Let $\bar{t} \leq \log n_0$ be the first integer satisfying the above. We claim that $t \leq \bar{t}$. Since otherwise $\bar{t} < t$, and we would have $n_{\bar{t}} > T_{\bar{t}}$ (which certainly implies $n_{\bar{t}+1} > T_{\bar{t}}^{1/(c\rho)}$) by our choice of $\bar{t}$. This contradicts our choice of t. Therefore, we have established that $t \leq \log n_0$.

Now we turn to analyse d_i for $i \leq t$. Note that $d_0 = 2n_0$, and for $i \geq 1$, we have

$$d_i = O\big((d_{i-1} + n_{i-1}^\tau) \cdot \log T'_{i-1} + \log^2 T'_{i-1}\big) + n_i^c.$$

We will show that for every $i < t$, $d_i \leq 2n_i^c$. Clearly, this holds for $i = 0$.

Since $\log T'_{i-1} \leq \log T_{i-1} + O(\log n_{i-1}) \leq \lambda^{i-1} \cdot 2n_0 + O(\log n_{i-1}) \leq n_{i-1}$ (recall here that

$n_{i-1} = (n_0)^{\beta^{i-1}}$ and $\beta = 100\lambda$), we have

$$d_i \leq O\big((n_{i-1} + n_{i-1}^{\tau}) \cdot n_{i-1} + n_{i-1}^2\big) + n_i^c.$$

We can set β large enough so that $d_i \leq (n_{i-1})^{\beta} + n_i^c \leq 2 \cdot n_i^c$. From definition, we also have $d'_i \leq 2n_i^c + n_i^{\tau}$ for every $i < t$.

Now we are ready to justify that the conditions from (6.1) are satisfied for $i \in [t]$. By our choice of t and the definition of T'_{i-1}, we have $n_{i-1} \leq T_{i-1} \leq T'_{i-1}$. To see $d'_{i-1} \leq T'_{i-1}$ holds, recall that $T'_{i-1} = T_{i-1} \cdot n_{i-1}^{\tau}$, and we have $d'_{i-1} \leq 2n_{i-1}^c + n_{i-1}^{\tau} \leq T_{i-1} \cdot n_{i-1}^{\tau} = T'_{i-1}$ by setting $\tau > c$. We also have that $c \log T'_{i-1} = c(\log T_{i-1} + \tau \log n_{i-1}) = c(\lambda^i \cdot 2n_0 + \tau \log n_{i-1}) < n_i$ since $n_0 < (n_i)^{1/\beta}$ and $\lambda^i \leq \log_{n_0} n_i$. Finally, by our choice of t, we have $n_i \leq T_{i-1}^{1/(c\rho)} < \big(T'_{i-1}\big)^{1/(c\rho)}$.

Informal argument of the correctness. We first give a somewhat informal argument below, and then give the precise argument later.

We will argue that for every $\ell \in \mathbb{N}$, there exists an $i \in \{0, 1, \ldots, t^{(\ell)}\}$ that our polynomial-time pseudodeterministic algorithm for constructing an element from Q works on input length $n_i^{(\ell)}$.

Let $i \geq 0$ be the largest integer such that $\mathsf{H}_i \subseteq \{0,1\}^{n_i}$ is a hitting set of Q_{n_i}. (Note that such i exists, since $\mathsf{H}_0 = \{0,1\}^{n_0}$ is a hitting set of Q_{n_0}.) If $i = t$, then we can simply run BF_t to obtain an element in Q_{n_t} deterministically. Note that this takes time $\mathrm{poly}(T_t) = \mathrm{poly}(n_t)$, since by our choice of t, $T_t \leq n_t^{c \cdot \beta \cdot \rho}$.

Otherwise, we have $i < t$. In this case, we know that $Q_{n_{i+1}}$ avoids the hitting set H_{i+1} (here we use the fact that $Q_{n_{i+1}}$ accepts more than an $n_{i+1}^{-\rho}$ fraction of strings from $\{0,1\}^{n_{i+1}}$). By the reconstruction part of Theorem 6.3.1, there is a $\mathrm{poly}(n_{i+1}) \cdot d'_i$ randomised time algorithm that simulates BF_i with probability at least $1 - 2^{n_{i+1}}$. Since H_i is a hitting set for Q_{n_i}, this gives us a pseudodeterministic algorithm with $\mathrm{poly}(n_{i+1})$ time that finds a canonical element in Q_{n_i}. Since $n_{i+1} = \mathrm{poly}(n_i)$, our pseudodeterministic algorithm runs in polynomial time.

Formal description of the algorithm B. First, note that by our choice of t and β, it holds that $n_0^{(\ell+1)} > n_{t^{(\ell)}}^{(\ell)}$. On an input length $n \in \mathbb{N}_{\geq 1}$, our algorithm B is defined as follows:

1. Given input 1^n for $n \in \mathbb{N}_{\geq 1}$.
2. Compute the largest $\ell \in \mathbb{N}$ such that $n_0^{(\ell)} \leq n$, then compute the largest i such that $n_i^{(\ell)} \leq n$. Output $\perp$ and abort immediately if $n_i^{(\ell)} \neq n$. From now on we use n_i, T_i, d_i, etc. to denote $n_i^{(\ell)}, T_i^{(\ell)}, d_i^{(\ell)}$, etc.
3. For every $j \in \{0, 1, \ldots, i\}$, compute $T_j, T'_j, d_j, d'_j, \mathsf{TM}_{\mathsf{H}_j}, \mathsf{TM}_{\mathsf{BF}_j}$. There are two cases:
   - Case I: $n_{i+1} \leq T_i^{1/(c\rho)}$: In this case, we have that $i < t$. Run
   $$\big(\mathsf{R}^{\mathsf{ct}}\big)^{Q_{n_{i+1}}}_{n_i, T'_i, d'_i, n_{i+1}, \kappa, \rho}(\mathsf{TM}_{\mathsf{BF}_i})$$
   and set z_n to be its output.

- Case II: $n_{i+1} > T_i^{1/(c\rho)}$: In this case, we have that $t \leq i$. Compute t first (recall that t is the first integer such that $n_{t+1} > T_t^{1/(c\rho)}$). Output $\perp$ and abort immediately if $i > t$. Otherwise, construct C_{BF_i} from $\mathsf{TM}_{\mathsf{BF}_i}$ and set $z_n = C_{\mathsf{BF}_i}(1^n)$.

4. Output z_n if $A_Q(z_n) = 1$ and $\perp$ otherwise.

From our choice of parameters and Theorem 6.3.1, the algorithm B runs in $\mathrm{poly}(n)$ time.

Analysis of the algorithm B. Finally we show that the algorithm B satisfies our requirements. We call an input length $n \in \mathbb{N}_{\geq 1}$ *valid* if there exist $\ell \in \mathbb{N}$ and $i \in \{0, \ldots, t^{(\ell)}\}$ such that $n = n_i^{(\ell)}$, and we call n *invalid* otherwise.[19] For every $n \in \mathbb{N}_{\geq 1}$, let y_n be the lexicographically first element in Q_n.

For every invalid $n \in \mathbb{N}_{\geq 1}$, we simply set $x_n = y_n$. For every valid $n \in \mathbb{N}_{\geq 1}$, we set x_n as follows:

$$x_n = \begin{cases} C_{\mathsf{BF}_i}(1^{n_i}), & \text{if } C_{\mathsf{BF}_i}(1^{n_i}) \in Q_{n_i}, \\ y_n, & \text{if otherwise.} \end{cases}$$

We first observe that for all invalid $n \in \mathbb{N}_{\geq 1}$, it holds that $B(1^n) = \perp$ with probability 1. Now we are ready to show that for every $n \in \mathbb{N}_{\geq 1}$, $\Pr_B[B(1^n) \notin \{x_n, \perp\}] \leq 2^{-n}$. Clearly, we only need to consider valid n.

Fix a valid $n \in \mathbb{N}_{\geq 1}$. From the soundness of the reconstruction part of Theorem 6.3.1, it follows that $z_n \in \{C_{\mathsf{BF}_i}(1^n), \perp\}$ with probability at least $1 - 2^{-n}$ (if $i = t$, then $z_n = C_{\mathsf{BF}_i}(1^n)$ with probability 1). If $C_{\mathsf{BF}_i}(1^{n_i}) \in Q_{n_i}$, then $x_n = C_{\mathsf{BF}_i}(1^{n_i})$ and $z_n \in \{x_n, \perp\}$ with high probability; otherwise we have $z_n = \perp$. In both cases, the soundness of B holds.

Next, we show that for infinitely many $n \in \mathbb{N}_{\geq 1}$, we have $\Pr_B[B(1^n) = x_n] \geq 1 - 2^{-n}$. Following the informal argument, for every $\ell \in \mathbb{N}$, let $i \geq 0$ be the largest integer such that $\mathsf{H}_i \subseteq \{0,1\}^{n_i^{(\ell)}}$ is a hitting set of $Q_{n_i^{(\ell)}}$. Letting $n = n_i^{(\ell)}$, we will show that $B(1^n)$ outputs x_n with probability at least $1 - 2^{-n}$, which would finish the proof.

If $i = t$, since H_i is a hitting set for Q_n, it follows that $z_n = C_{\mathsf{BF}_i}(1^n) \in Q_n$, and we have $B(1^n) = x_n$ with probability 1. If $i < t$, we know that $Q_{n_{i+1}}$ $(1/n_{i+1}^{\rho})$-avoids the hitting set H_{i+1}. By the completeness of the reconstruction part of Theorem 6.3.1, we have that $z_n = (\mathsf{R}^{\mathsf{ct}})^{Q_{n_{i+1}}}_{n_i, T_i', d_i', n_{i+1}, \kappa, \rho}(\mathsf{TM}_{\mathsf{BF}_i})$ equals $C_{\mathsf{BF}_i}(1^n)$ with probability at least $1 - 2^{-n}$. Moreover, in this case, since H_i is a hitting set of Q_n, we know $z_n \in Q_n$ and $z_n = x_n$, which completes the proof. □

Let B be the algorithm given by Theorem 6.1.1. We note that, by using 1 bit of advice to encode if a given input length n satisfies $\Pr_B[B(1^n) = x_n] \geq 1 - 2^{-n}$, we can obtain an efficient algorithm that outputs a canonical answer with high probability (*i.e.*, satisfies the promise of a pseudodeterministic algorithm) *on all input lengths* and is correct on infinitely many of them. We state the result below as it might be useful in future work.

[19] By our choice of parameters, such pair (ℓ, i) is unique for a valid n.

Corollary 6.3.2 (Pseudodeterministic Polynomial-Time Construction with 1 Bit of Advice that Succeeds Infinitely Often)**.** *Let Q be a dense and easy language. There exist a polynomial-time probabilistic algorithm A and a sequence of advice bits $\{\alpha_i \in \{0,1\}\}_{i \in \mathbb{N}_{\geq 1}}$ such that*

- *for all $n \in \mathbb{N}_{\geq 1}$, $A(1^n, \alpha_n)$ outputs a canonical $x_n \in \{0,1\}^n$ with probability at least $1-2^{-n}$, and*
- *for infinitely many $n \in \mathbb{N}_{\geq 1}$, $x_n \in Q \cap \{0,1\}^n$.*

6.4 Modified Shaltiel–Umans Generator with Uniform Learning Reconstruction

In order to prove Theorem 6.3.1, we will need the following result.

Theorem 6.4.1 (A HSG with Uniform Learning Reconstruction)**.** *There exist an algorithm H and a probabilistic oracle algorithm $\mathsf{R}^{(-)}$ such that the following holds. Let p be a nice power of 2, m be a power of 3, $\Delta, M \in \mathbb{N}$ with $p > \Delta^2 m^7 M^9$, and let $\vec{\ell} := (p, m, M, \Delta)$ be the input parameters.*

- *The generator $\mathsf{H}_{\vec{\ell}}$ takes as input a polynomial $P \colon \mathbb{F}_p^m \to \mathbb{F}_p$ with total degree at most Δ, specified as a list of p^m evaluations of P on all points from $\mathbb{F}_p^m$ in the lexicographic order, and outputs a set of strings in $\{0,1\}^M$. Moreover, $\mathsf{H}_{\vec{\ell}}$ can be implemented by a logspace-uniform circuit of size $\mathrm{poly}(p^m)$ and depth $\mathrm{poly}(\log p, m, M)$.*
- *The reconstruction algorithm $\mathsf{R}_{\vec{\ell}}^{D,P}$, where $D \colon \{0,1\}^M \to \{0,1\}$ is any function that $(1/M)$-avoids $\mathsf{H}_{\vec{\ell}}(P)$, runs in time $\mathrm{poly}(p, m)$ and outputs, with probability at least $1-1/p^m$, a D-oracle circuit that computes P.*

The rest of this section is dedicated to the proof of Theorem 6.4.1.

6.4.1 Technical Tools

Error-Correcting Codes

Theorem 6.4.2 (List-Decoding Reed–Solomon Codes [Sud97])**.** *Let b, a, and d be integers such that $a > \sqrt{2d \cdot b}$. Given b distinct pairs (x_i, y_i) in a field $\mathbb{F}$, there are at most $2 \cdot b/a$ polynomials g of degree d such that $g(x_i) = y_i$ for at least a pairs. Furthermore, a list of all such polynomials can be computed in time $\mathrm{poly}(b, \log |\mathbb{F}|)$.*

In particular, if $a = \alpha \cdot b$ for some $0 < \alpha \leq 1$, provided that $\alpha > \sqrt{2d/b}$, there are at most $O(1/\alpha)$ degree-d polynomials that agree with an α-fraction of the b pairs.

Generator Matrices

Definition 6.4.3 (Generator Matrices)**.** Let p be a power of 2 and $m \in \mathbb{N}$. We say that $A \in \mathbb{F}_p^{m \times m}$ is a *generator matrix* for $\mathbb{F}_p^m$ if A is invertible, $A^{p^m - 1} = I$, and $\{A^i \cdot \vec{v}\}_{1 \leq i < p^m} = \mathbb{F}_p^m \setminus \{\vec{0}\}$ for any nonzero $\vec{v} \in \mathbb{F}_p^m$.[20]

[20] In fact, it is not hard to see that the third condition implies the first two. We include those two conditions in this definition as they will be useful later.

Theorem 6.4.4 ([Sho92])**.** *Let $n \in \mathbb{N}$. Given any irreducible polynomial f of degree n over $\mathbb{F}_2$, one can deterministically construct in time* $\mathrm{poly}(n)$ *a set S_n that contains at least one primitive root of the multiplicative group of $\mathbb{F}_2[\mathbf{x}]/(f)$.*

We need the following lemma to deterministically construct generator matrices. Note that it is unclear how to deterministically construct a single generator matrix. Instead, we reduce the task of constructing such matrices to the task of constructing primitive roots of $\mathbb{F}_{p^m}$. Then, we invoke Theorem 6.4.4 to construct a *set* of matrices that contains at least one generator matrix. It turns out that this set of matrices suffices for our purposes.

Lemma 6.4.5. *Let p be a nice power of 2 and m be a power of 3. One can deterministically construct in time* $\mathrm{poly}(\log p, m)$ *a set of matrices in $\mathbb{F}_p^{m\times m}$ that contains at least one generator matrix for $\mathbb{F}_p^m$.*

Proof Sketch. Roughly speaking, every primitive root of $\mathbb{F}_{p^m}$ corresponds to a generator matrix for $\mathbb{F}_p^m$, so the lemma is implied by Theorem 6.4.4.

First, if we let $p = 2^{2\cdot 3^\alpha}$ and $m = 3^\beta$, where $\alpha, \beta \in \mathbb{N}$, then the fields $\mathbb{F}_p$ and $\mathbb{F}_{p^m}$ have explicit representations

$$\mathbb{F}_{p^m} = \frac{\mathbb{F}_2[\mathbf{x}]}{\left(\mathbf{x}^{2\cdot 3^{\alpha+\beta}} + \mathbf{x}^{3^{\alpha+\beta}} + 1\right)} \quad \text{and} \quad \mathbb{F}_p = \frac{\mathbb{F}_2[\mathbf{y}]}{\left(\mathbf{y}^{2\cdot 3^{\alpha}} + \mathbf{y}^{3^{\alpha}} + 1\right)}.$$

Note that $\mathbb{F}_{p^m}$ can be viewed as a linear space over $\mathbb{F}_p$ of dimension m (i.e., $\mathbb{F}_p^m$) by identifying $\mathbf{x}^{3^\beta}$ with $\mathbf{y}$. The (field) addition operation over $\mathbb{F}_{p^m}$ coincides with the (linear space) addition operation over $\mathbb{F}_p^m$. For every element $g \in \mathbb{F}_{p^m}$, multiplication by g corresponds to a linear transformation over $\mathbb{F}_p^m$, i.e., there is a matrix A_g computable in polynomial time given g such that for every $a \in \mathbb{F}_{p^m}$, ga (as the product of two elements in the field $\mathbb{F}_{p^m}$) is equal to $A_g \cdot a$ (as a matrix-vector product over the vector space $\mathbb{F}_p^m$). It is easy to see that if g is a primitive root of the multiplication group $\mathbb{F}_{p^m}$, then A_g is a generator matrix for $\mathbb{F}_p^m$. The lemma now follows from Theorem 6.4.4. □

Random Self-Reducibility for Discrete Logarithm

Lemma 6.4.6. *There is a probabilistic polynomial-time oracle algorithm $\mathsf{DLCorr}^{(-)}$ such that the following holds. Let p be a power of 2, $m \in \mathbb{N}$, $\varepsilon > 0$, A be a generator matrix for $\mathbb{F}_p^m$, and let g be any probabilistic procedure that satisfies*

$$\Pr_{\vec{v}\leftarrow \mathbb{F}_p^m\setminus\{\vec{0}\},\, g}\left[g(\vec{v}) \text{ outputs } i \in [p^m - 1] \text{ such that } A^i \cdot \vec{1} = \vec{v}\right] \geq \varepsilon.$$

Then for every $\vec{u} \in \mathbb{F}_p^m \setminus \{\vec{0}\}$, $\mathsf{DLCorr}^g(p, m, 1^{\lceil 1/\varepsilon\rceil}, A, \vec{u})$ outputs $\ell \in [p^m - 1]$ such that $A^\ell \cdot \vec{1} = \vec{u}$ with probability at least $2/3$.

Proof Sketch. The algorithm is an adaptation of the worst-case to average-case reduction for the discrete logarithm problem. Given $\vec{u} \in \mathbb{F}_p^m \setminus \{\vec{0}\}$, we pick a random $j \in [p^m - 1]$ and set $\vec{v} := A^j \cdot \vec{u}$. Let $i := g(\vec{v})$. Since $\vec{v}$ is uniformly distributed, with probability at least ε we have $A^i \cdot \vec{1} = \vec{v}$. We check if this is the case in polynomial time (note that we can compute A^i in

polynomial time by repeated squaring). Suppose this is indeed the case, then $A^i \cdot \vec{1} = \vec{v} = A^j \cdot \vec{u}$. Recall that A is invertible. If $i > j$, we output $\ell := i - j$. If $i = j$, we have $\vec{u} = \vec{1}$. In this case, we output $\ell := p^m - 1$. Finally, if $j > i$, we output $\ell := t - (j - i)$.

By sampling $O(1/\varepsilon)$ many values of j, with probability at least $2/3$, there is at least one invocation $i := g(\vec{v})$ such that $A^i \cdot \vec{1} = \vec{v}$ indeed holds. Therefore, the success probability of our algorithm is at least $2/3$. □

Pseudorandom Generators from One-Way Permutations

Theorem 6.4.7 ([BM84, Yao82, GL89])**.** *There exist a deterministic oracle algorithm* $\mathsf{CryptoG}^{(-)}$ *and a probabilistic oracle algorithm* $\mathsf{Invert}^{(-)}$ *such that the following holds. Let* $s, M \in \mathbb{N}$ *be the input parameters, and let* $f\colon \{0,1\}^s \to \{0,1\}^s$ *be a permutation.*

1. $\mathsf{CryptoG}^f_{s,M}$ *outputs a set of* 2^{2s} M*-bit strings. Moreover,* $\mathsf{CryptoG}^f_{s,M}$ *can be implemented by a logspace-uniform* f*-oracle circuit of size* $\mathrm{poly}(2^s, M)$ *and depth* $\mathrm{poly}(s, M)$*.*

2. $\mathsf{Invert}^{(-)}_{s,M}$ *takes* $x \in \{0,1\}^s$ *as input and runs in* $\mathrm{poly}(s, M)$ *time. For any function* $D\colon \{0,1\}^M \to \{0,1\}$ *that* ε*-distinguishes* $\mathsf{CryptoG}^f_{s,M}$ *from* $\{0,1\}^M$*, we have*

$$\Pr_{x \leftarrow \{0,1\}^s}\left[\mathsf{Invert}^{f,D}_{s,M}(x) = f^{-1}(x)\right] \geq \frac{\varepsilon}{\mathrm{poly}(M)}.$$

Proof Sketch. The generator $\mathsf{CryptoG}^{(-)}$ follows from the well-known construction of pseudorandom generators from one-way permutations using the Goldreich–Levin Theorem [GL89]. More specifically,

$$\mathsf{CryptoG}^f_{s,M} := \bigcup_{x,r \in \{0,1\}^s} \Big(\langle x, r\rangle, \langle f(x), r\rangle, \langle f(f(x)), r\rangle, \ldots, \Big\langle f^{(M-1)}(x), r\Big\rangle\Big),$$

where $\langle\cdot\rangle$ denotes the inner product mod 2 function and $f^{(i)}$ denotes the composition of f with itself i times.

The "inverting" algorithm $\mathsf{Invert}^{(-)}$ and its correctness rely on standard techniques in pseudorandomness such as the hybrid argument, Yao's theorem on the equivalence between pseudorandomness and unpredictability [Yao82], and the Goldreich–Levin decoding algorithm [GL89]. (See *e.g.*, [AB09, Section 9.3].)

Finally, to see that $\mathsf{CryptoG}^f_{s,M}$ can be implemented by a logspace-uniform f-oracle circuit of size $\mathrm{poly}(2^s, M)$ and depth $\mathrm{poly}(M)$, we first note that there is a Turing machine that given $s, M \in \mathbb{N}$ and $x, r \in \{0,1\}^s$, computes the M-bit string

$$\langle x, r\rangle, \langle f(x), r\rangle, \langle f(f(x)), r\rangle, \ldots, \langle f^{M-1}(x), r\rangle$$

in $\mathrm{poly}(s, M)$ time using f as an oracle. Then by the fact that any time-t Turing machine can be simulated by a logspace-uniform circuit of size $O(t^2)$, computing a single M-bit string in $\mathsf{CryptoG}^f_{s,M}$ can be done using a logspace-uniform f-oracle circuit of size $\mathrm{poly}(s, M)$. The desired conclusion follows from the observation that we can compute these 2^{2s} M-bit strings in parallel. □

Self-Correction for Polynomials

Theorem 6.4.8 (A Self-Corrector for Polynomials, cf. [GS92, Sud95])**.** *There is a probabilistic oracle algorithm* $\mathsf{PCorr}^{(-)}$ *such that the following holds. Let* p *be a power of* 2*,* $m, \Delta \in \mathbb{N}$ *such that* $\Delta < p/3$*. Let* $g\colon \mathbb{F}_p^m \to \mathbb{F}_p$ *be such that there exists a polynomial* P *of total degree at most* Δ *for which*

$$\Pr_{\vec{x} \leftarrow \mathbb{F}_p^m}[g(\vec{x}) \neq P(\vec{x})] \leq 1/4.$$

Then for all $\vec{x} \in \mathbb{F}_p^m$*,* $\mathsf{PCorr}^g(p, m, \Delta, \vec{x})$ *runs in time* $\mathrm{poly}(\Delta, \log p, m)$ *and outputs* $P(\vec{x})$ *with probability at least* $2/3$*.*

6.4.2 The Shaltiel–Umans Generator

We state a version of the hitting set generator constructed by Shaltiel and Umans [SU05] that will be convenient for our purposes.

Theorem 6.4.9 (Implicit in [SU05])**.** *There exist a deterministic algorithm* HSU *and a probabilistic oracle algorithm* $\mathsf{RSU}^{(-)}$ *such that the following holds. Let* p *be a power of* 2*,* $m, M, \Delta \in \mathbb{N}$ *with* $p > \Delta^2 m^7 M^9$*,* $\vec{\ell} := (p, m, M, \Delta)$ *be the input parameters, and let*

- $P\colon \mathbb{F}_p^m \to \mathbb{F}_p$ *be a polynomial with total degree at most* Δ*, given as a list of* p^m *evaluations of* P *on all points from* $\mathbb{F}_p^m$ *in lexicographic order, and*
- A *be a generator matrix for* $\mathbb{F}_p^m$*.*

Then

1. *The generator* $\mathsf{HSU}_{\vec{\ell}}(P, A)$ *outputs a set of strings in* $\{0,1\}^M$*. Moreover,* $\mathsf{HSU}_{\vec{\ell}}$ *can be implemented by a logspace-uniform circuit of size* $\mathrm{poly}(p^m)$ *and depth* $\mathrm{poly}(\log p, m)$*.*
2. *The reconstruction algorithm* $\mathsf{RSU}_{\vec{\ell}}^{D,P}(A)$*, where* $D\colon \{0,1\}^M \to \{0,1\}$ *is any function that* $(1/M)$*-avoids* $\mathsf{HSU}_{\vec{\ell}}(P, A)$*, runs in* $\mathrm{poly}(p, m)$ *time and outputs, with probability at least* $1 - 1/p^{2m}$*, a vector* $\vec{v} \in \mathbb{F}_p^m \setminus \{\vec{0}\}$ *and a* D*-oracle circuit* $C : [p^m - 1] \to \mathbb{F}_p$ *such that*

$$C(i) = P(A^i \cdot \vec{v}) \text{ for every } i \in [p^m - 1].$$

The statement of Theorem 6.4.9 and the HSG result of [SU05] differ in two aspects:

- First, we use a *polynomial* instead of a Boolean function to construct the HSG, which fits more naturally into the framework of Chen–Tell [CT21a] (see also Section 6.5).
- Second, we explicitly calculated a circuit depth upper bound for computing the HSG, which is not stated in [SU05].

Nevertheless, Theorem 6.4.9 easily follows from the arguments in [SU05]. For completeness, we review the construction of [SU05] and present a proof sketch of Theorem 6.4.9 in this subsection.

The generator. We first construct m candidate "p-ary PRGs" $G^{(0)}_{p\text{-ary}}, G^{(1)}_{p\text{-ary}}, \cdots, G^{(m-1)}_{p\text{-ary}} : \mathbb{F}_p^m \to \mathbb{F}_p^M$; note that the inputs and outputs of these "p-ary PRGs" are elements in $\mathbb{F}_p$. In particular:

$$G^{(j)}_{p\text{-ary}}(\vec{x}) = \Big(P(A^{p^j \cdot 1}\vec{x}), P(A^{p^j \cdot 2}\vec{x}), \cdots, P(A^{p^j \cdot M}\vec{x})\Big).$$

Then we convert each p-ary PRG into a (usual binary) PRG by invoking [SU05, Lemma 5.6]. More precisely, for each $0 \le j < m$, we interpret $G^{(j)}_{p\text{-ary}}$ as a PRG that takes a binary seed of length $m \log p$ and outputs M elements in $\{0,1\}^{\log p}$, using the canonical bijection $\kappa^{(\log p)}$ between $\mathbb{F}_p$ and $\{0,1\}^{\log p}$. Then, for $G^{(j)}_{p\text{-ary}} : \{0,1\}^{m \log p} \to (\{0,1\}^{\log p})^M$, given seeds $x \in \{0,1\}^{m \log p}$ and $r \in \{0,1\}^{\log p}$, we define

$$G^{(j)}(x,r) = \Big(\langle G^{(j)}_{p\text{-ary}}(x)_1, r\rangle, \langle G^{(j)}_{p\text{-ary}}(x)_2, r\rangle, \ldots, \langle G^{(j)}_{p\text{-ary}}(x)_M, r\rangle\Big).$$

Here, $\langle\cdot\rangle$ denotes the inner product mod 2 function. In other words, we combine $G^{(j)}_{p\text{-ary}}$ with the *Hadamard code* to obtain a Boolean PRG $G^{(j)} \colon \{0,1\}^{m \log p + \log p} \to \{0,1\}^M$.

Finally, our HSG will be the union of all PRGs $G^{(j)}$. That is, our algorithm $\mathsf{HSU}_{\vec{\ell}}(P, A)$ enumerates every $0 \le j < m$, $x \in \{0,1\}^{m \log p}$, $r \in \{0,1\}^{\log p}$, and prints the string $G^{(j)}(x,r)$.

To see that $\mathsf{HSU}_{\vec{\ell}}$ can be computed by a logspace-uniform low-depth circuit, we argue that given appropriate indexes j and i, the i-th bit of $G^{(j)}(x,r)$ can be computed by a logspace-uniform low-depth circuit. The bit we want to compute is

$$G^{(j)}(x,r)_i = \langle G^{(j)}_{p\text{-ary}}(x)_i, r\rangle = \langle P(A^{p^j \cdot i}\vec{x}), r\rangle,$$

where $\vec{x}$ is the vector in $\mathbb{F}_p^m$ encoded by x. By repeated squaring, we can output a (logspace-uniform) circuit of size and depth $\mathrm{poly}(\log p, m)$ that computes $A^{p^j \cdot i}$. Multiplying $A^{p^j \cdot i}$ with $\vec{x}$, indexing (*i.e.*, finding the $(A^{p^j \cdot i}\vec{x})$-th entry of P), and computing Boolean inner product have logspace-uniform circuits of size $\mathrm{poly}(M, p^m) = \mathrm{poly}(p^m)$ and depth $\mathrm{poly}(m, \log p, \log M) = \mathrm{poly}(\log p, m)$. Since we need to output $m \cdot p^{m+1}$ strings of length M and each output bit can be computed by a logspace-uniform circuit of size $\mathrm{poly}(p^m)$ and depth $\mathrm{poly}(\log p, m)$, the complexity upper bounds for computing $\mathsf{HSU}_{\vec{\ell}}$ follows.

Now we consider the reconstruction algorithm. Suppose there is an adversary $D : \{0,1\}^M \to \{0,1\}$ that $(1/M)$-avoids $\mathsf{HSU}_{\vec{\ell}}(P, A)$. It follows that D $(1/M)$-distinguishes every binary PRG $G^{(j)}$.

From distinguishers to next-element predictors. For each $0 \le j < m$, we use D to build a "next-element predictor" $D^{(j)}$ for $G^{(j)}_{p\text{-ary}}$. Since D $(1/M)$-distinguishes $G^{(j)}$, it can be used to build a next-*bit* predictor $D^{(j)}_{\mathsf{bit}}$ such that

$$\Pr_{i \leftarrow [M], x \leftarrow \{0,1\}^{m \log p}, r \leftarrow \{0,1\}^{\log p}}\Big[D^{(j)}_{\mathsf{bit}}\Big(G^{(j)}(x,r)_1, \ldots, G^{(j)}(x,r)_{i-1}\Big) = G^{(j)}(x,r)_i\Big] \ge 1/2 + 1/M^2.$$

Therefore, with probability $\ge 1/2M^2$ over $i \leftarrow [M]$ and $x \leftarrow \{0,1\}^{m \log p}$, the probability over $r \leftarrow \{0,1\}^{\log p}$ that $D^{(j)}_{\mathsf{bit}}$ predicts the i-th bit of $G^{(j)}(x,r)$ given its first $i-1$ bits correctly is at least $1/2 + 1/2M^2$. In this case, using the list-decoding algorithm for Hadamard code [GL89],

we can find a list of $O(M^4)$ elements that contains $G^{(j)}_{p\text{-ary}}(x)_i$. (In fact, the trivial list-decoding algorithm suffices here, since it runs in time $\mathrm{poly}(p)$.) We call this procedure the *next-element predictor* $D^{(j)}$; it takes as input

$$u_{M-1} := P(A^{-(M-1)p^j}\vec{x}), u_{M-2} := P(A^{-(M-2)p^j}\vec{x}), \ldots, u_1 := P(A^{-p^j}\vec{x}),$$

where $\vec{x} \leftarrow \mathbb{F}_p^m$ is a random vector. It randomly selects $i \leftarrow [M]$, invokes $D^{(j)}_{\mathsf{bit}}$ and the list-decoding algorithm for the Hadamard code, and outputs a list of $O(M^4)$ elements. With probability $\Omega(1/M^2)$ over $\vec{x} \leftarrow \mathbb{F}_p^m$ and the internal randomness of $D^{(j)}_{\mathsf{bit}}$, this list will contain $P(\vec{x})$.

We repeat $D^{(j)}$ for $O(m \log p)$ times and fix its internal randomness, so that in what follows we can assume $D^{(j)}$ is deterministic. With probability at least $1-1/(10p^{2m})$, for every $0 \le j < m$, $D^{(j)}$ will be correct in the following sense: For some $\rho := 1/\Theta(M^2 m \log p)$, $D^{(j)}$ outputs ρ^{-2} elements, and

$$\Pr_{\vec{x}\leftarrow\mathbb{F}_p^m}\left[P(\vec{x}) \in D^{(j)}(u_{M-1}, u_{M-2}, \ldots, u_1)\right] > \rho.$$

Learn Next Curve. We will use the following notation from [SU05]. Let $r := O(m \log p)$ be a parameter denoting the number of reference points, and $v := (m+1)r - 1$ denotes the degree of curves.[21] A *curve* is a polynomial $C : \mathbb{F}_p \to \mathbb{F}_p^m$ with degree v. (That is, each coordinate of C is a univariate polynomial of degree v over $\mathbb{F}_p$.) Recall that $A \in \mathbb{F}_p^{m\times m}$ is the generator matrix. We use AC to denote the curve where for each $t \in \mathbb{F}_p$, $AC(t) = A \cdot C(t)$ (note that AC is still a degree-v polynomial). We also use $P(C)$ to denote the function such that for every $t \in \mathbb{F}_p$, $P(C)(t) = P(C(t))$; the *evaluation table* of $P(C)$ is the length-p vector where for every $t \in \mathbb{F}_p$, the t-th entry of the vector is $P(C(t))$.

Now, we recall the implementation of an important subroutine called Learn Next Curve as defined in [SU05, Section 5.5]. Learn Next Curve takes as input a next curve $C : \mathbb{F}_p \to \mathbb{F}_p^m$, a set of reference points $R \subseteq \mathbb{F}_p$ of size r, a stride $0 \le j < m$, as well as input evaluations; the input evaluations consist of two parts, namely the evaluation tables of $P(A^{-ip^j}C)$ for every $1 \le i < M$ and the values of $P(C(t))$ for every $t \in R$. The intended output evaluations consist of the evaluation table of $P(C)$.

In particular, Learn Next Curve starts by obtaining a set of ρ^{-2} values

$$S_t := D^{(j)}\left(P(A^{-(M-1)p^j}C(t)), P(A^{-(M-2)p^j}C(t)), \ldots, P(A^{-p^j}C(t))\right)$$

for each $t \in \mathbb{F}_p$. Then it calls the algorithm from Theorem 6.4.2 on the pairs $\{(t,e)\}_{t\in\mathbb{F}_p, e\in S_t}$ to obtain the list of all polynomials Q such that $Q(t) \in S_t$ for many coordinates t. (This takes $\mathrm{poly}(p\rho^{-2}, \log p) \le \mathrm{poly}(p, m)$ time.) If this list contains a unique polynomial Q such that $Q(t) = P(C(t))$ for every $t \in R$, then we output this polynomial; otherwise, we output $\perp$. It is clear that Learn Next Curve runs in $\mathrm{poly}(p, m)$ time.

We say Learn Next Curve *succeeds* (on next curve, reference points, and stride), if whenever the input evaluations are the intended values, the output evaluations are also the intended values. Let

$$\varepsilon_{\mathrm{LNC}} := O(v\rho^{-1}/p)^{v/2} + (8\rho^{-3})(v \deg(P)/p)^r.$$

[21] The parameter v is set in the proof of [SU05, Lemma 5.14].

It is proven in [SU05, Lemma 5.12] that, assuming $p > 32\deg(P)v/\rho^4$, if the next curve and reference points are chosen uniformly at random, Learn Next Curve succeeds with probability $1 - \varepsilon_{\mathrm{LNC}}$. Since $\deg(P) = \Delta$, $\rho^{-1} = \Theta(M^2 m \log p)$, $v = O(m^2 \log p)$, and $p > \Delta^2 m^7 M^9$, it is indeed the case that $p > 32\deg(P)v/\rho^4$. Also note that

$$\varepsilon_{\mathrm{LNC}} \le O(\rho^3/32\deg(P))^{v/2} + (8\rho^{-3})(\rho^4/32)^r \le (1/2)^{r-1} \ll 1/(10p^{4m}).$$

A first attempt for the reconstruction algorithm would be as follows. Let $i \in [p^m - 1]$, and suppose that we want to compute $P(A^i\vec{1})$. We write i in p-ary as $i = \sum_{j=0}^{m-1} i_j p^j$ (where each $i_j \in \{0, 1, \ldots, p-1\}$). Recall that for each next curve C and stride j, given the evaluation tables of $P(A^{-kp^j}C)$ for every $1 \le k < M$, we can learn the evaluation table of $P(C)$ in one invocation of Learn Next Curve. Therefore, we proceed in m rounds, where for each $0 \le l < m$, the l-th round performs the following computation:

- Let $i' := \sum_{j=0}^{l-1} i_j p^j$. Suppose that at the beginning of the l-th round, we already know the evaluation tables of $P(A^{kp^l+i'}C)$ for each $1 \le k < M$. (For $l = 0$, these values can be hardcoded as advice; for $l \ge 1$, they should be obtained from the previous round.) We invoke Learn Next Curve $M(p-1)$ times with stride l to obtain the evaluation tables of $P(A^{kp^l+i'}C)$ for each $1 \le k < M \cdot p$. The l-th round ends here; note that we have obtained the evaluation tables required in the $(l+1)$-th round (namely $P(A^{kp^{l+1}+i_l p^l+i'}C)$ for every $1 \le k < M$).

However, there is one issue with this approach: to learn a curve C, we also need to provide Learn Next Curve with the evaluations of some reference points on C. To resolve this issue, [SU05] introduced an *interleaved learning* procedure that involves two curves C_1 and C_2. These two curves possess nice intersection properties: for certain choices of k and l, A^kC_1 and A^lC_2 intersect on at least r points. This enables us to, for example, learn the evaluation table of $P(A^lC_2)$ whenever we know the evaluation table of $P(A^kC_1)$, by using the evaluations of $P(A^kC_1)$ at reference points R, where R is the intersection of A^kC_1 and A^lC_2.

Interleaved learning. In what follows, we use $[C_1 \cap C_2]$ to denote the set $\{t \in \mathbb{F}_p : C_1(t) = C_2(t)\}$. We say two curves C_1 and C_2 are *good* if they satisfy the following properties:

- $C_1(1) \neq \vec{0}$;
- for all $1 \le i < p^m$ and all $0 \le j < m$, $[A^{i+p^j}C_1 \cap A^iC_2]$ and $[A^iC_1 \cap A^iC_2]$ are of size $\ge r$;
- for all $1 \le i < p^m$ and all $0 \le j < m$, Learn Next Curve succeeds given next curve $A^{i+p^j}C_1$, reference points $[A^{i+p^j}C_1 \cap A^iC_2]$, and stride j; and
- for all $1 \le i < p^m$ and all $0 \le j < m$, Learn Next Curve succeeds given next curve A^iC_2, reference points $[A^iC_1 \cap A^iC_2]$, and stride j.

By [SU05, Lemma 5.14], there is a $\mathrm{poly}(v, p)$-time randomised algorithm that, with probability $1 - 2mp^m \cdot \varepsilon_{\mathrm{LNC}} \ge 1 - 1/(10p^{2m})$, outputs two curves C_1 and C_2 that are good.

The basic step in the reconstruction algorithm is called *interleaved learning* in [SU05]. This step has the following guarantee: For a stride j, given the correct evaluation tables of

$P(A^{i-kp^j}C_1)$ and $P(A^{i-kp^j}C_2)$ for every $1 \le k < M$, we can compute the correct evaluation tables of $P(A^iC_1)$ and $P(A^iC_2)$. In particular, *interleaved learning* consists of the following two steps:

- first, we invoke Learn Next Curve with next curve A^iC_1, reference points $[A^{i-p^j}C_2 \cap A^iC_1]$, and stride j;
- then, we invoke Learn Next Curve with next curve A^iC_2, reference points $[A^iC_1 \cap A^iC_2]$, and stride j.

Note that we assume that all invocations of Learn Next Curve succeed, as this happens with high probability $(1 - 1/(10p^{2m}))$.

The reconstruction algorithm. Recall that our reconstruction algorithm needs to output two elements: a vector $\vec{v} \in \mathbb{F}_p^m \setminus \{\vec{0}\}$ and a D-oracle circuit $C : [p^m - 1] \to \mathbb{F}_p$ such that $C(i) = P(A^i \cdot \vec{v})$ for every $i \in [p^m - 1]$.

We first compute the curves C_1 and C_2 that are good with probability $1 - 1/(10p^{2m})$. Our reconstruction algorithm will be correct provided that C_1 and C_2 are good (and that we fixed good internal randomness of our next-element predictors $D^{(j)}$); this happens with probability $\ge 1 - 1/(10p^{2m}) - 1/(10p^{2m}) \ge 1 - 1/p^{2m}$. The vector we output will be $\vec{v} := C_1(1)$ (which is non-zero if C_1 and C_2 are good). It remains to output a circuit C such that for every $i \in [p^m - 1]$, $C(i) = P(A^i \cdot \vec{v})$.

Given an integer i, our circuit C first writes i in p-ary as $i = \sum_{j=0}^{m-1} i_j p^j$. Then, it proceeds in m rounds, where for each $0 \le l < m$, the l-th round performs the following:

- Let $i' := \sum_{j=0}^{l-1} i_j p^j$. Suppose that at the beginning of the l-th round, we already know the evaluation tables of $P(A^{kp^l+i'}C_1)$ and $P(A^{kp^l+i'}C_2)$ for each $1 \le k < M$. We perform interleaved learning $M(p-1)$ times with stride l to obtain the evaluation tables of $P(A^{kp^l+i'}C_1)$ and $P(A^{kp^l+i'}C_2)$ for each $1 \le k < M \cdot p$. The l-th round ends here; note that we have obtained the evaluation tables required to perform the $(l+1)$-th round (namely, $P(A^{kp^{l+1}+i_lp^l+i'}C_1)$ and $P(A^{kp^{l+1}+i_lp^l+i'}C_2)$ for every $1 \le k < M$).

Finally, after the $(m-1)$-th round, we have obtained the evaluation table of $P(A^iC_1)$, and we can simply output $P(A^iC_1(1)) = P(A^i\vec{v})$ as the answer.

Note that the interleaved learning procedure needs to invoke the next-element predictor, therefore our circuit C will be a D-oracle circuit. Also, at the beginning of the first (0-th) round, we need the evaluation tables of $P(A^kC_1)$ and $P(A^kC_2)$ for each $0 \le k < M$. Our reconstruction algorithm can simply query the polynomial P to obtain these values and hardcode them into our circuit C. It is clear that our reconstruction algorithm runs in $\mathrm{poly}(p, m)$ time and succeeds with probability $\ge 1 - 1/p^{2m}$.

6.4.3 Modified Shaltiel–Umans Generator: Proof of Theorem 6.4.1

In this subsection, we prove Theorem 6.4.1, which is restated below.

Theorem 6.4.1 (A HSG with Uniform Learning Reconstruction)**.** *There exist an algorithm* H *and a probabilistic oracle algorithm* $\mathsf{R}^{(-)}$ *such that the following holds. Let* p *be a nice power*

of 2, m *be a power of* 3, $\Delta, M \in \mathbb{N}$ *with* $p > \Delta^2 m^7 M^9$, *and let* $\vec{\ell} := (p, m, M, \Delta)$ *be the input parameters.*

- *The generator* $\mathsf{H}_{\vec{\ell}}$ *takes as input a polynomial* $P\colon \mathbb{F}_p^m \to \mathbb{F}_p$ *with total degree at most* Δ, *specified as a list of* p^m *evaluations of* P *on all points from* $\mathbb{F}_p^m$ *in the lexicographic order, and outputs a set of strings in* $\{0,1\}^M$. *Moreover,* $\mathsf{H}_{\vec{\ell}}$ *can be implemented by a logspace-uniform circuit of size* $\mathrm{poly}(p^m)$ *and depth* $\mathrm{poly}(\log p, m, M)$.
- *The reconstruction algorithm* $\mathsf{R}_{\vec{\ell}}^{D,P}$, *where* $D\colon \{0,1\}^M \to \{0,1\}$ *is any function that* $(1/M)$*-avoids* $\mathsf{H}_{\vec{\ell}}(P)$, *runs in time* $\mathrm{poly}(p, m)$ *and outputs, with probability at least* $1 - 1/p^m$, *a* D*-oracle circuit that computes* P.

Proof. One difference between our generator and the Shaltiel–Umans generator (Theorem 6.4.9) is that the reconstruction procedure in the latter only learns a circuit C_0 that computes the mapping $i \mapsto P(A^i \cdot \vec{v})$ (for some $\vec{v}$ output by the reconstruction procedure), where A is the generator matrix used in the Shaltiel–Umans construction, instead of a circuit that computes P itself. Let us assume for simplicity that the circuit C_0 computes $i \mapsto P(A^i \cdot \vec{1})$. Note that if given $\vec{x} \in \mathbb{F}_p^m \setminus \{\vec{0}\}$ (which is the input on which we intend to evaluate P), we could *efficiently* compute the value $i \in [p^m - 1]$ such that $A^i \cdot \vec{1} = \vec{x}$, then we would be able to combine this with the circuit C_0 to compute P (roughly speaking, by first computing i and then outputting $C_0(i)$). However, there are two issues with this approach:

1. First, we do not know the generator matrix A, since we need our reconstruction algorithm to be uniform and hence cannot hardcode A.
2. Second, the task of finding such i given $\vec{x}$ and A is essentially the *discrete logarithm problem*, for which no efficient algorithm is known.

To handle the first issue, we will construct our generator using the Shaltiel–Umans construction based on a generator matrix from a small set S given by Lemma 6.4.5. Then, in the reconstruction, we will try all the matrices from S, which can be generated efficiently, to obtain a list of candidate circuits. We then select from the list a circuit that is sufficiently close to P and use a self-corrector to compute P everywhere. For the second issue, we first observe that the mapping $f\colon i \mapsto A^i \cdot \vec{1}$ is a *permutation*. Treating f as a "cryptographic one-way permutation" and invoking Theorem 6.4.7, we can construct a "cryptographic pseudorandom generator", which has a uniform reconstruction algorithm. We can then combine the output of this "cryptographic pseudorandom generator" with that of the Shaltiel–Umans generator so that if there is an algorithm D that avoids this combined generator, then D can also be used to invert f efficiently! Details follow.

The construction of H. For a matrix $A \in \mathbb{F}_p^{m \times m}$, let $f_A\colon [p^m - 1] \cup \{0\} \to \mathbb{F}_p^m$ be such that

$$f_A(i) := \begin{cases} 0^n & \text{if } i = 0 \\ A^i \cdot \vec{1} & \text{if } 1 \le i < p^m. \end{cases}$$

We will also view f as a function mapping s bits to s bits, where $s := m \cdot \log p$. Also note that if A is a generator matrix for $\mathbb{F}_p^m$, then f_A is a permutation.

Let HSU be the generator from Theorem 6.4.9 and $\mathsf{CryptoG}^{(-)}$ be the generator from Theorem 6.4.7. Also, let $S \subseteq \mathbb{F}_p^{m \times m}$ be the set of matrices constructed using Lemma 6.4.5. We define

$$\mathsf{H}_{\vec{\ell}}(P) := \bigcup_{A \in S} \left(\mathsf{HSU}_{\vec{\ell}}(P, A) \bigcup \mathsf{CryptoG}_{s,M}^{f_A} \right).$$

The complexity of H. We argue that $\mathsf{H}_{\vec{\ell}}$ can be implemented by a logspace-uniform circuit of size $\mathrm{poly}(p^m)$ and depth $\mathrm{poly}(\log p, m, M)$.

First note that given A, f_A can be computed in $\mathrm{poly}(\log p, m)$ time. Then again by the fact that every time-t Turing machine can be simulated by a logspace-uniform circuit of size $O(t^2)$, f_A can be computed by a logspace-uniform circuit of size $\mathrm{poly}(\log p, m)$. This means given A, $\mathsf{CryptoG}_{s,M}^{f_A}$, which by Theorem 6.4.7 has a logspace-uniform f_A-oracle circuit of size $\mathrm{poly}(2^s, M)$ and depth $\mathrm{poly}(s, M)$, can be implemented by a logspace-uniform circuit of size $\mathrm{poly}(p^m)$ and depth $\mathrm{poly}(\log p, m, M)$, where we have used that $s = m \cdot \log p$ and $M \leq p^{1/9}$. Also, by Theorem 6.4.9, $\mathsf{HSU}_{\vec{\ell}}$ has a logspace-uniform circuit of size $\mathrm{poly}(p^m)$ and depth $\mathrm{poly}(\log p, m, M)$. To compute $\mathsf{H}_{\vec{\ell}}(P)$, we just need to compute $\mathsf{HSU}_{\vec{\ell}}(P, A)$ and $\mathsf{CryptoG}_{s,M}^{f_A}$ for all $A \in S$ in parallel, where S can also be computed in time $\mathrm{poly}(\log p, m)$ and hence has logspace-uniform circuit of size $\mathrm{poly}(\log p, m)$. This yields a logspace-uniform circuit of size $\mathrm{poly}(p^m)$ and depth $\mathrm{poly}(\log p, m, M)$ computing $\mathsf{H}_{\vec{\ell}}$.

The reconstruction. Given oracle access to the polynomial P and a function D that $(1/M)$-avoids $\mathsf{H}_{\vec{\ell}}(P)$, we want to output a D-circuit that computes P. We do this in two stages. In the first stage, we obtain a list of candidate circuits, one for each $A \in S$, that (with high probability) contains at least one circuit that computes P. In the second stage, we will select, from the list of candidate circuits, one that is sufficiently close to P and combine it with a self-corrector to obtain a circuit that computes P on all inputs.

We now describe the first stage. Let A^* be the lexicographically first matrix in S that is a generator matrix for $\mathbb{F}_p^m$, and consider the two sets

$$\mathsf{HSU}_{\vec{\ell}}(P, A^*) \quad \text{and} \quad \mathsf{CryptoG}_{s,M}^{f_{A^*}},$$

which are subsets of $\mathsf{H}_{\vec{\ell}}(P)$. Since D avoids $\mathsf{H}_{\vec{\ell}}$, it also avoids *both* $\mathsf{HSU}_{\vec{\ell}}(P, A^*)$ and $\mathsf{CryptoG}_{s,M}^{f_{A^*}}$.

Assume for a moment that we are given the matrix A^*. We will construct a circuit C_{A^*} as follows. Let $\mathsf{RSU}^{(-)}$ and $\mathsf{Invert}^{(-)}$ be the oracle algorithms from Theorem 6.4.9 and Theorem 6.4.7 respectively. We first run $\mathsf{RSU}_{\vec{\ell}}^{D,P}(A^*)$ to obtain a D-oracle circuit C'_{A^*} and some $\vec{v} \in \mathbb{F}_p^m \setminus \{\vec{0}\}$. By the property of $\mathsf{RSU}^{(-)}$ (Item 2 of Theorem 6.4.9) and the fact that D avoids $\mathsf{HSU}_{\vec{\ell}}(P, A^*)$, we get that, with probability at least $1 - 1/p^{2m}$, for every $i \in [p^m - 1]$,

$$C'_{A^*}(i) = P((A^*)^i \cdot \vec{v}). \tag{6.2}$$

Similarly, by the property of $\mathsf{Invert}^{(-)}$ (Item 2 of Theorem 6.4.7) and the fact that D avoids

$\mathsf{CryptoG}_{s,M}^{f_{A^*}}$, we get that

$$\Pr_{x \leftarrow \{0,1\}^s}\left[\mathsf{Invert}_{s,M}^{f_{A^*},D}(x) = f_{A^*}^{-1}(x)\right] \geq \frac{1}{\mathrm{poly}(M)}.$$

By combining

$$g := \mathsf{Invert}_{s,M}^{f_{A^*},D}$$

with the algorithm $\mathsf{DLCorr}^{(-)}$ from Lemma 6.4.6, we get that for every $\vec{x} \in \mathbb{F}_p^m$, with probability at least $2/3$ over the internal randomness of DLCorr^g,

$$\mathsf{DLCorr}^g\left(p, m, 1^{\mathrm{poly}(M)}, A^*, \vec{x}\right) = f_{A^*}^{-1}(\vec{x}).$$

By using standard error reduction techniques (to reduce the error from $2/3$ to $1/(10p^{2m})$) and by fixing the internal randomness (that hopefully works correctly for all p^m inputs), we can obtain, in time $\mathrm{poly}(p, m)$ and with probability at least $1 - 1/(10p^m)$, a D-oracle circuit C''_{A^*} such that for every $\vec{x} \in \mathbb{F}_p^m$,

$$C''_{A^*}(\vec{x}) = f_{A^*}^{-1}(\vec{x}). \tag{6.3}$$

That is, given $\vec{x} \in \mathbb{F}_p^m \setminus \{\vec{0}\}$, $C''_{A^*}(\vec{x})$ outputs $i \in [p^m - 1]$ such that $(A^*)^i \cdot \vec{1} = \vec{x}$. This is almost what we need except that we want the circuit to output i such that $(A^*)^i \cdot \vec{v} = \vec{x}$. We further construct such a circuit C'''_{A^*} as follow. Given $\vec{x} \in \mathbb{F}_p^m$, we first compute

$$j := C''_{A^*}(\vec{v}) \quad \text{and} \quad k := C''_{A^*}(\vec{x}).$$

That is, $\vec{v} = (A^*)^j \cdot \vec{1}$ and $\vec{x} = (A^*)^k \cdot \vec{1}$. We then output i depending on the values of j and k. On the one hand, if $j < k$, we let $i := k - j$. Then

$$(A^*)^i \cdot \vec{v} = (A^*)^{k-j} \cdot (A^*)^j \cdot \vec{1} = (A^*)^k \cdot \vec{1} = \vec{x}.$$

On the other hand, if $k \leq j$, we let $i := p^m - 1 - (j - k)$, which yields

$$(A^*)^i \cdot \vec{v} = (A^*)^{p^m - 1 - j + k} \cdot (A^*)^j \cdot \vec{1} = I \cdot (A^*)^k \cdot \vec{1} = \vec{x}.$$

Now we have a circuit C'''_{A^*} that given $\vec{x} \in \mathbb{F}_p^m \setminus \{\vec{0}\}$, outputs $i \in [p^m - 1]$ such that $(A^*)^i \cdot \vec{v} = \vec{x}$ and a circuit C'_{A^*} that given $i \in [p^m - 1]$, computes $P((A^*)^i \cdot \vec{v})$. We then construct the circuit

$$C_{A^*}(\vec{x}) := \begin{cases} P(\vec{0}) & \text{if } \vec{x} = \vec{0} \\ C'_{A^*}(C'''_{A^*}(\vec{x})) & \text{if } \vec{x} \in \mathbb{F}_p^m \setminus \{\vec{0}\}. \end{cases}$$

Note that we can hardwire the value of $P(\vec{0})$. Also notice that if both Equations 6.2 and 6.3 are true (which happens with probability at least $1 - 1/(9p^m)$), we will get that for all $\vec{x} \in \mathbb{F}_p^m$,

$$C_{A^*}(\vec{x}) = P(\vec{x}).$$

However, we don't know the matrix A^*. Instead, we will run the above procedure for each $A \in S$ to obtain a list $\mathcal{C} := \{C_A\}_{A \in S}$ of candidate circuits C_A. Then, with probability at least

$1 - 1/(9p^m)$, $\mathcal{C}$ contains at least one circuit (in particular, C_{A^*}) that computes the polynomial P.

Given the list of candidate circuits $\mathcal{C}$, we now describe the second stage. First of all, given a circuit $C_A \in \mathcal{C}$, we want to check if C_A is sufficiently close to P.

Claim 6.4.10. *There is a randomised algorithm* $\mathsf{IsClose}$ *that, given a circuit* $B\colon \mathbb{F}_p^m \to \mathbb{F}_p$, $\delta \in (0,1]$, *and oracle access to the polynomial* P, *runs in time* $\mathrm{poly}(|B|) \cdot \log(1/\delta)$ *such that*

- *if* $\Pr_{\vec{x}}[B(\vec{x}) = P(\vec{x})] = 1$, *the algorithm accepts with probability* 1, *and*
- *if* $\Pr_{\vec{x}}[B(\vec{x}) = P(\vec{x})] \leq 3/4$, *the algorithm rejects with probability at least* $1 - \delta$.

Proof of Claim 6.4.10. The algorithm picks $3\log(1/\delta)$ points uniformly at random from $\mathbb{F}_p^m$ and checks if B and P agree on all those points. If so, the algorithm accepts; otherwise, it rejects. Note that if $\Pr_{\vec{x}}[B(\vec{x}) = P(\vec{x})] \leq 3/4$, then the probability that it accepts is at most $(3/4)^{3\log(1/\delta)} < \delta$. ◇

For each $C_A \in \mathcal{C}$, we run $\mathsf{IsClose}^P(C_A, \delta := 1/(4|\mathcal{C}|p^m))$ and pick the first one that the algorithm accepts. By the fact that $\mathcal{C}$ contains at least one circuit that computes P and by the property of the algorithm $\mathsf{IsClose}$ (Claim 6.4.10), with probability at least $1 - 1/(4p^m)$, we will obtain some D-oracle circuit C_{close} such that

$$\Pr_{\vec{x} \leftarrow \mathbb{F}_p^m}[C_{\mathsf{close}}(\vec{x}) = P(\vec{x})] > 3/4. \tag{6.4}$$

Conditioned on Equation 6.4, by combining C_{close} with the self-corrector $\mathsf{PCorr}^{(-)}$ from Theorem 6.4.8, we get that for every $\vec{x} \in \mathbb{F}_p^m$, $\mathsf{PCorr}^{C_{\mathsf{close}}}(p, m, \Delta, \vec{x}) = P(\vec{x})$ with probability at least $2/3$ (over the internal randomness of $\mathsf{PCorr}^{C_{\mathsf{close}}}$). Again, by using standard error reduction techniques and by picking a randomness uniformly at random, we can obtain in time $\mathrm{poly}(p, m)$, with probability at least $1 - 1/(4p^m)$, a D-oracle circuit C that computes P.

By a union bound, the above procedure gives, with probability at least $1 - 1/p^m$, a D-oracle circuit that computes the polynomial P.

Finally, it is easy to verify that the running time is $\mathrm{poly}(p, m)$. □

6.5 Improved Chen–Tell Targeted Hitting Set Generator

In this section, we prove Theorem 6.3.1, showing how to build a reconstructive hitting set generator from any uniform low-depth circuit.

6.5.1 Layered-Polynomial Representation

The first step is to "arithmetise" our low-depth circuit into a *layered-polynomial representation*. Roughly speaking, given a (uniform) circuit C of depth d and size T, we will produce a table of size $d' \times T'$ where $d' \approx d$ and $T' = \mathrm{poly}(T)$, such that the following key properties hold:

(Low-degree.) Each row is the "truth table" of a low-degree polynomial (thus admits self-correction properties).

(Faithful representation.) Given oracle access to the d'-th row, we can compute the output of $C(1^n)$ quickly.

(Downward self-reducibility.) For each $2 \le i \le d'$, given oracle access to the $(i-1)$-th polynomial, we can quickly compute the output of the i-th polynomial on a given input. Moreover, the entries of the first row (corresponding to $i = 1$) can be computed quickly.

Later, we will use these properties of the layered-polynomial representation to compile them into a reconstructive HSG.

We now formally describe our layered-polynomial representation, which can be proved by modifying the construction in [CT21a]. In the following, letting p be a power of 2, and $f : \mathbb{F}_p^\ell \to \mathbb{F}_p$, we use $\mathtt{tt}(f)$ to denote the length-$(p^\ell \cdot \log p)$ Boolean string that consists of p^ℓ blocks, where the i-th block corresponds to the Boolean encoding of the i-th element in $\mathbb{F}_p$.

Theorem 6.5.1 (Layered-Polynomial Representation)**.** *There exist universal constants $c, c', \beta > 1$ such that the following holds. Let $\kappa \in \mathbb{N}$ and let $T, d, n, h, p \in \mathbb{N}$ all be sufficiently large such that (1) $d \le T$ and $n \le T$, and (2) h, p are both nice powers of 2 and $\log T \le h < p \le h^{27} \le T$. (Recall that p is a nice power of 2 if $p = 2^{2 \cdot 3^\lambda}$ for some $\lambda \in \mathbb{N}$.)*

Let $\vec{\ell} := (\kappa, T, d, n, h, p)$ be the input parameters, and let $\mathbb{F} := \mathbb{F}_p$. For a Turing machine TM with description size $|\mathsf{TM}| = \kappa \cdot \log T$, let

$$C_{\mathsf{TM}} := \mathsf{Circuit}[T, \kappa \cdot \log T, n, n](\mathsf{TM}).$$

Assuming $C_{\mathsf{TM}} \neq \bot$ and C_{TM} has depth at most d, there are $d' := c\kappa \cdot \log^2 T \cdot (d + \kappa^2 \log T)$ polynomials $\left(P_i^{\vec{\ell},\mathsf{TM}}\right)_{i \in [d']}$ such that the following hold (below we write $P_i^{\vec{\ell},\mathsf{TM}}$ as P_i for simplicity):

1. **(Arithmetic setting.)** *Let $H \subset \mathbb{F}$ be the first h elements of $\mathbb{F}$, and let m be the smallest power of 3 such that $h^m \ge T^{\beta\kappa}$. Each polynomial is from $\mathbb{F}^{3m}$ to $\mathbb{F}$ and has total degree at most $\Delta = c \cdot h \cdot \log^3(T)$.*

2. **(Faithful representation.)** *Fix an injective function $\mathsf{id} : [n] \to H^m$ in an arbitrary but canonical way.[22] For every $i \in [n]$, $(C_{\mathsf{TM}}(1^n))_i = P_{d'}(\mathsf{id}(i), 0^{2m})$.*

3. **(Complexity of the polynomials.)** *Let $T_{\mathsf{poly}} := T^{c \cdot \kappa}$ and $d_{\mathsf{poly}} := c \cdot (d \log T + \kappa^2 \log^2 T)$. There is a Turing machine $\mathsf{TM}_{\mathsf{poly}}$ of description length $\log T_{\mathsf{poly}}$ such that for*

$$C_{\mathsf{poly}} := \mathsf{Circuit}\left[T_{\mathsf{poly}}, \log T_{\mathsf{poly}}, \log d', |\mathbb{F}|^{3m} \cdot \log |\mathbb{F}|\right](\mathsf{TM}_{\mathsf{poly}}),$$

   it holds that (1) for every $i \in [d']$ $C_{\mathsf{poly}}(i) = \mathtt{tt}(P_i)$ and (2) C_{poly} has depth d_{poly}.

   Moreover, there is a polynomial-time algorithm $\mathbb{A}_{\vec{\ell}}^{\mathsf{poly}}$ that takes $\mathsf{TM} \in \{0,1\}^{\kappa \log T}$ as input, and outputs the description of $\mathsf{TM}_{\mathsf{poly}}$.

4. **(Downward self-reducibility.)** *There is a $\max(n, h) \cdot h^{c'}$-time algorithm Base that takes inputs $\vec{\ell}$, $\mathsf{TM} \in \{0,1\}^{\kappa \cdot \log T}$, and $\vec{w} \in \mathbb{F}^{3m}$, outputs $P_1(\vec{w})$.*

[22] For simplicity, we will ignore the complexity of computing id and its inverse since it is negligible.

Also, there is an $h^{c'}$-time oracle algorithm DSR *that takes inputs* $\vec{\ell}$, $\mathsf{TM} \in \{0,1\}^{\kappa \cdot \log T}$, $i \in \{2, \ldots, d'\}$, *and* $\vec{w} \in \mathbb{F}^{3m}$, *and oracle access to a polynomial* $\widetilde{P} \colon \mathbb{F}^{3m} \to \mathbb{F}$, *such that when it is given* P_{i-1} *as the oracle, it outputs* $P_i(\vec{w})$.

Proof. Recall that we use $\vec{\ell}$ to denote the input parameters (κ, T, d, n, h, p). We will follow the proof of [CT21a, Proposition 4.7], which in turn follows [Gol17] (see also [Gol18]). In the following, we will simply use C to denote the (low-depth) circuit $C_{\mathsf{TM}} = \mathsf{Circuit}[T, \kappa \cdot \log T, n, n](\mathsf{TM})$ for notational convenience, but we stress that C depends on both $\vec{\ell}$ and TM (and so does the later circuits constructed from C).

Construction of a Highly Uniform Circuit D

We first construct a circuit D that has better uniformity and preserves the functionality of C, i.e., $D(1^n) = C(1^n)$. Given input 1^n, D first computes a description of $C = \mathsf{Circuit}[T, \kappa \cdot \log T, n, n](\mathsf{TM})$ (represented as a $T \times T \times T$ tensor) and then computes the Eval function $\langle \langle C \rangle, n, d \rangle \mapsto C(1^n)$. Let $s := \kappa \cdot \log T$ and $s' := O(s + \log(3 \log T))$ be such that each configuration of TM on $3 \log T$-bit inputs can be described by s' bits.

The circuit D is constructed by composing the following three sub-circuits. Let $\mu \in \mathbb{N}$ be a sufficiently large universal constant. We will describe and analyse their complexities (or state the complexity bounds proved in [CT21a, Gol17]).

1. (**Computing the adjacency matrices for configurations.**) The first circuit $D^{(1)}$ takes n bits as input (which are supposed to be 1^n), outputs a list of T^3 matrices from $\{0,1\}^{2^{s'} \times 2^{s'}}$, such that the (u,v,w)-th matrix[23] $M^{(u,v,w)}$ satisfies the following condition: for every $\gamma, \gamma' \in \{0,1\}^{s'}$, $M^{(u,v,w)}[\gamma, \gamma'] = 1$ if and only if $\mathbb{A}_{\mathsf{nxt}}(\mathsf{TM}, s, (u,v,w), \gamma, \gamma')$ (*i.e.*, γ' is the configuration obtained by running TM for one step on configuration γ and input (u,v,w) with space bound s). Recall we assumed that if γ is the accepting or the rejecting configuration, then its next configuration is γ itself.

   Complexity of $D^{(1)}$. $D^{(1)}$ can be implemented by a projection (*i.e.*, depth $d_{D^{(1)}} = 2$ and size $T_{D^{(1)}} = T^3 \cdot 2^{2s'}$).[24] Moreover, from Fact 6.2.2, given $\vec{\ell}$ and TM, in polynomial time we can compute a Turing machine $\mathsf{TM}_{D^{(1)}} \in \{0,1\}^{(\kappa+\mu) \cdot \log T}$ such that

$$\mathsf{Circuit}\left[T_{D^{(1)}}, s_{D^{(1)}}, n, T^3 \cdot 2^{2s'}\right](\mathsf{TM}_{D^{(1)}}) = D^{(1)},$$

   where $s_{D^{(1)}} = \mu \cdot s'$.

2. (**Computing the adjacency relation tensor of C via matrix multiplication.**) The second circuit $D^{(2)}$ takes a list of T^3 matrices from $\{0,1\}^{2^{s'} \times 2^{s'}}$ as input, and outputs a tensor from $\{0,1\}^{T \times T \times T}$ followed by the encoding of a pair (n,d).

   In more detail, given the output of $D^{(1)}(1^n)$, for every $(u,v,w) \in [T]^3$, it determines whether $\mathsf{TM}(u,v,w) = 1$ by computing $(M^{(u,v,w)})^{2^{s'}}$, which can be done by repeated squaring s' times. This gives the adjacent relation tensor of C.

[23] We use $(u,v,w) \in [T]^3$ to denote the integer $(u-1)T^2 + (v-1)T + w \in [T^3]$.

[24] Note that we can implement projections and restrictions of input bits to 0 and 1 using two layers of NAND gates.

Complexity of $D^{(2)}$. $D^{(2)}$ can be implemented by a circuit of depth $d_{D^{(2)}} = \mu \cdot (s')^2$ and size $T_{D^{(2)}} = T^3 \cdot 2^{\mu s'}$. Moreover, from [CT21a, Gol17] (note that $D^{(2)}$ does not depend on TM), given $\vec{\ell}$, in polynomial time we can compute a Turing machine $\mathsf{TM}_{D^{(2)}} \in \{0,1\}^{\mu \cdot \log T}$ such that

$$\mathsf{Circuit}\Big[T_{D^{(2)}}, s_{D^{(2)}}, T^3 \cdot 2^{2s'}, T^3 + |(n,d)|\Big](\mathsf{TM}_{D^{(2)}}) = D^{(2)},$$

where $s_{D^{(2)}} = \mu \cdot s'$.

3. (**Computing** Eval.) The final circuit $D^{(3)}$ takes $\langle\langle C\rangle, n, d\rangle$ as input, and outputs $\mathsf{Eval}(\langle C\rangle, n, d)$.

   Complexity of $D^{(3)}$. $D^{(3)}$ can be implemented by a circuit of depth $d_{D^{(3)}} = \mu \cdot d \cdot \log T$ and size $T_{D^{(3)}} = T^{\mu}$. Moreover, from [CT21a, Gol17] (note that $D^{(3)}$ does not depend on TM), given $\vec{\ell}$, in polynomial-time we can compute a Turing machine $\mathsf{TM}_{D^{(3)}} \in \{0,1\}^{\mu \cdot \log T}$ such that

$$\mathsf{Circuit}\big[T_{D^{(3)}}, s_{D^{(3)}}, T^3 + |(n,d)|, n\big](\mathsf{TM}_{D^{(3)}}) = D^{(3)},$$

   where $s_{D^{(3)}} = \mu \cdot s'$.

Formally, we have

$$D = D^{(3)} \circ D^{(2)} \circ D^{(1)}.$$

Let $\beta \in \mathbb{N}$ be a sufficiently large constant such that $\beta \geq 100\mu$. The following claim summarizes the required properties of D for us.

Claim 6.5.2. *The following properties about the circuit D are true.*

1. *The depth of D is $d_D = \beta \cdot (\kappa^2 \cdot \log^2 T + d \cdot \log T)$ and the width of D is $T'_D = T^{\beta\kappa}$.*
2. *The layered adjacency relation function $\Phi' \colon [d_D] \times \{0,1\}^{3\log(T'_D)} \to \{0,1\}$ of D can be decided by a formula of depth $O(\log\log T)$ and size $O(\log^3 T)$. Moreover, there is an algorithm $\mathbb{A}_{\Phi'}$ that, given $\vec{\ell}$ and TM as input, outputs the formula above in $O(\kappa \log T)$ space.*
3. *There is a Turing machine $\mathsf{TM}_D \in \{0,1\}^{\beta\kappa\log T}$ such that*

$$\mathsf{Circuit}[T_D, s_D, n, n](\mathsf{TM}_D) = D,$$ [25]

   where $T_D = T'_D \cdot (d_D + 1)$ and $s_D = \beta\kappa \log T$. Moreover, given $\vec{\ell}$ and TM as input, the description of TM_D can be computed in polynomial time.

Proof of Claim 6.5.2. By construction, the size of D is bounded by $\mathrm{poly}(T) \cdot 2^{O(s')} \leq T^{O(\kappa)}$ (recall that $s' = O(s + \log(3\log T))$ and $s = \kappa \log T$), and its depth is bounded by $O(s^2 + d \cdot \log T)$. The first bullet then follows directly from the fact that β is sufficiently large.

Recall that the $D^{(1)}$ part of D has depth $d_{D^{(1)}} = 2$. To see the complexity of computing $\Phi'(i, -, -, -)$ for $i > 2$, we note that the layers corresponding to $D^{(2)}$ and $D^{(3)}$ *do not* depend on TM. Hence the complexity of computing their layered adjacent relation function follows directly

[25] Note that $\mathsf{Circuit}$ generates the unlayered version of D of size $T'_D \cdot (d_D + 1)$, Without loss of generality we can assume the first T' gates are on the first layer, the next T' gates are on the second layer, and so on.

from [CT21a, Claim 4.7.1].[26] Also, the complexity of computing $\Phi'(i,-,-,-)$ for $i \in \{1,2\}$ follows directly from Fact 6.2.2. To see the moreover part, again, the case for $i > 2$ follows from [CT21a, Claim 4.7.1], and the case for $i \in \{1,2\}$ follows from Fact 6.2.2.[27]

Finally, to obtain the algorithm that computes TM_D, we simply apply the composition $\mathbb{A}_{\mathsf{comp}}$ (from Fact 6.2.4) twice to compose the circuits $D^{(1)}, D^{(2)}, D^{(3)}$ in order and add some dummy gates to the circuit. The space bound and the description size bound can also be verified easily. ◇

Arithmetisation of D

The construction of the polynomials and their corresponding algorithms can then be done in the same way as in [CT21a]. We only state the necessary changes to establish our theorem.

Note that $|\mathbb{F}|^m = p^m \leq \mathrm{poly}(h^{27m}) \leq T^{O(\beta\kappa)} \leq T^{O(\kappa)}$ (β is a universal constant), from our assumption that $p \leq h^{27}$ and our choice of m.

First, we need an arithmetisation of each $\Phi'_i := \Phi'(i,-,-,-)$.

Claim 6.5.3. *For $i \in [d_D]$ there exists a polynomial $\hat{\Phi}_i \colon \mathbb{F}^{3m} \to \mathbb{F}$ that satisfies the following:*

1. *For every $(\vec{w},\vec{u},\vec{v}) = z_1, ..., z_{3m} \in H^{3m}$ we have that $\hat{\Phi}_i(\vec{w},\vec{u},\vec{v}) = 1$ if gate $\vec{w}$ in the i^{th} layer of D is fed by gates $\vec{u}$ and $\vec{v}$ in the $(i-1)^{th}$ layer of D, and $\hat{\Phi}_i(\vec{w},\vec{u},\vec{v}) = 0$ otherwise.*
2. *The degree of $\hat{\Phi}_i$ is at most $O(h \cdot \log^3 T)$. Moreover, there exists an algorithm that on input $\vec{\ell}, \mathsf{TM}, i, \vec{w}, \vec{u}, \vec{v}$, computes $\hat{\Phi}_i(\vec{w},\vec{u},\vec{v})$ in $\mathrm{poly}(h)$ time.*
3. *For a universal constant $c_1 > 1$, there exists a circuit $C_{\hat{\Phi}}$ of size $T_{\hat{\Phi}} := T^{c_1\kappa}$ and depth $c_1\kappa \cdot \log T$ that on input $i \in [d_D]$ outputs $\mathtt{tt}(\hat{\Phi}_i) \in \mathbb{F}^{|\mathbb{F}|^{3m}}$ (represented as a Boolean string). Moreover, there is a polynomial-time algorithm $\mathbb{A}_{\hat{\Phi}}$ that takes $\vec{\ell}$ and $\mathsf{TM} \in \{0,1\}^{\kappa \log T}$ as input, and outputs the description of a Turing machine $\mathsf{TM}_{\hat{\Phi}} \in \{0,1\}^{c_1\kappa \log T}$ such that*

$$C_{\hat{\Phi}} = \mathsf{Circuit}\big[T_{\hat{\Phi}}, c_1 \cdot \kappa \log T, \log(d_D+1), |\mathbb{F}|^{3m} \log |\mathbb{F}|\big](\mathsf{TM}_{\hat{\Phi}}).$$

Proof Sketch of Claim 6.5.3. We first define $\hat{\Phi}_i$ and then establish each item separately.

Construction of $\hat{\Phi}_i$. Let $F_{\Phi'}$ be the $O(\log\log T)$-depth $O(\log^3 T)$-size formula computing $\Phi' \colon [d_D] \times \{0,1\}^{3 \cdot \log T'_D} \to \{0,1\}$ from Claim 6.5.2. For every $i \in [d_D]$, let F_i be the restriction of $F_{\Phi'}$ by fixing the first input to be i. Then, we arithmetise F_i by replacing every $\mathtt{NAND}$ gate in F_i by an arithmetic gate $\widetilde{\mathtt{NAND}} : \mathbb{F}^2 \to \mathbb{F}$ computing $\widetilde{\mathtt{NAND}}(u,v) := 1 - uv$. Denote the new formula (which is now an *arithmetic* formula) by $\hat{F}_i$.

For each j, let $\pi_j \colon H \to \{0,1\}$ be the mapping that maps $z \in H$ to the j-th bit of the encoding of z. Note that since H consists of the smallest h elements in $\mathbb{F}$, we know that $\pi(z) = (\pi_1(z), \ldots, \pi_{\log h}(z))$ is a bijection between H and $\{0,1\}^{\log h}$.[28]

[26] We note that although [CT21a, Gol17] only claims a polylog(T) bound on the formula size, the formula is indeed very simple and its size and depth can be easily bounded by $O(\log^3 T)$ and $O(\log\log T)$, respectively; see [Gol17, Page 8-9] for details.

[27] Strictly speaking, we need to combine the formulas for two cases to obtain a single formula for Φ'. The overhead of doing so is negligible, thus we omit this discussion here.

[28] More specifically, by our specific encoding of H as strings from $\{0,1\}^{\log |\mathbb{F}|}$, $\pi(z)$ is simply the first $\log h$ bits of the encoding of z, hence it can be computed by a projection.

For each $j \in [\log h]$, let $\hat{\pi}_j \colon \mathbb{F} \to \mathbb{F}$ be the unique degree-$(h-1)$ extension of π_j to $\mathbb{F}$, that can be computed via standard interpolation via logspace-uniform circuits of $O(\log(h \cdot \log|\mathbb{F}|)) = O(\log T)$ depth and polylog(T) size [HAB02, HV06] (see [CT21a, Claim 4.7.2] for the details). We also let $\hat{\pi}(z) = (\hat{\pi}_1(z), \dots, \hat{\pi}_{\log h}(z))$. Then, we set

$$\hat{\Phi}_i(z_1, \dots, z_{3m}) := \hat{F}_i(\hat{\pi}(z_1), \hat{\pi}(z_2), \dots, \hat{\pi}(z_{3m})).$$

We also use $F_{\hat{\Phi}_i}$ to denote the *arithmetic* formula on the right side above that computes the formula $\hat{\Phi}_i$.

From the construction above, the first two items of the claim can be proved identically as [CT21a, Claim 4.7.2]. It remains to establish the third item.

Construction of $C_{\hat{\Phi}}$. We hardwire the description of $F_{\Phi'}$ into $C_{\hat{\Phi}}$. The circuit $C_{\hat{\Phi}}$ takes $i \in [d_D]$ as input, performs the above computation to obtain a description of the arithmetic formula $F_{\hat{\Phi}_i}$, and then outputs the truth table of $F_{\hat{\Phi}_i}$ by evaluating it on all vectors in $\mathbb{F}^{3m}$.

In more detail, computing the description of $F_{\hat{\Phi}_i}$ from the description of $F_{\Phi'}$ and i can be done in $O(\log T)$ depth and polylog(T) size. $C_{\hat{\Phi}}$ then evaluates $F_{\hat{\Phi}_i}$ on all vectors from $\mathbb{F}^m$, which can be done in poly$(|\mathbb{F}|^m)$ size and $O(\log(|\mathbb{F}|^m))$ depth. The third bullet (except for the moreover part) then follows by setting c_1 to be sufficiently large and recalling that $|\mathbb{F}|^m \leq T^{O(\beta\kappa)}$.

Establishing the uniformity. Finally, we establish the moreover part of the third bullet. Let $\mu_{\hat{\Phi}} \in \mathbb{N}$ be a sufficiently large universal constant that depends on the space complexity of the algorithm $\mathbb{A}_{\Phi'}$ from Claim 6.5.2.

Our algorithm $\mathbb{A}_{\hat{\Phi}}$ works as follows:

1. We first construct a Turing machine $\mathsf{TM}_{[1]}$ with $\vec{\ell}$ and TM hardwired. $\mathsf{TM}_{[1]}$ corresponds to a circuit $C_{[1]}$ that takes $i \in [d_D]$ as input and outputs i together with the description of $F_{\Phi'}$.[29] $C_{[1]}$ has depth $d_{[1]} = O(1)$ and size $T_{[1]} = \text{polylog}(T)$. Let $s_{[1]} = \mu_{\hat{\Phi}} \cdot \kappa \log T$, we have

   $$C_{[1]} = \mathsf{Circuit}\big[T_{[1]}, s_{[1]}, \log d_D, \log d_D + |F_{\Phi'}|\big]\big(\mathsf{TM}_{[1]}\big)$$

   and $\mathsf{TM}_{[1]}$ has description size at most $|\mathsf{TM}| + \mu_{\hat{\Phi}} \cdot \log T = (\kappa + \mu_{\hat{\Phi}}) \cdot \log T$.

   Here, we crucially use the fact that the algorithm $\mathbb{A}_{\Phi'}$ from Claim 6.5.2 runs in $O(\kappa \log T)$ space (and $\mu_{\hat{\Phi}}$ is sufficiently large).

2. Then we construct a Turing machine $\mathsf{TM}_{[2]}$ that corresponds to a circuit $C_{[2]}$ that takes $i \in [d_D]$ together with the description of $F_{\Phi'}$ as input and outputs $\mathtt{tt}(\hat{\Phi}_i)$. By the discussion above, from $\vec{\ell}$ we can compute a Turing machine $\mathsf{TM}_{[2]} \in \{0,1\}^{\mu_{\hat{\Phi}} \kappa \log T}$ such that for $T_{[2]} = \text{poly}(|\mathbb{F}|^m) \leq T^{\mu_{\hat{\Phi}}\kappa}$, $d_{[2]} = O(\log(|\mathbb{F}|^m)) \leq \mu_{\hat{\Phi}}\kappa \cdot \log T$, $s_{[2]} = \mu_{\hat{\Phi}}\kappa \log T$, we have

   $$C_{[2]} = \mathsf{Circuit}\big[T_{[2]}, s_{[2]}, \log d_D + |F_{\Phi'}|, |\mathbb{F}|^{3m}\big]\big(\mathsf{TM}_{[2]}\big)\ ,$$

   and $C_{[2]}$ has depth $d_{[2]}$.

[29] Precisely, $\mathsf{TM}_{[1]}$ simulates $\mathbb{A}_{\Phi'}$ on input $\vec{\ell}$ and TM to construct a projection that maps i to $(i, F_{\Phi'})$.

3. Finally, $\mathbb{A}_{\hat{\Phi}}$ composes $C_{[1]}$ and $C_{[2]}$ by applying Fact 6.2.4 and outputs the obtained Turing machine as $\mathsf{TM}_{\hat{\Phi}}$. Setting c_1 sufficiently large completes the proof. ◇

Then we define the following polynomials, according to [CT21a, Definition 4.6].

Input polynomial. Let $\alpha_0 \colon H^m \to \{0,1\}$ represent the string $1^n 0^{h^m - n}$, and let $\hat{\alpha}_0 \colon \mathbb{F}^m \to \mathbb{F}$ be the "arithmetisation" of α_0, defined by

$$\hat{\alpha}_0(\vec{w}) = \sum_{\vec{z} \in H^{m'} \times \{0\}^{m-m'}} \delta_{\vec{z}}(\vec{w}) \cdot \alpha_0(\vec{z}).$$

Here, $m' \leq m$ is the minimal integer such that $h^{m'} \geq n$, and $\delta_{\vec{z}}$ is Kronecker's delta function (*i.e.*, $\delta_{\vec{z}}(\vec{w}) = \prod_{j \in [m]} \prod_{a \in H \setminus \{z_j\}} \frac{w_j - a}{z_j - a}$).

Layer polynomials. For each $i \in [d_D]$, let $\alpha_i \colon H^m \to \{0,1\}$ represent the values of the gates at the i^{th} layer of D in the computation of $D(1^n)$ (with zeroes in locations that do not index valid gates). Recall that we consider circuits consisting of NAND gates, where for $a, b \in \{0,1\}$ we have $\mathsf{NAND}(a,b) = 1 - a \cdot b$. We define $\hat{\alpha}_i \colon \mathbb{F}^m \to \mathbb{F}$ as

$$\hat{\alpha}_i(\vec{w}) = \sum_{\vec{u}, \vec{v} \in H^m} \hat{\Phi}_i(\vec{w}, \vec{u}, \vec{v}) \cdot \left(1 - \hat{\alpha}_{i-1}(\vec{u}) \cdot \hat{\alpha}_{i-1}(\vec{v})\right). \tag{6.5}$$

Note that $\hat{\alpha}_i$ is the "arithmetisation" of α_i in the sense that for every $\vec{w} \in H^m$, $\alpha_i(\vec{w}) = \hat{\alpha}_i(\vec{w})$.

Sumcheck polynomials. For each $i \in [d_D]$, let $\hat{\alpha}_{i,0} \colon \mathbb{F}^{3m} \to \mathbb{F}$ be the polynomial

$$\hat{\alpha}_{i,0}(\vec{w}, \sigma_1, ..., \sigma_{2m}) = \hat{\Phi}_i(\vec{w}, \sigma_{1,...,m}, \sigma_{m+1,...,2m}) \cdot \left(1 - \hat{\alpha}_{i-1}(\sigma_{1,...,m}) \cdot \hat{\alpha}_{i-1}(\sigma_{m+1,...,2m})\right). \tag{6.6}$$

For every $j \in [2m]$, let $\hat{\alpha}_{i,j} \colon \mathbb{F}^{3m-j} \to \mathbb{F}$ be the polynomial

$$\hat{\alpha}_{i,j}(\vec{w}, \sigma_1, ..., \sigma_{2m-j}) = \sum_{\sigma_{2m-j+1}, ..., \sigma_{2m} \in H} \hat{\Phi}_i(\vec{w}, \sigma_{1,...,m}, \sigma_{m+1,...,2m}) \cdot \left(1 - \hat{\alpha}_{i-1}(\sigma_{1,...,m}) \cdot \hat{\alpha}_{i-1}(\sigma_{m+1,...,2m})\right), \tag{6.7}$$

where $\sigma_{k,...,k+r} = \sigma_k, \sigma_{k+1}, ..., \sigma_{k+r}$. It is easy to check that $\hat{\alpha}_{i,2m} = \hat{\alpha}_i$.

We are now ready to define the sequence $(P_i)_{i \in [d']} = \left(P_i^{\vec{\ell}, \mathsf{TM}}\right)_{i \in [d']}$. We set $d' := (2m+1) \cdot d_D + 1$ and

$$(P_i)_{i \in [d']} = (\hat{\alpha}_0, \hat{\alpha}_{1,0}, \ldots, \hat{\alpha}_{1,2m}, \hat{\alpha}_{2,0}, \ldots, \hat{\alpha}_{2,2m}, \ldots, \hat{\alpha}_{d_D,0}, \ldots, \hat{\alpha}_{d_D,2m}).$$

For those $\hat{\alpha}_{i,j}$ (and $\hat{\alpha}_0$) that take less than $3m$ variables, we add some dummy variables at the end to make all polynomials take exactly $3m$ variables.

From the definitions of m and d_D, we have $m \leq 3 \cdot \beta\kappa \cdot \log T + 1$ and $d_D = \beta \cdot (\kappa^2 \cdot \log^2 T + d \cdot \log T)$. Hence, we have $d' = (2m+1) \cdot d_D + 1 \leq c\kappa \cdot \log^2 T \cdot (d + \kappa^2 \log T)$ as desired.[30]

[30] We can add identical polynomials at the end to make d' exactly $c\kappa \cdot \log^2 T \cdot (d + \kappa^2 \log T)$ as in the theorem statement.

Below we verify the desired properties of the sequence $(P_i)_{i\in[d']}$.

Arithmetic setting, faithful representation, and downward self-reducibility. First, the degree bounds of all these polynomials follow directly from their definitions and from the degree bound on $\hat{\Phi}_i$ (from Claim 6.5.3). The faithful representation property also follows directly from the definition of α_{d_D} and $\hat{\alpha}_{d_D,2m} = \hat{\alpha}_{d_D}$. Finally, the downward self-reducibility of the polynomials follows from the complexity of computing $\hat{\Phi}_i$ (from Claim 6.5.3) and the definitions of these polynomials, similarly to the proof of [CT21a, Proposition 4.7].

Complexity of the Polynomials

Now we verify the complexity of computing these polynomials. The argument below is straightforward but tedious. We first give a high-level overview.

High-level overview of the construction. To construct the circuit C_{poly} that maps $i' \in [d']$ to $\mathtt{tt}(P_i)$, we will construct three sub-circuits C_α, $C_{\mathtt{tt}\text{-}\hat{\Phi}}$, and C_{arith} such that:

1. C_α maps $i' \in [d']$ to $(\mathtt{tt}(\alpha_{i-1}), i, j)$. Here, if $i' \geq 2$, then $i \in \{1, \ldots, d_D\}$ and $j \in \{0, 1, \ldots, 2m\}$ satisfies that $P_{i'} = \hat{\alpha}_{i,j}$ and $\mathtt{tt}(\alpha_{i-1}) \in \{0,1\}^{h^m}$ denotes the values of the gates at the i-th layer of D. If $i' = 1$, then we consider $i = j = 0$ and C_α outputs $(\mathtt{tt}(\alpha_0), 0, 0)$.
2. $C_{\mathtt{tt}\text{-}\hat{\Phi}}$ maps $(\mathtt{tt}(\alpha_{i-1}), i, j)$ to $(\mathtt{tt}(\alpha_{i-1}), i, j, \mathtt{tt}(\hat{\Phi}_i))$.
3. C_{arith} maps $(\mathtt{tt}(\alpha_{i-1}), i, j, \mathtt{tt}(\hat{\Phi}_i))$ to $\mathtt{tt}(\hat{\alpha}_{i,j}) \in \mathbb{F}^{|\mathbb{F}|^{3m}}$.

C_{poly} is then simply $C_{\mathsf{arith}} \circ C_{\mathtt{tt}\text{-}\hat{\Phi}} \circ C_\alpha$. To compute the Turing machine $\mathsf{TM}_{\mathsf{poly}}$ that corresponds to C_{poly}, we construct the Turing machines TM_α, $\mathsf{TM}_{\mathtt{tt}\text{-}\hat{\Phi}}$, and $\mathsf{TM}_{\mathsf{arith}}$ corresponding to the three circuit above, and compose them using Fact 6.2.4.

Construction of C_α and TM_α. First, we construct a circuit C_α that takes as input $i' \in [d']$ and outputs $(\mathtt{tt}(\alpha_{i-1}), i, j)$. To construct C_α, we first compute i and j from i' using basic arithmetic, and then truncate D up to its i-th layer. It is easy to see that given the Turing machine TM_D that specifies the circuit D, in polynomial-time we can construct a Turing machine $\mathsf{TM}_\alpha \in \{0,1\}^{|\mathsf{TM}_D|+\mu}$ such that (in what follows, we write $|\langle i, j\rangle| = \log(d_D + 1) + \log(2m + 1)$ for convenience):

$$\mathsf{Circuit}[T_\alpha, s_\alpha, \log(d'), h^m + |\langle i, j\rangle|](\mathsf{TM}_\alpha) = C_\alpha,$$

where $T_\alpha = \mu \cdot T_D$, $s_\alpha = 2s_D$. Moreover, C_α has depth at most $d_\alpha = 2 \cdot d_D$.

Construction of $C_{\mathtt{tt}\text{-}\hat{\Phi}}$ and $\mathsf{TM}_{\mathtt{tt}\text{-}\hat{\Phi}}$. Let c_1 be the universal constant from Claim 6.5.3. Next we construct a circuit $C_{\mathtt{tt}\text{-}\hat{\Phi}}$ that on input $(\mathtt{tt}(\alpha_{i-1}), i, j)$, outputs $(\mathtt{tt}(\alpha_{i-1}), i, j, \mathtt{tt}(\hat{\Phi}_i))$. It is straightforward to obtain this circuit from the circuit $C_{\hat{\Phi}}$ constructed in Claim 6.5.3. In other words, given $\vec{\ell}$ and $\mathsf{TM}_{\hat{\Phi}} \in \{0,1\}^{c_1 \kappa \log T}$ as input (where $\mathsf{TM}_{\hat{\Phi}}$ is the Turing machine that generates $C_{\hat{\Phi}}$ as defined in Claim 6.5.3), we can compute a Turing machine $\mathsf{TM}_{\mathtt{tt}\text{-}\hat{\Phi}} \in$

$\{0,1\}^{2\cdot c_1\kappa\log T}$ such that

$$\mathsf{Circuit}[T_{\mathtt{tt}\text{-}\hat{\Phi}}, s_{\mathtt{tt}\text{-}\hat{\Phi}}, h^m + |\langle i,j\rangle|, h^m + |\langle i,j\rangle| + |\mathbb{F}|^{3m}\log|\mathbb{F}|](\mathsf{TM}_{\mathtt{tt}\text{-}\hat{\Phi}}) = C_{\mathtt{tt}\text{-}\hat{\Phi}},$$

where $T_{\mathtt{tt}\text{-}\hat{\Phi}} = T^{2c_1\kappa}$, $s_{\mathtt{tt}\text{-}\hat{\Phi}} = 2c_1\kappa\log T$. Moreover, $C_{\mathtt{tt}\text{-}\hat{\Phi}}$ has depth $d_{\mathtt{tt}\text{-}\hat{\Phi}} = 2c_1\kappa\log T$.

Construction of C_{arith} and $\mathsf{TM}_{\mathsf{arith}}$. We construct a circuit C_{arith} that maps

$$(\mathtt{tt}(\alpha_{i-1}), i, j, \mathtt{tt}(\hat{\Phi}_i))$$

to $\mathtt{tt}(\hat{\alpha}_{i,j}) \in \mathbb{F}^{|\mathbb{F}|^{3m}}$. (Recall that throughout this proof we view $\hat{\alpha}_{i,j}$ as a $3m$-variable polynomial by adding dummy variables at the end.) Suppose that $i \geq 1$ (the base case $i = j = 0$ can be handled similarly). If $j = 0$, C_{arith} computes $\mathtt{tt}(\hat{\alpha}_{i,0})$ using Equation 6.6, otherwise ($j \geq 1$) C_{arith} computes $\mathtt{tt}(\hat{\alpha}_{i,j})$ using Equation 6.7. (Note that both Equation 6.6 and Equation 6.7 only depend on the values of $\hat{\alpha}_{i-1}$ over H^m, which is exactly $\mathtt{tt}(\alpha_{i-1})$ due to our arithmetisation.) Since arithmetic operations over $\mathbb{F}$ (including iterated addition, multiplication, and inverse) are in logspace-uniform NC^1 [HAB02, HV06],[31] it follows that C_{arith} is of $T_{\mathsf{arith}} := \mathrm{poly}(|\mathbb{F}|^m)$ size and $d_{\mathsf{arith}} := O(m\log|\mathbb{F}|)$ depth. Moreover, C_{arith} does not depend on TM, and we can compute a Turing machine $\mathsf{TM}^{\mathsf{arith}}$ from $\vec{\ell}$ in time $\mathrm{polylog}(T)$ such that

$$\mathsf{Circuit}[T_{\mathsf{arith}}, s_{\mathsf{arith}}, h^m + |\langle i,j\rangle| + |\mathbb{F}|^{3m}\log|\mathbb{F}|, |\mathbb{F}|^{3m}\log|\mathbb{F}|](\mathsf{TM}^{\mathsf{arith}}) = C_{\mathsf{arith}},$$

where $s_{\mathsf{arith}} = \mu\cdot\beta\kappa\log T$.

Composing TM_α, $\mathsf{TM}_{\mathtt{tt}\text{-}\hat{\Phi}}$, and $\mathsf{TM}_{\mathsf{arith}}$ by applying Fact 6.2.4 twice gives the desired Turing machine $\mathsf{TM}_{\mathsf{poly}}$ that computes the desired circuit C_{poly}.

Complexity of C_{poly} and $\mathsf{TM}_{\mathsf{poly}}$. Finally, we verify that $\mathsf{TM}_{\mathsf{poly}}$ and C_{poly} satisfy our requirements. First, from the discussions above, we can bound the size of C_{poly} by $T_\alpha + T_{\mathtt{tt}\text{-}\hat{\Phi}} + T_{\mathsf{arith}} \leq T_{\mathrm{poly}} = 2^{c\cdot\kappa\log T}$ by picking a sufficiently large c. Note that $m\log|\mathbb{F}| = \log(p^m) \leq O(\kappa\log T)$. The depth of C_{poly} can be bounded by $d_{\mathsf{poly}} = d_\alpha + d_{\mathtt{tt}\text{-}\hat{\Phi}} + d_{\mathsf{arith}} \leq c\cdot(\kappa^2\cdot\log^2 T + d\cdot\log T)$ as desired.

From Fact 6.2.4, we have that

$$|\mathsf{TM}_{\mathsf{poly}}| \leq 100\cdot\left(|\mathsf{TM}_\alpha| + |\mathsf{TM}_{\mathtt{tt}\text{-}\hat{\Phi}}| + |\mathsf{TM}_{\mathsf{arith}}| + \log(|\mathbb{F}|^{3m})\right) \leq c\cdot\kappa\log T = \log T_{\mathsf{poly}}$$

by setting c sufficiently large. The space complexity of $\mathsf{TM}_{\mathsf{poly}}$ can be bounded by

$$100\cdot\left(s_\alpha + s_{\mathtt{tt}\text{-}\hat{\Phi}} + s_{\mathsf{arith}}\right) \leq c\cdot\kappa\log T = \log T_{\mathsf{poly}}$$

as well. This completes the proof. □

[31] It is in fact in logtime-uniform TC^0, but here we only need it to be in logspace-uniform NC^1.

6.5.2 Improved Chen–Tell Generator: Proof of Theorem 6.3.1

Now we are ready to prove Theorem 6.3.1 by plugging every polynomial from Theorem 6.5.1 into our modified Shaltiel–Umans generator (Theorem 6.4.1).

Proof of Theorem 6.3.1. We first observe that we can assume $\rho = 1$ without loss of generality. To see how the general case follows from the case that $\rho = 1$, letting $M' = M^\rho$, we can simply define $\mathsf{H}^{\mathsf{ct}}_{n,T,d,M,\kappa,\rho}$ as the set of strings obtained by truncating every string from $\mathsf{H}^{\mathsf{ct}}_{n,T,d,M',\kappa,1}$ to their first M bits. The reconstruction algorithm $\mathsf{R}^{\mathsf{ct}}_{n,T,d,M,\kappa,\rho}$ can then be obtained by slightly modifying $\mathsf{R}^{\mathsf{ct}}_{n,T,d,M',\kappa,1}$.

Let

$$\vec{\ell}_{\mathsf{ct}} = (n, T, d, M, \kappa, 1)$$

be the input parameters from the theorem statement, and c be a sufficiently large universal constant. From the assumption, we have $n \le T, d \le T$, and $c \cdot \log T \le M \le T^{1/c}$. Let

$$C_{\mathsf{TM}} = \mathsf{Circuit}[T, \kappa \cdot \log T, n, n](\mathsf{TM}).$$

The layered-polynomial representation. Let c_0, c_0', β be the universal constants from Theorem 6.5.1. Let h be the *smallest* nice power of 2 such that $h \ge M$, $p := h^{27}$, m be the smallest power of 3 such that $h^m \ge T^{\beta\kappa}$, and $\mathbb{F} = \mathbb{F}_p$. Note that p is also a nice power of 2 and $h \le M^3$.

We will invoke Theorem 6.5.1 with input parameters

$$\vec{\ell}_{\mathsf{poly}} = (\kappa, T, d, n, h, p).$$

Note that from their definitions and our assumption $M \ge c \log T$, we have $\log T \le h < p \le h^{27} \le M^{81} \le T$ (assuming $c \ge 81$ is large enough), meaning that the requirements on the input parameters of Theorem 6.5.1 are satisfied.

We first apply Theorem 6.5.1 with input parameters $\vec{\ell}_{\mathsf{poly}}$ and Turing machine TM to obtain $d' = c_0\kappa \cdot \log^2 T \cdot (d + \kappa^2 \log T)$ polynomials $(P_i)_{i\in[d']} = \left(P_i^{\vec{\ell},\mathsf{TM}}\right)_{i\in[d']}$.

Hitting set H^{ct}. Let $\mathsf{H}^{\mathsf{layer}}$ and $\mathsf{R}^{\mathsf{layer}}$ denote the H and R algorithms from Theorem 6.4.1, respectively.[32] Let $\Delta = c_0 h \log^3(T)$,

$$\vec{\ell}_{\mathsf{layer}} = (p, 3m, M, \Delta)$$

be the input parameters when applying Theorem 6.4.1. We can verify that $p > \Delta^2 (3m)^7 M^9$, *i.e.*, the requirement on the input parameters of Theorem 6.4.1 is satisfied.

We then define $\mathsf{H}^{\mathsf{ct}}_{\vec{\ell}_{\mathsf{ct}}}(\mathsf{TM})$ as the union of $\mathsf{H}^{\mathsf{layer}}_{\vec{\ell}_{\mathsf{layer}}}(P_i)$ for every $i \in [d']$. Next, we analyse the complexity of computing $\mathsf{H}^{\mathsf{ct}}_{\vec{\ell}_{\mathsf{ct}}}(\mathsf{TM})$. First, from Theorem 6.5.1, letting $T_{\mathsf{poly}} = T^{c_0\cdot\kappa}$ and $d_{\mathsf{poly}} = c_0 \cdot (d \log T + \kappa^2 \log^2 T)$, there is a polynomial-time algorithm $\mathbb{A}^{\mathsf{poly}}_{\vec{\ell}}$ that takes $\mathsf{TM} \in \{0,1\}^{\kappa \log T}$

[32] The superscript layer highlights the fact that they are applied to each layer of the polynomial representation of the circuit.

as input, and outputs a description of Turing machine $\mathsf{TM}_{\mathsf{poly}} \in \{0,1\}^{\log T_{\mathsf{poly}}}$ such that for

$$C_{\mathsf{poly}} = \mathsf{Circuit}\big[T_{\mathsf{poly}}, \log T_{\mathsf{poly}}, \log d', |\mathbb{F}|^{3m} \cdot \log|\mathbb{F}|\big](\mathsf{TM}_{\mathsf{poly}})$$

it holds that (1) for every $i \in [d']$ $C_{\mathsf{poly}}(i) = \mathtt{tt}(P_i)$ and (2) C_{poly} has depth d_{poly}.

Second, from Theorem 6.4.1, there is a logspace-uniform circuit family with input parameters $\vec{\ell}_{\mathsf{layer}}$, size $\mathrm{poly}(p^m)$, and depth $\mathrm{poly}(\log p, m, M)$ such that for every $i \in [d']$, it outputs $\mathsf{H}^{\mathsf{layer}}_{\vec{\ell}_{\mathsf{layer}}}(P_i)$ when taking $\mathtt{tt}(P_i)$ as input. Note that $\mathrm{poly}(p^m) \le T^{O(\beta\kappa)}$ and $\mathrm{poly}(\log p, m, M) \le \mathrm{poly}(M)$. Applying Fact 6.2.4 to compose the machines above and enumerating over all $i \in [d']$,[33] we obtain the desired circuit C_{H} (note that c is sufficiently large).

Reconstruction R^{ct}. For every $i \in \{2, \ldots, d'\}$, the reconstruction algorithm R^{ct} attempts to construct a $\mathrm{poly}(p, m, \log(Md'))$-size D-oracle circuit E_i that computes P_i. A formal description of R^{ct} is as follows:

- We start with the circuit $E_1(\vec{x}) = \mathsf{Base}(\vec{\ell}, \mathsf{TM}, \vec{x})$ that computes the polynomial P_1.
- For every $i \in \{2, \ldots, d'\}$:
  1. We first construct a procedure $\widetilde{P}_i$ computing P_i using the D-oracle circuit E^D_{i-1} for P_{i-1} and the downward self-reducibility for P_i. In particular, on input $\vec{x} \in \mathbb{F}^{3m}$, let
  $$\widetilde{P}_i(\vec{x}) := \mathsf{DSR}^{E^D_{i-1}}(\vec{\ell}, \mathsf{TM}, i, \vec{x}).$$
  2. Run $\big(\mathsf{R}^{\mathsf{layer}}\big)^{D,\widetilde{P}_i}_{\vec{\ell}_{\mathsf{layer}}}$ which outputs a D-oracle circuit $\widetilde{E}^D_i$ in $\mathrm{poly}(p, m, M)$ time.
  3. Let $t := c_1 \cdot m \cdot \log p$ for a sufficiently large constant $c_1 > 1$. Take t i.i.d. samples $\vec{x}_1, \ldots, \vec{x}_t$ from $\mathbb{F}^{3m}$. Check that for every $j \in [t]$, $\widetilde{E}^D_i(\vec{x}_j) = \widetilde{P}_i(\vec{x}_j)$. If any condition does not hold, the algorithm outputs $\perp$ and aborts immediately.
  4. Let E_i be a D-oracle circuit constructed as follows:
     (a) Draw $t = \Theta(m \log p)$ i.i.d. samples of random strings $r_1, \ldots, r_t$ used by PCorr. (Recall PCorr is the self-corrector for low-degree polynomials in Theorem 6.4.8.)
     (b) Set $E_i(\vec{x}) = \mathsf{MAJ}_{k\in[t]}\ \mathsf{PCorr}^{\widetilde{E}_i}(p, 3m, \Delta, \vec{x}; r_k)$ for all $\vec{x} \in \mathbb{F}^{3m}$.
- For every $j \in [n]$, output $E^D_{d'}(\mathsf{id}(j), 0^{2m})$.

For ease of notation, for every $i' \in \{2, \ldots, d'\}$, we use $\tau_{i'}$ to denote the randomness used when running the algorithm above with $i = i'$, and we use $\tau_{\le i}$ to denote $\tau_1, \ldots, \tau_i$. Also, if E_i is not constructed by the algorithm (meaning that the algorithm aborts before constructing E_i), we set $E_i = \perp$.

From Theorem 6.4.8, Theorem 6.4.1, and Theorem 6.5.1, the running time of the algorithm above can be bounded by

$$\mathrm{poly}(p, m, h, \log(Md')) \cdot (d' + n) \le \mathrm{poly}(M) \cdot (d' + n) \le \mathrm{poly}(M) \cdot (d + n).$$

[33] Enumerating all $i \in [d']$ only adds a $O(\log d')$ additive overhead in depth and a $O(d')$ multiplicative blowup in size, which are negligible.

The last inequality follows from the fact that $M \geq \log T$ and hence $d' = c_0\kappa \cdot \log^2 T \cdot (d + \kappa^2 \log T) \leq \mathrm{poly}(M) \cdot d$. Now we establish the soundness and completeness of the reconstruction. We show the following claim.

Claim 6.5.4. *Fix $D\colon \{0,1\}^M \to \{0,1\}$. For every $i \in \{2,\ldots,d'\}$, for every fixed $\tau_{\leq i-1}$, if E_{i-1}^D computes P_{i-1} or $i = 2$,[34] then with probability at least $1 - 1/p^m$ over τ_i the following holds:*

- **(Soundness.)** *If $E_i \neq \perp$, then E_i^D computes P_i.*
- **(Completeness.)** *If D $(1/M)$-avoids $\mathsf{H}^{\mathsf{layer}}_{\vec{\ell}_{\mathsf{layer}}}(P_i)$, then E_i^D computes P_i.*

Before establishing the claim, we show that it implies the completeness and soundness of the reconstruction. To see the soundness, note that by induction over all $i \in \{2,\ldots,d'\}$, with probability at least $1 - d'/p^m > 9/10$, it holds that if $E_{d'} \neq \perp$, then $E_{d'}$ computes $P_{d'}$, meaning the reconstruction outputs the correct output $C_{\mathsf{TM}}(1^n)$. To see the completeness, note that an oracle $D\colon \{0,1\}^M \to \{0,1\}$ that $(1/M)$-avoids $\mathsf{H}^{\mathsf{ct}}_{\vec{\ell}}(\mathsf{TM})$ also $(1/M)$-avoids $\mathsf{H}^{\mathsf{layer}}_{\vec{\ell}_{\mathsf{layer}}}(P_i)$ for every $i \in [d']$. Hence, by induction over $i \in \{2,\ldots,d'\}$, with probability at least $1 - d'/p^m > 9/10$, it holds that E_i computes P_i for every $i \in \{2,\ldots,d'\}$. Thus the reconstruction will output $C_{\mathsf{TM}}(1^n)$. The success probability $9/10$ can be amplified to $1-2^{-M}$ by running the reconstruction algorithm $O(M)$ times independently and outputting the answer that occurs most frequently.

Finally, we prove the claim.

Proof of Claim 6.5.4. We first establish the soundness. From the assumption that E_{i-1}^D computes P_{i-1} or $i = 2$ and the downward self-reducibility property of Theorem 6.5.1, it follows that $\widetilde{P}_i$ computes P_i. Therefore, $E_i \neq \perp$ means that $\widetilde{E}_i$ has passed the test in Step 3, meaning that with probability at least $1 - p^{-4m}$ over the randomness in Step 3, it holds that $\widetilde{E}_i$ agrees P_i on at least $3/4$ fraction of inputs from $\mathbb{F}^{3m}$. This then means that with probability at least $1 - p^{-3m}$ over the randomness in Step 4(a), we have E_i^D computes P_i.

The completeness follows immediately from Theorem 6.4.1. (Here $\widetilde{E}_i^D$ already computes P_i with probability at least $1 - 1/p^m$.) ◇

This completes the proof of Theorem 6.3.1. □

[34] Note that $\tau_{\leq i-1}$ determines E_{i-1}.

Chapter 7

Near-Maximum Circuit Lower Bounds and New Algorithms for Range Avoidance

7.1 Introduction

Proving lower bounds against non-uniform computation (i.e., circuit lower bounds) is one of the most important challenges in theoretical computer science. From Shannon's counting argument [Sha49, FM05], we know that almost all n-bit Boolean functions have *near-maximum* ($2^n/n$) circuit complexity.[1] Therefore, the task of proving circuit lower bounds is simply to *pinpoint* one such hard function. More formally, one fundamental question is:

> What is the smallest complexity class that contains a language of exponential ($2^{\Omega(n)}$) circuit complexity?

Compared with super-polynomial lower bounds, exponential lower bounds are interesting in their own right for the following reasons. First, an exponential lower bound would make Shannon's argument *fully constructive*. Second, exponential lower bounds have more applications than super-polynomial lower bounds: For example, if one can show that E has no $2^{o(n)}$-size circuits, then we would obtain polynomial-time derandomisation of BPP [NW94, IW97], while super-polynomial lower bounds such as $\mathsf{EXP} \not\subset \mathsf{P}/_{\mathrm{poly}}$ only imply subexponential time derandomisation of BPP.[2]

Unfortunately, despite its importance, our knowledge about exponential lower bounds is quite limited. Kannan [Kan82] showed that there is a function in $\Sigma_3\mathsf{E} \cap \Pi_3\mathsf{E}$ that requires maximum

[1] All n-input Boolean functions can be computed by a circuit of size $(1 + \frac{3\log n}{n} + O(\frac{1}{n}))2^n/n$ [Lup58, FM05], while most Boolean functions require circuits of size $(1 + \frac{\log n}{n} - O(\frac{1}{n}))2^n/n$ [FM05]. Hence, we say an n-bit Boolean function has *near-maximum* circuit complexity if its circuit complexity is at least $2^n/n$.

[2] $\mathsf{E} = \mathsf{DTIME}[2^{O(n)}]$ denotes *single-exponential* time and $\mathsf{EXP} = \mathsf{DTIME}[2^{n^{O(1)}}]$ denotes *exponential* time; classes such as E^{NP} and $\mathsf{EXP}^{\mathsf{NP}}$ are defined analogously. Exponential time and single-exponential time are basically interchangeable in the context of super-polynomial lower bounds (by a padding argument); the exponential lower bounds proven in this chapter will be stated for single-exponential time classes since this makes our results stronger. Below, $\Sigma_3\mathsf{E}$ and $\Pi_3\mathsf{E}$ denote the exponential-time versions of $\Sigma_3\mathsf{P} = \mathsf{NP}^{\mathsf{NP}^{\mathsf{NP}}}$ and $\Pi_3\mathsf{P} = \mathsf{coNP}^{\mathsf{NP}^{\mathsf{NP}}}$, respectively.

circuit complexity; the complexity of the hard function was later improved to $\Delta_3\mathsf{E} = \mathsf{E}^{\Sigma_2\mathsf{P}}$ by Miltersen, Vinodchandran, and Watanabe [MVW99], via a simple binary search argument. This is **essentially all we know** regarding exponential circuit lower bounds.[3]

We remark that Kannan [Kan82, Theorem 4] claimed that $\Sigma_2\mathsf{E} \cap \Pi_2\mathsf{E}$ requires exponential circuit complexity, but [MVW99] pointed out a gap in Kannan's proof, and suggested that exponential lower bounds for $\Sigma_2\mathsf{E} \cap \Pi_2\mathsf{E}$ were "reopened and considered an open problem." Recently, Vyas and Williams [VW23] emphasised our lack of knowledge regarding the circuit complexity of $\Sigma_2\mathsf{EXP}$, even with respect to *relativising* proof techniques. In particular, the following question has been open for at least 20 years (indeed, if we count from [Kan82], it would be at least 40 years):

Open Problem 7.1.1. *Can we prove that* $\Sigma_2\mathsf{EXP} \not\subset \mathsf{SIZE}[2^{\varepsilon n}]$ *for some absolute constant* $\varepsilon > 0$*, or at least show a relativisation barrier for proving such a lower bound?*

The half-exponential barrier. There is a richer literature regarding super-polynomial lower bounds than exponential lower bounds. Kannan [Kan82] proved that the class $\Sigma_2\mathsf{E} \cap \Pi_2\mathsf{E}$ does not have polynomial-size circuits. Subsequent works proved super-polynomial circuit lower bounds for exponential-time complexity classes such as $\mathsf{ZPEXP}^{\mathsf{NP}}$ [KW98, BCG+96], $\mathsf{S}_2\mathsf{EXP}$ [CCHO05, Cai07], PEXP [Vin05, Aar06], and MAEXP [BFT98, San09].

Unfortunately, all these works fail to prove exponential lower bounds. All of their proofs go through certain *Karp–Lipton* collapses [KL80]; such a proof strategy runs into a so-called "half-exponential barrier", preventing us from getting exponential lower bounds. See Section 7.1.4 for a detailed discussion.

7.1.1 Our Results

New near-maximum circuit lower bounds

In this work, we *overcome* the half-exponential barrier mentioned above and resolve Open Problem 7.1.1 by showing that both $\Sigma_2\mathsf{E}$ and $(\Sigma_2\mathsf{E} \cap \Pi_2\mathsf{E})/_1$ require near-maximum $(2^n/n)$ circuit complexity. Moreover, our proof indeed *relativises*:

Theorem 7.1.2. $\Sigma_2\mathsf{E} \not\subset \mathsf{SIZE}[2^n/n]$ *and* $(\Sigma_2\mathsf{E} \cap \Pi_2\mathsf{E})/_1 \not\subset \mathsf{SIZE}[2^n/n]$*. Moreover, they hold in every relativised world.*

Up to one bit of advice, we finally provide a proof of Kannan's original claim in [Kan82, Theorem 4]. Moreover, with some more work, we extend our lower bounds to the smaller complexity class $\mathsf{S}_2\mathsf{E}/_1$ (see Definition 7.2.1 for a formal definition), again with a relativising proof:

Theorem 7.1.3. $\mathsf{S}_2\mathsf{E}/_1 \not\subset \mathsf{SIZE}[2^n/n]$*. Moreover, this holds in every relativised world.*

[3] We also mention that Hirahara, Lu, and Ren [HLR23] recently proved that for every constant $\varepsilon > 0$, $\mathsf{BPE}^{\mathsf{MCSP}}/_{2^{\varepsilon n}}$ requires near-maximum circuit complexity, where MCSP is the Minimum Circuit Size Problem [KC00]. However, the hard function they constructed requires subexponentially $(2^{\varepsilon n})$ many advice bits to describe.

The symmetric time class $\mathsf{S_2E}$. $\mathsf{S_2E}$ can be seen as a "randomised" version of $\mathsf{E^{NP}}$ since it is sandwiched between $\mathsf{E^{NP}}$ and $\mathsf{ZPE^{NP}}$: it is easy to show that $\mathsf{E^{NP}} \subseteq \mathsf{S_2E}$ [RS98], and it is also known that $\mathsf{S_2E} \subseteq \mathsf{ZPE^{NP}}$ [Cai07]. We also note that under plausible derandomisation assumptions (e.g., $\mathsf{E^{NP}}$ requires $2^{\Omega(n)}$-size SAT-oracle circuits), all three classes simply collapse to $\mathsf{E^{NP}}$ [KvM02].

Hence, our results also imply a near-maximum circuit lower bound for the class $\mathsf{ZPE^{NP}}/_1 \subseteq (\Sigma_2\mathsf{E} \cap \Pi_2\mathsf{E})/_1$. This vastly improves the previous lower bound for $\Delta_3\mathsf{E} = \mathsf{E}^{\Sigma_2\mathsf{P}}$.

Corollary 7.1.4. $\mathsf{ZPE^{NP}}/_1 \not\subset \mathsf{SIZE}[2^n/n]$. *Moreover, this holds in every relativised world.*

New algorithms for the range avoidance problem

Actually, our circuit lower bounds are implied by our new algorithms for solving the range avoidance problem. There is a trivial $\mathsf{FZPP^{NP}}$ algorithm solving Avoid: randomly generate strings $y \in \{0,1\}^{n+1}$ and output the first y that is outside the range of C (note that we need an NP oracle to verify if $y \notin \mathrm{Range}(C)$). Recall that, as demonstrated by Korten [Kor21, Section 3], Avoid captures the complexity of explicit construction problems whose solutions are guaranteed to exist by the probabilistic method (more precisely, the dual weak pigeonhole principle [Kra01b, Jeř04]), in the sense that constructing such objects reduces to Avoid. This includes many important objects in mathematics and theoretical computer science, including Ramsey graphs [Erd59], rigid matrices [Val77, GLW22, GGNS23], two-source extractors [CZ19, Li23], linear codes [GLW22], hard truth tables [Kor21], and strings with maximum time-bounded Kolmogorov complexity (i.e., $\mathrm{K}^{\mathrm{poly}}$-random strings) [RSW22]. Hence, derandomising the trivial $\mathsf{FZPP^{NP}}$ algorithm for Avoid would imply explicit constructions for all these important objects.

Our results: new pseudodeterministic algorithms for Avoid. We show that, *unconditionally*, the trivial $\mathsf{FZPP^{NP}}$ algorithm for Avoid can be made *pseudodeterministic* on infinitely many input lengths. A *pseudodeterministic* algorithm [GG11] is a randomised algorithm that outputs the same *canonical* answer on most computational paths. In particular, we have:

Theorem 7.1.5. *For every constant $d \geq 1$, there is a randomised algorithm $\mathcal{A}$ with an NP oracle such that the following holds for infinitely many integers n. For every circuit $C\colon \{0,1\}^n \to \{0,1\}^{n+1}$ of size at most n^d, there is a string $y_C \in \{0,1\}^n \setminus \mathrm{Range}(C)$ such that $\mathcal{A}(C)$ either outputs y_C or $\perp$, and the probability (over the internal randomness of $\mathcal{A}$) that $\mathcal{A}(C)$ outputs y_C is at least $2/3$. Moreover, this theorem holds in every relativised world.*

As a corollary, for every problem in APEPP, we obtain zero-error pseudodeterministic constructions with an NP oracle and one bit of advice ($\mathsf{FZPP^{NP}}/_1$) that works infinitely often[4]:

Corollary 7.1.6 (Informal)**.** *There are infinitely-often zero-error pseudodeterministic constructions for the following objects with an NP oracle and one bit of advice: Ramsey graphs, rigid matrices, two-source extractors, linear codes, hard truth tables, and $\mathrm{K}^{\mathrm{poly}}$-random strings.*

[4]The one-bit advice encodes whether our algorithm succeeds on a given input length; it is needed since on bad input lengths, our algorithm might not be pseudodeterministic (i.e., there may not be a canonical answer that is outputted with high probability).

Actually, we obtain single-valued $\mathsf{FS_2P}/_1$ algorithms for the explicit construction problems above (see Definition 7.2.2), and the pseudodeterministic $\mathsf{FZPP^{NP}}/_1$ algorithms follow from Cai's theorem that $\mathsf{S_2P} \subseteq \mathsf{ZPP^{NP}}$ [Cai07]. We stated them as pseudodeterministic $\mathsf{FZPP^{NP}}/_1$ algorithms since this notion is better known than the notion of single-valued $\mathsf{FS_2P}/_1$ algorithms.

Theorem 7.1.5 is tantalizingly close to an infinitely-often $\mathsf{FP^{NP}}$ algorithm for Avoid (with the only caveat of being *zero-error* instead of being completely *deterministic*). However, since an $\mathsf{FP^{NP}}$ algorithm for range avoidance would imply near-maximum circuit lower bounds for $\mathsf{E^{NP}}$, we expect that it would require fundamentally new ideas to completely derandomise our algorithm. Previously, Hirahara, Lu, and Ren [HLR23, Theorem 36] presented an infinitely-often pseudodeterministic $\mathsf{FZPP^{NP}}$ algorithm for the range avoidance problem using n^ε bits of advice, for any small constant $\varepsilon > 0$. Our result improves the above in two aspects: first, we reduce the number of advice bits to 1; second, our techniques relativise but their techniques do not.

Lower bounds against non-uniform computation with maximum advice length. Finally, our results also imply lower bounds against non-uniform computation with maximum advice length. We mention this corollary because it is a stronger statement than circuit lower bounds, and similar lower bounds appeared recently in the literature of super-fast derandomisation [CT21b].

Corollary 7.1.7. *For every $\alpha(n) \geq \omega(1)$ and any constant $k \geq 1$, $\mathsf{S_2E}/_1 \not\subset \mathsf{TIME}[2^{kn}]/_{2^n-\alpha(n)}$. The same holds for $\Sigma_2\mathsf{E}$, $(\Sigma_2\mathsf{E} \cap \Pi_2\mathsf{E})/_1$, and $\mathsf{ZPE^{NP}}/_1$ in place of $\mathsf{S_2E}/_1$. Moreover, this holds in every relativised world.*

7.1.2 Intuitions

In the following, we present some high-level intuitions for our new circuit lower bounds.

Perspective: single-valued constructions

A key perspective in this chapter is to view circuit lower bounds (for exponential-time classes) as *single-valued* constructions of hard truth tables. This perspective is folklore; it was also emphasised in recent papers on the range avoidance problem [Kor21, RSW22].

Let $\Pi \subseteq \{0,1\}^\star$ be an *ε-dense* property, i.e., for every integer $N \in \mathbb{N}$, $|\Pi_N| \geq \varepsilon \cdot 2^N$. (In what follows, we use $\Pi_N := \Pi \cap \{0,1\}^N$ to denote the length-N slice of Π.) As a concrete example, let Π_{hard} be the set of hard truth tables, i.e., a string $tt \in \Pi_{\text{hard}}$ if and only if it is the truth table of a function $f\colon \{0,1\}^n \to \{0,1\}$ whose circuit complexity is at least $2^n/n$, where $n := \log N$. (We assume that $n := \log N$ is an integer.) Shannon's argument [Sha49, FM05] shows that Π_{hard} is a $1/2$-dense property. We are interested in the following question:

What is the complexity of *single-valued* constructions for any string in Π_{hard}?

Here, informally speaking, a computation is *single-valued* if each of its computational paths either fails or outputs the *same* value. For example, an NP machine M is a single-valued construction for Π if there is a "canonical" string $y \in \Pi$ such that (1) M outputs y on every accepting computational path; (2) M has at least one accepting computational path. (That is, it is an NPSV construction in the sense of [BLS85, FHOS93, Sel94, HNOS96].) Similarly, a BPP

machine M is a single-valued construction for Π if there is a "canonical" string $y \in \Pi$ such that M outputs y on most (say $\geq 2/3$ fraction of) computational paths. (In other words, single-valued ZPP and BPP constructions are another name for *pseudodeterministic constructions* [GG11].)[5]

Hence, the task of proving circuit lower bounds is equivalent to the task of *defining*, i.e., single-value constructing, a hard function, in the smallest possible complexity class. For example, a single-valued BPP construction (i.e., pseudodeterministic construction) for Π_{hard} is equivalent to the circuit lower bound $\mathsf{BPE} \not\subset \text{i.o.-}\mathsf{SIZE}[2^n/n]$.[6] In this regard, the previous near-maximum circuit lower bound for $\Delta_3\mathsf{E} := \mathsf{E}^{\Sigma_2\mathsf{P}}$ [MVW99] can be summarised in one sentence: The lexicographically first string in Π_{hard} can be constructed in $\Delta_3\mathsf{P} := \mathsf{P}^{\Sigma_2\mathsf{P}}$ (which is necessarily single-valued).

Reduction to Avoid. It was observed in [KKMP21, Kor21] that explicit construction of elements from Π_{hard} is a special case of range avoidance: Let $\mathsf{TT}\colon \{0,1\}^{N-1} \to \{0,1\}^N$ (here $N = 2^n$) be a circuit that maps the description of a $2^n/n$-size circuit into its 2^n-length truth table (by [FM05], this circuit can be encoded by $N-1$ bits). Hence, a single-valued algorithm solving Avoid for TT is equivalent to a single-valued construction for Π_{hard}. This explains how our new range avoidance algorithms imply our new circuit lower bounds (as mentioned in Section 7.1.1).

In the rest of Section 7.1.2, we will only consider the special case of Avoid where the input circuit for range avoidance is a P-uniform circuit family. Specifically, let $\{C_n\colon \{0,1\}^n \to \{0,1\}^{2n}\}_{n\in\mathbb{N}}$ be a P-uniform family of circuits, where $|C_n| \leq \text{poly}(n)$.[7] Our goal is to find an algorithm A such that for infinitely many n, $A(1^n) \in \{0,1\}^{2n} \setminus \text{Range}(C_n)$; see Section 7.5.3 and Section 7.5.4 for how to turn this into an algorithm that works for arbitrary input circuit with a single bit of stretch. Also, since from now on we will not talk about truth tables anymore, we will use n instead of N to denote the input length of Avoid instances.

The iterative win-win paradigm of [CLO+23]

In a recent work, Chen, Lu, Oliveira, Ren, and Santhanam [CLO+23] introduced the *iterative win-win* paradigm for explicit constructions, and used that to obtain a polynomial-time pseudodeterministic construction of primes that works infinitely often. Since our construction algorithm closely follows their paradigm, it is instructive to take a detour and give a high-level overview of how the construction from [CLO+23] works.[8]

[5] Note that the trivial construction algorithms are not single-valued in general. For example, a trivial $\Sigma_2\mathsf{P} = \mathsf{NP}^{\mathsf{NP}}$ construction algorithm for Π_{hard} is to guess a hard truth table tt and use the NP oracle to verify that tt does not have size-$N/\log N$ circuits; however, different accepting computational paths of this computation would output different hard truth tables. Similarly, a trivial BPP construction algorithm for every dense property Π is to output a random string, but there is no *canonical* answer that is outputted with high probability. In other words, these construction algorithms do not *define* anything; instead, a single-valued construction algorithm should *define* some particular string in Π.

[6] To see this, note that (1) $\mathsf{BPE} \not\subset \text{i.o.-}\mathsf{SIZE}[2^n/n]$ implies a simple single-valued BPP construction for Π_{hard}: given $N = 2^n$, output the truth table of L_n (L restricted to n-bit inputs), where $L \in \mathsf{BPE}$ is the hard language not in $\mathsf{SIZE}[2^n/n]$; and (2) assuming a single-valued BPP construction A for Π_{hard}, one can define a hard language L such that the truth table of L_n is the output of $A(1^{2^n})$, and observe that $L \in \mathsf{BPE}$.

[7] We assume that C_n stretches n bits to $2n$ bits instead of $n+1$ bits for simplicity; Korten [Kor21] showed that there is a P^{NP} reduction from the range avoidance problem with stretch $n+1$ to the range avoidance problem with stretch $2n$.

[8] Indeed, for every $1/\text{poly}(n)$-dense property $\Pi \in \mathsf{P}$, they obtained a polynomial-time algorithm A such that for infinitely many $n \in \mathbb{N}$, there exists $y_n \in \Pi_n$ such that $A(1^n)$ outputs y_n with probability at least $2/3$. By [AKS04]

In this paradigm, for a (starting) input length n_0 and some $t = O(\log n_0)$, we will consider an increasing sequence of input lengths $n_0, n_1, \ldots, n_t$ (jumping ahead, we will set $n_{i+1} = n_i^\beta$ for a large constant β), and show that our construction algorithm succeeds on at least one of the input lengths. By varying n_0, we can construct infinitely many such sequences of input lengths that are pairwise disjoint, and therefore our algorithm succeeds on infinitely many input lengths.

In more detail, fixing a sequence of input lengths $n_0, n_1, \ldots, n_t$ and letting Π be an ε-dense property, for each $i \in \{0, 1, \ldots, t\}$, we specify a (deterministic) algorithm ALG_i that takes 1^{n_i} as input and aims to construct an explicit element from Π_{n_i}. We let ALG_0 be the simple brute-force algorithm that enumerates all length-n_0 strings and finds the lexicographically first string in Π_{n_0}; it is easy to see that ALG_0 runs in $T_0 := 2^{O(n_0)}$ time.

The win-or-improve mechanism. The core of [CLO+23] is a novel *win-or-improve mechanism*, which is described by a (randomised) algorithm R. Roughly speaking, for input lengths n_i and n_{i+1}, $R(1^{n_i})$ attempts to *simulate* ALG_i *faster by using the oracle* $\Pi_{n_{i+1}}$ (hence it runs in $\mathrm{poly}(n_{i+1})$ time). The crucial property is the following win-win argument:

(**Win**) Either $R(1^{n_i})$ outputs $\mathsf{ALG}_i(1^{n_i})$ with probability at least $2/3$ over its internal randomness,

(**Improve**) or, from the failure of $R(1^{n_i})$, we can construct an algorithm ALG_{i+1} that outputs an explicit element from $\Pi_{n_{i+1}}$ and runs in $T_{i+1} = \mathrm{poly}(T_i)$ time.

We call the above (Win-or-Improve), since either we have a pseudodeterministic algorithm $R(1^{n_i})$ that constructs an explicit element from Π_{n_i} in $\mathrm{poly}(n_{i+1}) \leq \mathrm{poly}(n_i)$ time (since it simulates ALG_i), or we have an *improved* algorithm ALG_{i+1} at the input length n_{i+1} (for example, on input length n_1, the running time of ALG_1 is $2^{O\left(n_1^{1/\beta}\right)} \ll 2^{O(n_1)}$). The (Win-or-Improve) part in [CLO+23] is implemented via the Chen–Tell targeted hitting set generator [CT21a] (we omit the details here). Jumping ahead, in this chapter, we will implement a similar mechanism using Jeřábek and Korten's P^{NP} reduction from the range avoidance problem to constructing hard truth tables [Jeř04, Kor21].

Getting polynomial time. We briefly explain why (Win-or-Improve) implies a *polynomial-time* construction algorithm. Let α be an absolute constant such that we always have $T_{i+1} \leq T_i^\alpha$; we now set $\beta := 2\alpha$. Recall that $n_i = n_{i-1}^\beta$ for every i. The crucial observation is the following:

> Although T_0 is much larger than n_0, the sequence $\{T_i\}$ grows slower than $\{n_i\}$.

Indeed, a simple calculation shows that when $t = O(\log n_0)$, we will have $T_t \leq \mathrm{poly}(n_t)$; see [CLO+23, Section 1.3.1].

For each $0 \leq i < t$, if $R(1^{n_i})$ successfully simulates ALG_i, then we obtain an algorithm for input length n_i running in $\mathrm{poly}(n_{i+1}) \leq \mathrm{poly}(n_i)$ time. Otherwise, we have an algorithm ALG_{i+1} running in T_{i+1} time on input length n_{i+1}. Eventually, we will hit t such that $T_t \leq \mathrm{poly}(n_t)$, in which case ALG_t itself gives a polynomial-time construction on input length n_t. Therefore, we obtain a polynomial-time algorithm on at least one of the input lengths $n_0, n_1, \ldots, n_t$.

---

and the prime number theorem, the set of n-bit primes is such a property.

Algorithms for range-avoidance via Jeřábek–Korten reduction

Now we are ready to describe our new algorithms for Avoid. Roughly speaking, our new algorithm makes use of the iterative win-win argument introduced above, together with an easy-witness style argument [IKW02] and the Jeřábek–Korten reduction [Jeř04, Kor21].[9] In the following, we introduce the latter two ingredients and show how to chain them together via the iterative win-win argument.

An easy-witness style argument. Let BF be the $2^{O(n)}$-time brute-force algorithm outputting the lexicographically first non-output of C_n. Our first idea is to consider its *computational history*, a unique $2^{O(n)}$-length string h_{BF} (that can be computed in $2^{O(n)}$ time), and *branch on whether h_{BF} has a small circuit or not*. Suppose h_{BF} admits a, say, n^{α}-size circuit for some large α, then we apply an *easy-witness-style* argument [IKW02] to simulate BF by a single-valued $\mathsf{F}\Sigma_2\mathsf{P}$ algorithm running in $\mathrm{poly}(n^{\alpha}) = \mathrm{poly}(n)$ time (see Section 7.1.3). Hence, we obtained the desired algorithm when h_{BF} is easy.

However, it is less clear how to deal with the other case (when h_{BF} is hard) directly. The crucial observation is that we have gained the following ability: we can generate a string $h_{\mathsf{BF}} \in \{0,1\}^{2^{O(n)}}$ that has circuit complexity at least n^{α}, in only $2^{O(n)}$ time.

The Jeřábek–Korten reduction. We will apply the Jeřábek–Korten reduction to make use of the "gain" above. So it is worth taking a detour to review the main results of [Jeř04, Kor21]. Roughly speaking, this reduction gives **an algorithm that uses a hard truth table f to solve a derandomisation task: finding a non-output of the given circuit (that has more output bits than input bits).**[10]

Formally, the Jeřábek–Korten reduction is a P^{NP}-computable algorithm $\mathsf{Jeřábek–Korten}(C, f)$ that takes as inputs a circuit $C\colon \{0,1\}^n \to \{0,1\}^{2n}$ and a string $f \in \{0,1\}^T$ (think of $n \ll T$), and outputs a string $y \in \{0,1\}^{2n}$. The guarantee is that if the circuit complexity of f is sufficiently larger than the size of C, then the output y is not in the range of C.

This fits perfectly with our "gain" above: for $\beta \ll \alpha$ and $m = n^{\beta}$, $\mathsf{Jeřábek–Korten}(C_m, h_{\mathsf{BF}})$ solves Avoid for C_m since the circuit complexity of h_{BF}, n^{α}, is sufficiently larger than the size of C_m. Moreover, $\mathsf{Jeřábek–Korten}(C_m, h_{\mathsf{BF}})$ runs in only $2^{O(n)}$ time, which is much less than the brute-force running time $2^{O(m)}$. Therefore, we obtain an improved algorithm for Avoid on input length m.

The iterative win-win argument. What we described above is essentially the first stage of an *win-or-improve mechanism* similar to that from Section 7.1.2. Therefore, we only need to iterate the argument above to obtain a polynomial-time algorithm.

[9] Korten's result was inspired by Jeřábek [Jeř04], who proved that the dual weak pigeonhole principle is equivalent to the statement asserting the existence of Boolean functions with exponential circuit complexity in a certain fragment of Bounded Arithmetic.

[10] This is very similar to the classical hardness-vs-randomness connection [NW94, IW97], which can be understood as an algorithm that uses a hard truth table f (i.e., a truth table without small circuits) to solve another derandomisation task: estimating the acceptance probability of the given circuit. This explains why one may want to use the Jeřábek–Korten reduction to replace the Chen–Tell targeted generator construction [CT21a] from [CLO+23], as they are both hardness-vs-randomness connections.

For this purpose, we need to consider the computational history of not only BF, but also algorithms of the form $\textsf{Jeřábek–Korten}(C, f)$.[11] For any circuit C and "hard" truth table f, there is a *unique* "computational history" h of $\textsf{Jeřábek–Korten}(C, f)$, and the length of h is upper bounded by $\mathrm{poly}(|f|)$. We are able to prove the following statement akin to the *easy witness lemma* [IKW02]: if h admits a size-s circuit (think of $s \ll T$), then $\textsf{Jeřábek–Korten}(C, f)$ can be simulated by a single-valued $\mathsf{F\Sigma_2P}$ algorithm in time $\mathrm{poly}(s)$; see Section 7.1.3 for details on this argument.[12]

Now, following the iterative win-win paradigm of [CLO+23], for a (starting) input length n_0 and some $t = O(\log n_0)$, we consider an increasing sequence of input lengths $n_0, n_1, \ldots, n_t$, and show that our algorithm A succeeds on at least one of the input lengths (i.e., $A(1^{n_i}) \in \{0,1\}^{2n_i} \setminus \mathrm{Range}(C_{n_i})$ for some $i \in \{0, 1, \ldots, t\}$). For each $i \in \{0, 1, \ldots, t\}$, we specify an algorithm ALG_i of the form $\textsf{Jeřábek–Korten}(C_{n_i}, -)$ that aims to solve Avoid for C_{n_i}; in other words, we specify a string $f_i \in \{0,1\}^{T_i}$ for some T_i and let $\mathsf{ALG}_i := \textsf{Jeřábek–Korten}(C_{n_i}, f_i)$.

The algorithm ALG_0 is simply the brute force algorithm BF at input length n_0. (A convenient observation is that we can specify an exponentially long string $f_0 \in \{0,1\}^{2^{O(n_0)}}$ so that $\textsf{Jeřábek–Korten}(C_{n_0}, f_0)$ is equivalent to $\mathsf{BF} = \mathsf{ALG}_0$; see Fact 7.3.4.) For each $0 \le i < t$, to specify ALG_{i+1}, let f_{i+1} denote the history of the algorithm ALG_i, and consider the following win-or-improve mechanism.

(**Win**) If f_{i+1} admits an n_i^α-size circuit (for some large constant α), by our easy-witness argument, we can simulate ALG_i by a $\mathrm{poly}(n_i)$-time single-valued $\mathsf{F\Sigma_2P}$ algorithm.

(**Improve**) Otherwise f_{i+1} has circuit complexity at least n_i^α, we plug it into the Jeřábek–Korten reduction to solve Avoid for $C_{n_{i+1}}$. That is, we take $\mathsf{ALG}_{i+1} := \textsf{Jeřábek–Korten}(C_{n_{i+1}}, f_{i+1})$ as our new algorithm on input length n_{i+1}.

Let $T_i = |f_i|$, then $T_{i+1} \le \mathrm{poly}(T_i)$. By setting $n_{i+1} = n_i^\beta$ for a sufficiently large β, a similar analysis as [CLO+23] shows that for some $t = O(\log n_0)$ we would have $T_t \le \mathrm{poly}(n_t)$, meaning that ALG_t would be a $\mathrm{poly}(n_t)$-time $\mathsf{FP^{NP}}$ algorithm (thus also a single-valued $\mathsf{F\Sigma_2P}$ algorithm) solving Avoid for C_{n_t}. Putting everything together, we obtain a polynomial-time single-valued $\mathsf{F\Sigma_2P}$ algorithm that solves Avoid for at least one of the C_{n_i}.

The hardness condenser perspective. Below, we present another perspective on the construction above, which may help the reader understand it better. In the following, we fix $C_n \colon \{0,1\}^n \to \{0,1\}^{2n}$ to be the truth table generator $\mathsf{TT}_{n,2n}$ that maps an n-bit description of a $\log(2n)$-input circuit into its length-$2n$ truth table. Hence, instead of solving Avoid in general, our goal here is simply *constructing hard truth tables* (or equivalently, proving circuit lower bounds).

[11] Actually, we need to consider all algorithms ALG_i defined below and prove the properties of computational history for these algorithms. It turns out that all of ALG_i are of the form $\textsf{Jeřábek–Korten}(C, f)$ (including ALG_0), so in what follows we only consider the computational history of $\textsf{Jeřábek–Korten}(C, f)$.

[12] With an "encoded" version of history and more effort, we are able to simulate $\textsf{Jeřábek–Korten}(C, f)$ by a single-valued $\mathsf{FS_2P}$ algorithm in time $\mathrm{poly}(s)$, and that is how our $\mathsf{S_2E}$ lower bound is proved; see Section 7.1.3 for details.

We note that $\mathsf{Jeřábek–Korten}(\mathsf{TT}_{n,2n}, f)$ can then be interpreted as a *hardness condenser* [BS06]:[13] Given a truth table $f \in \{0,1\}^T$ whose circuit complexity is sufficiently larger than n, it outputs a length-$2n$ truth table that is maximally hard (i.e., without $n/\log n$-size circuits). The win-or-improve mechanism can be interpreted as an iterative application of this hardness condenser.

At the stage i, we consider the algorithm $\mathsf{ALG}_i \coloneqq \mathsf{Jeřábek–Korten}(\mathsf{TT}_{n_i,2n_i}, f_i)$, which runs in $T_i \approx |f_i|$ time and creates (roughly) n_i bits of hardness. (That is, the circuit complexity of the output of ALG_i is roughly n_i.) In the (**Win**) case above, ALG_i admits an n_i^α-size history f_{i+1} (with length approximately $|f_i|$) and can therefore be simulated in $\mathsf{F\Sigma_2P}$. The magic is that in the (**Improve**) case, we actually have access to *much more hardness than* n_i: the history string f_{i+1} has $n_i^\alpha \gg n_i$ bits of hardness. So we can *distill* these hardness by applying the condenser to f_{i+1} to obtain a maximally hard truth tables of length $2n_{i+1} = 2n_i^\beta$, establish the next algorithm $\mathsf{ALG}_{i+1} \coloneqq \mathsf{Jeřábek–Korten}(\mathsf{TT}_{n_{i+1},2n_{i+1}}, f_{i+1})$, and keep iterating.

Observe that the string f_{i+1} above has $n_i^\alpha > n_i^\beta = n_{i+1}$ bits of hardness. Since $|f_{i+1}| \approx |f_i|$ and $n_{i+1} = n_i^\beta$, the process above creates *harder and harder* strings, until $|f_{i+1}| \le n_{i+1} \le n_i^\alpha$, so the (**Win**) case must happen at some point.

7.1.3 Proof Overview

In this section, we elaborate on the computational history of Jeřábek–Korten and how the easy-witness-style argument gives us $\mathsf{F\Sigma_2P}$ and $\mathsf{FS_2P}$ algorithms.

The Jeřábek–Korten reduction

We first review the key concepts and results from [Kor21] that are needed. Given a circuit $C\colon \{0,1\}^n \to \{0,1\}^{2n}$ and a parameter $T \ge 2n$, we can builds another circuit $\mathsf{GGM}_T[C]$ stretching n bits to T bits as follows:[14]

- On input $x \in \{0,1\}^n$, we set $v_{0,0} = x$. For simplicity, we assume that $T/n = 2^k$ for some $k \in \mathbb{N}$. We build a full binary tree with $k+1$ layers; see Figure 7.1 for an example with $k = 3$.
- For every $i \in \{0,1,\dots,k-1\}$ and $j \in \{0,1,\dots,2^i-1\}$, we set $v_{i+1,2j}$ and $v_{i+1,2j+1}$ to be the first n bits and the last n bits of $C(v_{i,j})$, respectively.
- The output of $\mathsf{GGM}_T[C](x)$ is defined to be the concatenation of $v_{k,0}, v_{k,1}, \dots, v_{k,2^k-1}$.

The following two properties of $\mathsf{GGM}_T[C]$ are established in [Kor21], which will be useful for us:

1. Given $i \in [T], C$ and $x \in \{0,1\}^n$, by traversing the tree from the root towards the leaf with the i-th bit, one can compute the i-th bit of $\mathsf{GGM}_T[C](x)$ in $\mathrm{poly}(\mathsf{SIZE}(C), \log T)$ time. Consequently, for every x, $\mathsf{GGM}_T[C](x)$ has circuit complexity $\le \mathrm{poly}(\mathsf{SIZE}(C), \log T)$.

[13] A hardness condenser takes a long truth table f with certain hardness and outputs a shorter truth table with similar hardness.

[14] We use the name GGM because the construction is similar to the pseudorandom function generator of Goldreich, Goldwasser, and Micali [GGM86].

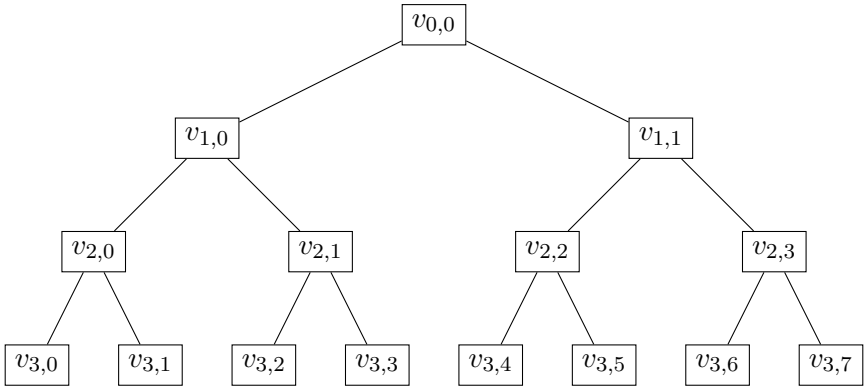


Figure 7.1: An illustration of the GGM Tree, in which, for instance, it holds that $(v_{3,4}, v_{3,5}) = C(v_{2,2})$.

2. There is a P^{NP} algorithm $\mathsf{Jeřábek\text{–}Korten}(C, f)$ that takes $f \in \{0,1\}^T \setminus \mathrm{Range}(\mathsf{GGM}_T[C])$ as input and outputs a string $u \in \{0,1\}^{2n} \setminus \mathrm{Range}(C)$. Note that this is a reduction from solving Avoid for C to solving Avoid for $\mathsf{GGM}_T[C]$.

In particular, letting f be a truth table whose circuit complexity is sufficiently larger than $\mathsf{SIZE}(C)$. By the first property above, f is not in the range of $\mathsf{GGM}_T[C]$, and therefore $\mathsf{Jeřábek\text{–}Korten}(C, f)$ solves Avoid for C. This confirms our description of Jeřábek–Korten in Section 7.1.1.

Computational history of Jeřábek–Korten and an easy-witness argument for $\mathsf{F\Sigma_2P}$ algorithms

The algorithm $\mathsf{Jeřábek\text{–}Korten}(C, f)$ works as follows: we first view f as the labels of the last layer of the binary tree, and try to reconstruct the whole binary tree, layer by layer (start from the bottom layer to the top layer, within each layer, start from the rightmost node to the leftmost one), by filling the labels of the intermediate nodes. To fill $v_{i,j}$, we use an NP oracle to find the lexicographically first string $u \in \{0,1\}^n$ such that $C(u) = v_{i+1,2j} \circ v_{i+1,2j+1}$, and set $v_{i,j} = u$. If no such u exists, the algorithm stops and report $v_{i+1,2j} \circ v_{i+1,2j+1}$ as the solution to Avoid for C. Observe that this reconstruction procedure must stop somewhere, since if it successfully reproduces all the labels in the binary tree, we would have $f = \mathsf{GGM}_T[C](v_{0,0}) \in \mathrm{Range}(\mathsf{GGM}_T[C])$, contradicting the assumption. See Lemma 7.3.3 for details.

The computational history of Jeřábek–Korten. The algorithm described above induces a natural description of the computational history of Jeřábek–Korten, denoted as $\mathsf{History}(C, f)$, as follows: the index $(i_\star, j_\star)$ when the algorithm stops (i.e., the algorithm fails to fill in $v_{i_\star,j_\star}$) concatenated with the labels of all the nodes generated by $\mathsf{Jeřábek\text{–}Korten}(C, f)$ (for the intermediate nodes with no label assigned, we set their labels to a special symbol $\perp$); see Figure 7.2 for an illustration. The length of this history is at most $5T$, and for convenience, we pad additional zeros at its end so that its length is exactly $5T$.

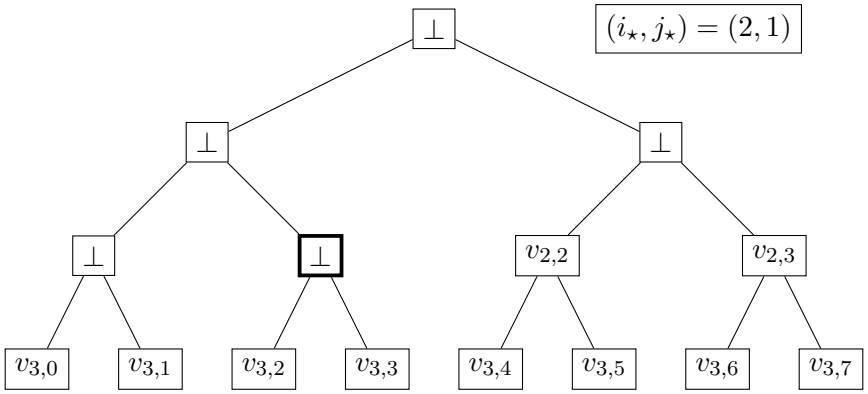


Figure 7.2: An illustration of the history of Jeřábek–Korten(C, f). Here we have $\mathsf{History}(C, f) = (2, 1) \circ \perp\perp\perp\perp\perp \circ v_{2,2} \circ v_{2,3} \circ v_{3,0} \circ \ldots \circ v_{3,7}$ and Jeřábek–Korten$(C, f) = v_{3,2} \circ v_{3,3}$.

A local characterisation of History(C, f). The crucial observation we make on $\mathsf{History}(C, f)$ is that it admits a local characterisation in the following sense: there is a family of local constraints $\{\psi_x\}_{x \in \{0,1\}^{\mathrm{poly}(n)}}$, where each $\psi_x \colon \{0,1\}^{5T} \times \{0,1\}^T \to \{0,1\}$ reads only $\mathrm{poly}(n)$ many bits of its input (we think about it as a local constraint since usually $n \ll T$), such that for fixed f, $\mathsf{History}(C, f) \circ f$ is the unique string making all the ψ_x outputting 1.

The constraints are follows: (1) for every leaf node $v_{k,i}$, its content is consistent with the corresponding block in f; (2) all labels at or before node $(i_\star, j_\star)$ are $\perp$;[15] (3) for every $z \in \{0,1\}^n$, $C(z) \neq v_{i_\star+1,2j_\star} \circ v_{i_\star+1,2j_\star+1}$ (meaning the algorithm fails at $v_{i_\star,j_\star}$); (4) for every (i, j) after $(i_\star, j_\star)$, $C(v_{i,j}) = v_{i+1,2j} \circ v_{i+1,2j+1}$ ($v_{i,j}$ is the correct label); (5) for every (i, j) after $(i_\star, j_\star)$ and for every $v' < v_{i,j}$, $C(v') \neq v_{i+1,2j} \circ v_{i+1,2j+1}$ ($v_{i,j}$ is the lexicographically first correct label). It is clear that each of these constraints above only reads $\mathrm{poly}(n)$ many bits from the input, and a careful examination shows they precisely **define** the string $\mathsf{History}(C, f)$.

A more intuitive way to look at these local constraints is to treat them as a $\mathrm{poly}(n)$-time oracle algorithm V_{History} that takes a string $x \in \mathrm{poly}(n)$ as input and two strings $h \in \{0,1\}^{5T}$ and $f \in \{0,1\}^T$ as oracles, and we simply let $V^{h,f}_{\mathsf{History}}(x) = \psi_x(h \circ f)$. Since the constraints above are all very simple and only read $\mathrm{poly}(n)$ bits of $h \circ f$, V_{History} runs in $\mathrm{poly}(n)$ time. In some sense, V_{History} is a local Π_1 verifier: it is local in the sense that it only queries $\mathrm{poly}(n)$ bits from its oracles, and it is Π_1 since it needs a universal quantifier over $x \in \{0,1\}^{\mathrm{poly}(n)}$ to perform all the checks.

FΣ_2P algorithms. Before we proceed, we give a formal definition of a single-valued FΣ_2P algorithm A. Here A is implemented by an algorithm V_A taking an input x and two $\mathrm{poly}(|x|)$-length witnesses π_1 and π_2. We say $A(x)$ outputs a string $z \in \{0,1\}^\ell$ (we assume $\ell = \ell(x)$ can be computed in polynomial time from x) if z is the *unique* length-ℓ string such that the following hold:

- there exists π_1 such that for every π_2, $V_{\mathsf{History}}(x, \pi_1, \pi_2, z) = 1$.[16]

[15] We say that (i, j) is before (after) $(i_\star, j_\star)$ if the pair (i, j) is lexicographically smaller (greater) than $(i_\star, j_\star)$.

[16] Note that our definition here is different from the formal definition we used in Definition 7.2.2. But from this definition, it is easier to see why FΣ_2P algorithms for constructing hard truth tables imply circuit lower bounds for Σ_2E.

We can view V_{History} as a verifier that checks whether z is the desired output using another universal quantifier: given a proof π_1 and a string $z \in \{0,1\}^\ell$. A accepts z if and only if *for every* π_2, $V_{\mathsf{History}}(x, \pi_1, \pi_2, z) = 1$. That is, A can perform exponentially many checks on π_1 and z, each taking $\mathrm{poly}(|x|)$ time.

The easy-witness argument. Now we are ready to elaborate on the easy-witness argument mentioned in Section 7.1.1. Recall that at stage i, we have $\mathsf{ALG}_i = \mathsf{Jeřábek\text{–}Korten}(C_{n_i}, f_i)$ and $f_{i+1} = \mathsf{History}(C_{n_i}, f_i)$ (the history of ALG_i). Assuming that f_{i+1} admits a $\mathrm{poly}(n_i)$-size circuit, we want to show that $\mathsf{Jeřábek\text{–}Korten}(C_{n_i}, f_i)$ can be simulated by a $\mathrm{poly}(n_i)$-time single-valued $\mathsf{F\Sigma_2P}$ algorithm.

Observe that for every $t \in [i+1]$, f_{t-1} is a substring of f_t since $f_t = \mathsf{History}(C_{n_{t-1}}, f_{t-1})$. Therefore, f_{i+1} admitting a $\mathrm{poly}(n_i)$-size circuit implies that all f_t admit $\mathrm{poly}(n_i)$-size circuits for $t \in [i]$. We can then implement A as follows: the proof π_1 is a $\mathrm{poly}(n_i)$-size circuit C_{i+1} supposed to compute f_{i+1}, from which one can obtain in polynomial time a sequence of circuits $C_1, \dots, C_i$ that are supposed to compute $f_1, \dots, f_i$, respectively. (Also, from Fact 7.3.4, one can easily construct a $\mathrm{poly}(n_0)$-size circuit C_0 computing f_0.) Next, for every $t \in \{0, 1, \dots, i\}$, A checks whether $\mathtt{tt}(C_{t+1}) \circ \mathtt{tt}(C_t)$ satisfies all the local constraints ψ_x's from the characterisation of $\mathsf{History}(C_{n_t}, f_t)$. In other words, A checks whether $V_{\mathsf{History}}^{C_{t+1}, C_t}(x) = 1$ for all $x \in \{0,1\}^{\mathrm{poly}(n_t)}$.

The crucial observation is that since all the C_t have size $\mathrm{poly}(n_i)$, each check above can be implemented in $\mathrm{poly}(n_i)$ time as they only read at most $\mathrm{poly}(n_i)$ bits from their input, despite that $\mathtt{tt}(C_{t+1}) \circ \mathtt{tt}(C_t)$ itself can be much longer than $\mathrm{poly}(n_i)$. Assuming that all the checks of A above are passed, by induction we know that $f_{t+1} = \mathsf{History}(C_{n_t}, f_t)$ for every $t \in \{0, 1, \dots, i\}$. Finally, A checks whether z corresponds to the answer described in $\mathtt{tt}(C_{i+1}) = f_{i+1}$.

Selectors and an easy-witness argument for $\mathsf{FS_2P}$ algorithms

Finally, we discuss how to implement the easy-witness argument above with a single-valued $\mathsf{FS_2P}$ algorithm. It is known that any single-valued $\mathsf{FS_2BPP}$ algorithm can be converted into an equivalent single-valued $\mathsf{FS_2P}$ algorithm outputting the same string [Can96, RS98] (see also the proof of Theorem 7.5.7 for a self-contained argument). Therefore, in the following, we aim to give a single-valued $\mathsf{FS_2BPP}$ algorithm for solving range avoidance, which is easier to achieve.

$\mathsf{FS_2BPP}$ algorithms and randomised selectors. Before we proceed, we give a formal definition of a single-valued $\mathsf{FS_2BPP}$ algorithm A. We implement A by a randomised algorithm V_A that takes an input x and two $\mathrm{poly}(|x|)$-length witnesses π_1 and π_2.[17] We say that $A(x)$ outputs a string $z \in \{0,1\}^\ell$ (we assume $\ell = \ell(x)$ can be computed in polynomial time from x) if the following hold:

- there exists a string h such that for every π, both $V_A(x, h, \pi)$ and $V_A(x, \pi, h)$ output z with probability at least $2/3$. (Note that such z must be unique if it exists.)

Actually, our algorithm A will be implemented as a randomised *selector*: given two potential proofs π_1 and π_2, it first selects the correct one and then outputs the string z induced by the

[17] $\mathsf{FS_2P}$ algorithms are the special case of $\mathsf{FS_2BPP}$ algorithms where the algorithm V_A is *deterministic*.

correct proof.[18]

Recap. Revising the algorithm in Section 7.1.2, our goal now is to give an $\mathsf{FS_2BPP}$ simulation of $\mathsf{Jeřábek\text{–}Korten}(C_{n_i}, f_i)$, assuming that $\mathsf{History}(C_{n_i}, f_i)$ admits a small circuit. Similar to the local Π_1 verifier used in the case of $\mathsf{F\Sigma_2P}$ algorithms, now we consider a local randomised selector V_{select} which takes oracles $\pi_1, \pi_2 \in \{0,1\}^{5T}$ and $f \in \{0,1\}^T$ such that if exactly one of the π_1 and π_2 is $\mathsf{History}(C, f)$, V_{select} outputs its index with high probability.

Assuming that $f_{i+1} = \mathsf{History}(C_{n_i}, f_i)$ admits a small circuit, one can similarly turn V_{select} into a single-valued $\mathsf{FS_2BPP}$ algorithms A computing $\mathsf{Jeřábek\text{–}Korten}(C_{n_i}, f_i)$: treat two proofs π_1 and π_2 as two small circuits C and D both supposed to compute f_{i+1}, from C and D we can obtain a sequence of circuits $\{C_t\}$ and $\{D_t\}$ supposed to compute the f_t for $t \in [i]$. Then we can use the selector V_{select} to decide for each $t \in [i+1]$ which of the C_t and D_t is the correct circuit for f_t. Finally, we output the answer encoded in the selected circuit for f_{i+1}; see the proof of Theorem 7.5.7 for details.[19]

Observation: it suffices to find the first differing node label. Ignore the $(i_\star, j_\star)$ part of the history for now. Let $\{v^1_{i,j}\}$ and $\{v^2_{i,j}\}$ be the node labels encoded in π_1 and π_2, respectively. We also assume that exactly one of them corresponds to the correct node labels in $\mathsf{History}(C, f)$. The crucial observation here is that, since the correct node labels are generated by a deterministic procedure *node by node* (from bottom to top and from rightmost to leftmost), it is possible to tell which of the $\{v^1_{i,j}\}$ and $\{v^2_{i,j}\}$ is correct given the largest (i', j') such that $v^1_{i',j'} \neq v^2_{i',j'}$. (Note that since all (i, j) are processed by $\mathsf{Jeřábek\text{–}Korten}(C, f)$ in reverse lexicographic order, this (i', j') corresponds to the first node label that the wrong process differs from the correct process, so we call this the first differing point.)

In more detail, assuming we know this (i', j'), we proceed by discussing several cases. First of all, if (i', j') corresponds to a leaf, then one can query f to figure out which of $v^1_{i',j'}$ and $v^2_{i',j'}$ is consistent with the corresponding block in f. Now we can assume (i', j') corresponds to an intermediate node. Since (i', j') is the first differing point, we know that $v^1_{i'+1,2j'} \circ v^1_{i'+1,2j'+1} = v^2_{i'+1,2j'} \circ v^2_{i'+1,2j'+1}$ (we let this string to be α for convenience). By the definition of $\mathsf{History}(C, f)$, it follows that the correct $v_{i',j'}$ should be uniquely determined by α, which means the selector only needs to read α, $v^1_{i',j'}$, and $v^2_{i',j'}$, and can then be implemented by a somewhat tedious case analysis (so it is local). We refer readers to the proof of Lemma 7.5.5 for the details and only highlight the most illuminating case here: if both of $v^1_{i',j'}$ and $v^2_{i',j'}$ are good (we say a string γ is good, if $\gamma \neq \perp$ and $C(\gamma) = \alpha$), we select the lexicographically smaller one. To handle the $(i_\star, j_\star)$ part, one needs some additional case analysis. We omit the details here and refer the reader to the proof of Lemma 7.5.5.

The takeaway here is that if we can find the first differing label (i', j'), then we can construct the selector V_{select} and hence the desired single-valued $\mathsf{FS_2BPP}$ algorithm.

[18] If both proofs are correct or neither proofs are correct, it can select an arbitrary one. The condition only applies when exactly one of the proofs is correct.

[19] However, for the reasons to be explained below, we will actually work with the encoded history instead of the history, which entails a lot of technical challenges in the actual proof.

Encoded history. However, the above assumes the knowledge of (i', j'). In general, if one is only given oracle access to $\{v^1_{i,j}\}$ and $\{v^2_{i,j}\}$, there is no $\mathrm{poly}(n)$-time oracle algorithm computing (i', j') because there might be exponentially many nodes. To resolve this issue, we will encode $\{v^1_{i,j}\}$ and $\{v^2_{i,j}\}$ via Reed–Muller codes.

Formally, recall that $\mathsf{History}(C, f)$ is the concatenation of $(i_\star, j_\star)$ and the string S, where S is the concatenation of all the labels on the binary tree. We now define the encoded history, denoted as $\widetilde{\mathsf{History}}(C, f)$, as the concatenation of $(i_\star, j_\star)$ and *a Reed–Muller encoding* of S. The new selector is given oracle access to two candidate encoded histories together with f. By applying low-degree tests and self-correction of polynomials, we can assume that the Reed–Muller parts of the two candidates are indeed low-degree polynomials. Then we can use a reduction to polynomial identity testing to compute the first differing point between $\{v^1_{i,j}\}$ and $\{v^2_{i,j}\}$ in randomised polynomial time. See the proof of Lemma 7.5.3 for the details. This part is similar to the selector construction from [Hir15].

7.1.4 Karp–Lipton collapses and the half-exponential barrier

In the following, we elaborate on the half-exponential barrier mentioned earlier in the introduction.[20] Let $\mathcal{C}$ be a "typical" uniform complexity class containing P, a *Karp–Lipton collapse* to $\mathcal{C}$ states that if a large class (say EXP) has polynomial-size circuits, then this class collapses to $\mathcal{C}$. For example, there is a Karp–Lipton collapse to $\mathcal{C} = \Sigma_2\mathsf{P}$:

> Suppose $\mathsf{EXP} \subseteq \mathsf{P}/_{\mathrm{poly}}$, then $\mathsf{EXP} = \Sigma_2\mathsf{P}$. ([KL80], attributed to Albert Meyer)

Now, assuming that $\mathsf{EXP} \subseteq \mathsf{P}/_{\mathrm{poly}} \implies \mathsf{EXP} = \mathcal{C}$, the following win-win analysis implies that $\mathcal{C}$-EXP, the exponential-time version of $\mathcal{C}$, is not in $\mathsf{P}/_{\mathrm{poly}}$: (1) if $\mathsf{EXP} \not\subset \mathsf{P}/_{\mathrm{poly}}$, then of course $\mathcal{C}$-$\mathsf{EXP} \supseteq \mathsf{EXP}$ does not have polynomial-size circuits; (2) otherwise $\mathsf{EXP} \subseteq \mathsf{P}/_{\mathrm{poly}}$. We have $\mathsf{EXP} = \mathcal{C}$ and by padding $\mathsf{EEXP} = \mathcal{C}$-$\mathsf{EXP}$. Since EEXP contains a function of maximum circuit complexity by direct diagonalisation, it follows that $\mathcal{C}$-EXP does not have polynomial-size circuits.

Karp–Lipton collapses are known for the classes $\Sigma_2\mathsf{P}$ [KL80], $\mathsf{ZPP}^{\mathsf{NP}}$ [BCG+96], $\mathsf{S}_2\mathsf{P}$ [Cai07] (attributed to Samik Sengupta), PP, MA [LFKN92, BFNW93], and $\mathsf{ZPP}^{\mathsf{MCSP}}$ [IKV18]. All the aforementioned super-polynomial circuit lower bounds for $\Sigma_2\mathsf{EXP}$, $\mathsf{ZPEXP}^{\mathsf{NP}}$, $\mathsf{S}_2\mathsf{EXP}$, PEXP, MAEXP, and $\mathsf{ZPEXP}^{\mathsf{MCSP}}$ are proven in this way.[21]

The half-exponential barrier. The above argument is very successful at proving various super-polynomial lower bounds. However, a closer look shows that it is only capable of proving *sub-half-exponential* circuit lower bounds. Indeed, suppose we want to show that $\mathcal{C}$-EXP does not have circuits of size $f(n)$. We will have to perform the following win-win analysis:

- if $\mathsf{EXP} \not\subset \mathsf{SIZE}[f(n)]$, then of course $\mathcal{C}$-$\mathsf{EXP} \supseteq \mathsf{EXP}$ does not have circuits of size $f(n)$;

[20] A function $f\colon \mathbb{N} \to \mathbb{N}$ is *sub-half-exponential* if $f(f(n)^c) = 2^{o(n)}$ for every constant $c \geq 1$, i.e., composing f twice yields a subexponential function. For example, for constants $c \geq 1$ and $\varepsilon > 0$, the functions $f(n) = n^c$ and $f(n) = 2^{\log^c n}$ are sub-half-exponential, but the functions $f(n) = 2^{n^\varepsilon}$ and $f(n) = 2^{\varepsilon n}$ are not.

[21] There are some evidences that Karp–Lipton collapses are essential for proving circuit lower bounds [CMMW19].

- if $\mathsf{EXP} \subseteq \mathsf{SIZE}[f(n)]$, then (a scaled-up version of) the Karp–Lipton collapse implies that EXP can be computed by a $\mathcal{C}$ machine of $\mathrm{poly}(f(n))$ time. Note that $\mathsf{TIME}[2^{\mathrm{poly}(f(n))}]$ does not have circuits of size $f(n)$ by direct diagonalisation. By padding, $\mathsf{TIME}[2^{\mathrm{poly}(f(n))}]$ can be computed by a $\mathcal{C}$ machine of $\mathrm{poly}(f(\mathrm{poly}(f(n))))$ time. Therefore, if f is sub-half-exponential (meaning $f(\mathrm{poly}(f(n))) = 2^{o(n)}$), then $\mathcal{C}$-EXP does not have circuits of size $f(n)$.

Intuitively speaking, the two cases above are *competing with each other*: we cannot get exponential lower bounds in both cases.

7.2 Preliminaries

Notation. We use $[n]$ to denote $\{1, 2, \ldots, n\}$. A search problem Π maps every input $x \in \{0,1\}^*$ into a solution set $\Pi_x \subseteq \{0,1\}^*$. We say an algorithm A solves the search problem Π on input x if $A(x) \in \Pi_x$.

7.2.1 Complexity Classes

We assume basic familiarity with computational complexity theory (see, e.g., [AB09, Gol08] for references). Below we recall the definition of $\mathsf{S_2TIME}[T(n)]$ [RS98, Can96].

Definition 7.2.1. Let $T\colon \mathbb{N} \to \mathbb{N}$. We say a language $L \in \mathsf{S_2TIME}[T(n)]$, if there exists an $O(T(n))$-time verifier $V(x, \pi_1, \pi_2)$ that takes $x \in \{0,1\}^n$ and $\pi_1, \pi_2 \in \{0,1\}^{T(n)}$ as input, satisfying that

- if $x \in L$, then there exists π_1 such that for every π_2, $V(x, \pi_1, \pi_2) = 1$, and
- if $x \notin L$, then there exists π_2 such that for every π_1, $V(x, \pi_1, \pi_2) = 0$.

Moreover, we say $L \in \mathsf{S_2E}$ if $L \in \mathsf{S_2TIME}[T(n)]$ for some $T(n) \leq 2^{O(n)}$, and $L \in \mathsf{S_2P}$ if $L \in \mathsf{S_2TIME}[p(n)]$ for some polynomial p.

It is known that $\mathsf{S_2P}$ contains MA and $\mathsf{P^{NP}}$ [RS98], and $\mathsf{S_2P}$ is contained in $\mathsf{ZPP^{NP}}$ [Cai07]. From its definition, it is also clear that $\mathsf{S_2P} \subseteq \Sigma_2\mathsf{P} \cap \Pi_2\mathsf{P}$.

7.2.2 Single-valued $\mathsf{F\Sigma_2P}$ and $\mathsf{FS_2P}$ Algorithms

We consider the following definitions of single-valued algorithms, which correspond to circuit lower bounds for $\Sigma_2\mathsf{E}$ and $\mathsf{S_2E}$.

Definition 7.2.2 (Single-valued $\mathsf{F\Sigma_2P}$ and $\mathsf{FS_2P}$ algorithms)**.** A single-valued $\mathsf{F\Sigma_2P}$ algorithm A is specified by a polynomial $\ell(\cdot)$ together with a polynomial-time algorithm $V_A(x, \pi_1, \pi_2)$. On an input $x \in \{0,1\}^*$, we say that A outputs $y_x \in \{0,1\}^*$, if the following hold:

(a) There is a $\pi_1 \in \{0,1\}^{\ell(|x|)}$ such that for every $\pi_2 \in \{0,1\}^{\ell(|x|)}$, $V_A(x, \pi_1, \pi_2)$ outputs y_x.

(b) For every $\pi_1 \in \{0,1\}^{\ell(|x|)}$, there is a $\pi_2 \in \{0,1\}^{\ell(|x|)}$ such that the output of $V_A(x, \pi_1, \pi_2)$ is either y_x or $\perp$ (where $\perp$ indicates "I don't know").

A single-valued $\mathsf{FS_2P}$ algorithm A is specified similarly, except that we replace the second condition above with the following:

(b') There is a $\pi_2 \in \{0,1\}^{\ell(|x|)}$ such that for every $\pi_1 \in \{0,1\}^{\ell(|x|)}$, $V_A(x, \pi_1, \pi_2)$ outputs y_x.

Now, we say that a single-valued $\mathsf{F\Sigma_2P}$ ($\mathsf{FS_2P}$) algorithm A solves a search problem Π on input x if it outputs a string y_x and $y_x \in \Pi_x$. Note that from Definition 7.2.2, if A outputs a string y_x, then y_x is unique.

For convenience, we mostly only consider single-valued algorithms $A(x)$ with fixed output lengths, meaning that the output length $|A(x)|$ only depends on $|x|$ and can be computed in polynomial time given $1^{|x|}$.[22]

Single-Valued $\mathsf{FS_2P}$ and $\mathsf{F\Sigma_2P}$ algorithms with $\mathsf{FP^{NP}}$ post-processing

We also need the fact that single-valued $\mathsf{FS_2P}$ or $\mathsf{F\Sigma_2P}$ algorithms with $\mathsf{FP^{NP}}$ post-processing can still be implemented by single-valued $\mathsf{FS_2P}$ or $\mathsf{F\Sigma_2P}$ algorithms, respectively. More specifically, we have:

Theorem 7.2.3. *Let $A(x)$ be a single-valued $\mathsf{FS_2P}$ (resp. $\mathsf{F\Sigma_2P}$) algorithm and $B(x,y)$ be an $\mathsf{FP^{NP}}$ algorithm, both with fixed output length. The function $f(x) \coloneqq B(x, A(x))$ also admits an $\mathsf{FS_2P}$ (resp. $\mathsf{F\Sigma_2P}$) algorithm.*

Proof. We only provide a proof for the case of single-valued $\mathsf{FS_2P}$ algorithms. Recall that the Lexicographically Maximum Satisfying Assignment problem (LMSAP) is defined as follows: given an n-variable formula ϕ together with an integer $k \in [n]$, one needs to decide whether $a_k = 1$, where $a_1, \ldots, a_n \in \{0,1\}^n$ is the lexicographically largest assignment satisfies ϕ. By [Kre88], LMSAP is $\mathsf{P^{NP}}$-complete.

Let $V_A(x, \pi_1, \pi_2)$ be the corresponding verifier for the single-valued $\mathsf{FS_2P}$ algorithm A. Let $L(x, y, i)$ be the $\mathsf{P^{NP}}$ language such that $L(x, y, i) = 1$ if and only if $B(x, y)_i = 1$. Let $\ell = |B(x, y)|$ be the output length of B. We now define a single-valued $\mathsf{FS_2P}$ algorithm $\widetilde{A}$ by defining the following verifier $V_{\widetilde{A}}$, and argue that $\widetilde{A}$ computes f.

The verifier $V_{\widetilde{A}}$ takes an input x and two proofs $\vec{\pi}_1$ and $\vec{\pi}_2$, where $\vec{\pi}_1$ consists of ω_1, acting as the second argument to V_A, and ℓ assignments $z_1^1, z_2^1, \ldots, z_\ell^1 \in \{0,1\}^m$. Similarly, $\vec{\pi}_2$ consists of ω_2 and $z_1^2, z_2^2, \ldots, z_\ell^2 \in \{0,1\}^m$.

First, $V_{\widetilde{A}}$ runs $V_A(x, \omega_1, \omega_2)$ to get $y \in \{0,1\}^{|A(x)|}$. Then it runs the reduction from $L(x, y, i)$ to LMSAP for every $i \in [\ell]$ to obtain ℓ instances $\{(\phi_i, k_i)\}_{i \in [\ell]}$, where ϕ_i is an m-variable formula and $k_i \in [m]$. (Without loss of generality by padding dummy variables, we may assume that the number of variables in ϕ_i is the same for each i, i.e., m; and that m only depends on $|x|$ and $|y|$.) Now, for every $\mu \in [2]$, we can define an answer $w_\mu \in \{0,1\}^\ell$ by $(w_\mu)_i = (z_i^\mu)_{k_i}$ (i.e., the value of $B(x, y)$, assuming that $\vec{\pi}_\mu$ consists of the lexicographically largest assignments for all the LMSAP instances).

In what follows, when we say that $V_{\widetilde{A}}$ *selects* the proof $\mu \in [2]$, we mean that $V_{\widetilde{A}}$ outputs w_μ and terminates. Then, $V_{\widetilde{A}}$ works as follows:

[22] If A takes multiple inputs like x, y, z, then the output length $A(x, y, z)$ only depends on $|x|, |y|, |z|$ and can be computed in polynomial time given $1^{|x|}$, $1^{|y|}$, and $1^{|z|}$.

1. For each $\mu \in [2]$, it first checks whether for every $i \in [\ell]$, z_i^μ satisfies ϕ_i. If only one of the μ passes all the checks, $V_{\widetilde{A}}$ selects that μ. If none of them pass all the checks, $V_{\widetilde{A}}$ selects 1. Otherwise, it continues to the next step.

2. Now, letting $Z^\mu = z_1^\mu \circ z_2^\mu \circ \ldots \circ z_\ell^\mu$ for each $\mu \in [2]$. $V_{\widetilde{A}}$ selects the μ with the lexicographically larger Z^μ. If $Z^1 = Z^2$, then $V_{\widetilde{A}}$ selects 1.

Now we claim that $\widetilde{A}$ computes $f(x)$, which can be established by setting $\vec{\pi}_1$ or $\vec{\pi}_2$ be the corresponding proof for V_A concatenated with all lexicographically largest assignments for the $\{\phi_i\}_{i\in[\ell]}$. □

7.2.3 The Truth Table Generator

Proving circuit lower bounds (for exponential-time classes) is equivalent to solving the range avoidance problem on the *truth table generator* $\mathsf{TT}_{n,s}$, defined as follows. It was shown in [FM05] that for $n, s \in \mathbb{N}$, any s-size n-input circuit C can be encoded as a *stack program* with description size $L_{n,s} := (s+1)(7+\log(n+s))$. The precise definition of stack programs does not matter (see [FM05] for a formal definition); the only property we need is that given s and n such that $n \le s \le 2^n$, in $\mathrm{poly}(2^n)$ time one can construct a circuit $\mathsf{TT}_{n,s} \colon \{0,1\}^{L_{n,s}} \to \{0,1\}^{2^n}$ mapping the description of a stack program into its truth table. By the equivalence between stack programs and circuits, it follows that any $f \in \{0,1\}^{2^n} \setminus \mathrm{Range}(\mathsf{TT}_{n,s})$ satisfies $\mathsf{SIZE}(f) > s$. Also, we note that for large enough $n \in \mathbb{N}$ and $s = 2^n/n$, we have $L_{n,s} < 2^n$.

Fact 7.2.4. *Let $s(n) \colon \mathbb{N} \to \mathbb{N}$. Suppose that there is a single-valued $\mathsf{FS_2P}$ algorithm A such that for infinitely many $n \in \mathbb{N}$, $A(1^{2^n})$ takes $\alpha(n)$ bits of advice and outputs a string $f_n \in \{0,1\}^{2^n} \setminus \mathrm{Range}(\mathsf{TT}_{n,s(n)})$. Then $\mathsf{S_2E}/_{\alpha(n)} \not\subset \mathsf{SIZE}[s(n)]$.*

Proof sketch. We define a language L such that the truth table of the characteristic function of $L \cap \{0,1\}^n$ is $A(1^{2^n})$. It is easy to see that $L \notin \mathsf{SIZE}[s(n)]$ and $L \in \mathsf{S_2E}/_{\alpha(n)}$. □

7.3 The Jeřábek–Korten Reduction

Our results crucially rely on a reduction in [Jeř04, Kor21] showing that proving circuit lower bounds is "the hardest explicit construction" under P^{NP} reductions.

Notation. Let s be a string of length n. We will always use 0-index (i.e., the first bit of s is s_0 and the last bit of s is s_{n-1}). Let $i < j$, we use $s_{[i,j]}$ to denote the substring of s from the i-th bit to the j-th bit, and $s_{[i,j)}$ to denote the substring of s from the i-th bit to the $(j-1)$-th bit. (Actually, we will use the notation $s_{[i,j)}$ more often than $s_{[i,j]}$ as it is convenient when we describe the GGM tree.) We also use $s_1 \circ s_2 \circ \cdots \circ s_k$ to denote the concatenation of k strings.

7.3.1 GGM Tree and the Reduction

We first recall the GGM tree construction from [GGM86], which is used in a crucial way by [Jeř04, Kor21].

Definition 7.3.1 (The GGM tree construction [GGM86])**.** Let $C\colon \{0,1\}^n \to \{0,1\}^{2n}$ be a circuit. Let $n, T \in \mathbb{N}$ be such that $T \geq 4n$ and let k be the smallest integer such that $2^k n \geq T$. The function $\mathsf{GGM}_T[C]\colon \{0,1\}^n \to \{0,1\}^T$ is defined as follows.

Consider a perfect binary tree with 2^k leaves, where the root is on level 0 and the leaves are on level k. Each node is assigned a binary string of length n, and for $0 \leq j < 2^i$, denote $v_{i,j} \in \{0,1\}^n$ the value assigned to the j-th node on level i. Let $x \in \{0,1\}^n$. We perform the following computation to obtain $\mathsf{GGM}_T[C](x)$: we set $v_{0,0} := x$, and for each $0 \leq i < k$, $0 \leq j < 2^i$, we set $v_{i+1,2j} := C(v_{i,j})_{[0,n)}$ (i.e., the first half of $C(v_{i,j})$) and $v_{i+1,2j+1} := C(v_{i,j})_{[n,2n)}$ (i.e., the second half of $C(v_{i,j})$). (We say the nodes $(i+1, 2j)$ and $(i+1, 2j+1)$ are "children" of (i, j).)

Finally, we concatenate all values of the leaves and take the first T bits as the output:

$$\mathsf{GGM}_T[C](x) := (v_{k,0} \circ v_{k,1} \circ \cdots \circ v_{k,2^k-1})_{[0,T)}.$$

Lemma 7.3.2 (The output of GGM tree has a small circuit)**.** *Let* $\mathsf{GGMEval}(C, T, x, i)$ *denote the* i*-th bit of* $\mathsf{GGM}_T[C](x)$*. There is an algorithm running in* $\widetilde{O}(|C| \cdot \log T)$ *time that, given* C, T, x, i*, outputs* $\mathsf{GGMEval}(C, T, x, i)$*.*

Proof Sketch. We first note that to compute the i-th bit of $\mathsf{GGM}_T[C](x) := (v_{k,0} \circ v_{k,1} \circ \cdots \circ v_{k,2^k-1})_{[0,T)}$, it suffices to compute $v_{k,\lfloor i/n \rfloor}$. Computing $v_{k,\lfloor i/n \rfloor}$ can be done by descending from the root of the GGM tree to the leave $(k, \lfloor i/n \rfloor)$, which takes $\widetilde{O}(|C| \cdot \log T)$ time. □

It is shown in [Kor21] that the range avoidance problem for C reduces to the range avoidance problem for $\mathsf{GGM}_T[C]$. In what follows, we review this proof, during which we also define the *computational history* of "solving range avoidance of C from $\mathsf{GGM}_T[C]$", which will be crucial in our main proof.

Lemma 7.3.3 (Reduction from solving range avoidance of C to solving range avoidance of $\mathsf{GGM}_T[C]$)**.** *Let* $C\colon \{0,1\}^n \to \{0,1\}^{2n}$ *be a circuit. Let* f *be a non-output of* $\mathsf{GGM}_T[C]$*, i.e.,* $f \in \{0,1\}^T \setminus \mathrm{Range}(\mathsf{GGM}_T[C])$*. Then,* $\mathsf{Jeřábek\text{–}Korten}(C, f)$ *(as defined in Algorithm 7.3.1) outputs a non-output of* C *in deterministic* $\mathrm{poly}(T, n)$ *time with an* NP *oracle.*

Proof Sketch. The running time of $\mathsf{Jeřábek\text{–}Korten}(C, f)$ follows directly from its description. Also, note that whenever $\mathsf{Jeřábek\text{–}Korten}(C, f)$ outputs a string $v_{i+1,2j} \circ v_{i+1,2j+1} \in \{0,1\}^{2n}$, it holds that this string is not in the range of C. Therefore, it suffices to show that when $f \in \{0,1\}^T \setminus \mathrm{Range}(\mathsf{GGM}_T[C])$, $\mathsf{Jeřábek\text{–}Korten}(C, f)$ does not return $\perp$.

Assume, towards a contradiction, that $\mathsf{Jeřábek\text{–}Korten}(C, f)$ returns $\perp$. This means that all the $\{v_{i,j}\}_{i,j}$ values are set. It follows from the algorithm description that $f = \mathsf{GGM}_T[C](v_{0,0})$, which contradicts the assumption that $f \in \{0,1\}^T \setminus \mathrm{Range}(\mathsf{GGM}_T[C])$. □

In addition, we observe the following trivial fact:

Fact 7.3.4. *Let* $C : \{0,1\}^n \to \{0,1\}^{2n}$ *be a circuit,* $T := 2^{2n} \cdot 2n$*, and* f *be the concatenation of all length-*$2n$ *strings (which has length* T*). Then* $f \notin \mathrm{Range}(\mathsf{GGM}_T[C])$*.*

One can combine Fact 7.3.4 with Lemma 7.3.3 to obtain a brute force algorithm that solves the range avoidance problem in $2^{O(n)}$ time with an NP oracle. Essentially, this brute force

Algorithm 7.3.1: Jeřábek–Korten(C, f): The Jeřábek–Korten reduction

Input: $C\colon \{0,1\}^n \to \{0,1\}^{2n}$ denotes the input circuit, and
$f \in \{0,1\}^T \setminus \mathrm{Range}(\mathsf{GGM}_T[C])$ denotes the input "hard" truth table
Output: A non-output of C
Data: The computational history of Jeřábek–Korten(C, f): a pair $(i_\star, j_\star)$ and an array $\{v_{i,j}\}_{i,j}$ where $i \in \{0, 1 \ldots, k\}$ and $j \in \{0, 1, \ldots, 2^i\}$.

Let $k \leftarrow \lceil \log_2(T/n) \rceil$;
Append f with $2^k n - |f|$ zeros at the end;
for $j \leftarrow 0$ to $2^k - 1$ **do**
    $v_{k,j} \leftarrow f_{[jn,(j+1)n)}$;
    /* the j-th "block" of f */
for $i \leftarrow k - 1$ downto 0 **do**
    for $j \leftarrow 2^i - 1$ downto 0 **do**
        Let $v_{i,j}$ be the lexicographically smallest string in $C^{-1}(v_{i+1,2j} \circ v_{i+1,2j+1})$;
        /* Note that this step needs to invoke the NP oracle */
        if $v_{i,j}$ does not exist **then**
            For every (i', j') such that $v_{i',j'}$ is not set yet, set $v_{i',j'} \leftarrow \bot$;
            Set $i_\star := i$, and $j_\star := j$;
            return $v_{i+1,2j} \circ v_{i+1,2j+1}$;
return $\bot$

algorithm tests every possible length-$2n$ string against the range of the circuit. It will be the basis of our win-win analysis in Section 7.4.

Finally, we give the following remark, showing that the Jeřábek–Korten reduction relativises.

Remark 7.3.5. Algorithm 7.3.1 and Lemma 7.3.3 *relativises*, in the sense that if the input is actually an oracle circuit C^O for some arbitrary oracle, the algorithm still works except now it needs to call an NP^O oracle to find the lexicographically smallest string in $C^{-1}(v_{i+1,2j} \circ v_{i+1,2j+1})$.

7.3.2 Π_1 Verification of the History of Jeřábek–Korten(C, f)

In what follows, we say that $(i, j) < (i', j')$ if either $i < i'$ or $(i = i'$ and $j < j')$ (that is, we consider the lexicographical order of pairs). Observe that Algorithm 7.3.1 processes all the pairs (i, j) in the reverse lexicographic order.

Definition 7.3.6 (The computational history of Jeřábek–Korten(C, f))**.** Let $n, T \in \mathbb{N}$ be such that $\log T \le n \le T$. Let $C\colon \{0,1\}^n \to \{0,1\}^{2n}$ be a circuit, and $f \in \{0,1\}^T$ be a "hard truth table" in the sense that $f \notin \mathrm{Range}(\mathsf{GGM}_T[C])$. The *computational history* of Jeřábek–Korten(C, f), denoted as

$$\mathsf{History}(C, f),$$

consists of $(i_\star, j_\star)$, as well as the concatenation of $v_{i,j}$ for every $0 \le i < k$ and $0 \le j < 2^i$, in the lexicographical order of (i, j) ($(i_\star, j_\star)$ and the $v_{i,j}$ are defined in Algorithm 7.3.1). Each $v_{i,j}$ is encoded by $n + 1$ bits $\mathsf{enc}(v_{i,j})$, where if $v_{i,j} \in \{0,1\}^n$ then $\mathsf{enc}(v_{i,j}) = 0 \circ v_{i,j}$, and if $v_{i,j} = \bot$ then $\mathsf{enc}(v_{i,j}) = 1^{n+1}$. The length of this history is at most $(2^{k+1} - 1)(n + 1) + 2 \log T \le 5T$, and for convenience we always pad zeros at the end so that its length becomes exactly $5T$.

The following lemma summarises the properties of the computational history construction above required for the $\Sigma_2\mathsf{E}$ lower bound in the next section.

Lemma 7.3.7. *Let $n, T \in \mathbb{N}$ be such that $\log T \leq n \leq T$. Let $C\colon \{0,1\}^n \to \{0,1\}^{2n}$ be a circuit and $f \in \{0,1\}^T \setminus \mathrm{Range}(\mathsf{GGM}_T[C])$. Let $h \coloneqq \mathsf{History}(C, f)$ and $z \coloneqq \text{Jeřábek–}\mathsf{Korten}(C, f)$.*

1. **(history contains input/output)** *There is a $\mathrm{poly}(\log T)$-time one-query oracle algorithm in and an $O(n)$-time oracle algorithm Output, both having input parameters T, n and taking a string $\tilde{h} \in \{0,1\}^{5T}$ as oracle, such that the following hold:*
   (a) *When given h as the oracle, $\mathsf{in}_{T,n}$ takes an additional input $i \in \{0, 1, \ldots, 5T-1\}$ and outputs f_i.*
   (b) *When given h as the oracle, $\mathsf{Output}_{T,n}$ outputs $z = \text{Jeřábek–}\mathsf{Korten}(C, f)$.*
2. **(Π_1 verification of the history)** *There is an oracle algorithm V with input parameters T, n such that the following holds:*
   (a) *V takes $\tilde{f} \in \{0,1\}^T, \tilde{h} \in \{0,1\}^{5T}$ as oracles and C and $w \in \{0,1\}^{5\cdot(\log T+n)}$ as inputs. It runs in $\mathrm{poly}(n)$ time.*
   (b) *$h = \mathsf{History}(C, f)$ is the unique string from $\{0,1\}^{5T}$ satisfying the following:*

$$V^{f,h}(C, w) = 1 \qquad \text{for every } w \in \{0,1\}^{5\cdot(\log T+n)}.$$

Proof. From the definition of $\mathsf{History}(C, f)$, the construction of $\mathsf{in}_{T,n}$ and $\mathsf{Output}_{T,n}$ are straightforward. Now we describe the verifier $V^{f,\tilde{h}}$, where $f \in \{0,1\}^T$ and $\tilde{h} \in \{0,1\}^{5T}$. Note that here we fix the first oracle of V to be the input truth table f, while the second oracle $\tilde{h}$ can be any string from $\{0,1\}^{5T}$.

First, V reads $(i_\star, j_\star)$ from $\tilde{h}$. Note that the rest of $\tilde{h}$ can be parsed as an array $\{v_{i,j}\}_{i,j}$ where $i \in \{0, 1 \ldots, k\}$ and $j \in \{0, 1, \ldots, 2^i\}$. We will think of V as performing at most $2^{|w|}$ checks, each of which *passes* or *fails*. To show the second item of the lemma, we need to show that (1) if a string $\tilde{h}$ passes all the checks, then it must be the case that $\tilde{h} = h$; and (2) h passes all the checks.

Specifically, V checks $\tilde{h}$ as follows:

- The values written on the leaves of $\{v_{i,j}\}$ are indeed f. That is, for every $j \in \{0, 1, \ldots, 2^k - 1\}$, check that $v_{k,j}$ is consistent with the corresponding block in f.
- For every $(i, j) > (i_\star, j_\star)$ such that $i < k$, $C(v_{i,j}) = v_{i+1,2j} \circ v_{i+1,2j+1}$. (That is, the value $v_{i,j}$ is consistent with its two children.)
- For every $(i, j) > (i_\star, j_\star)$ such that $i < k$, for every $x \in \{0,1\}^n$ that is lexicographically smaller than $v_{i,j}$, $C(x) \neq v_{i+1,2j} \circ v_{i+1,2j+1}$. (That is, the value $v_{i,j}$ is the lexicographically first preimage of its two children.)
- For every $x \in \{0,1\}^n$, $C(x) \neq v_{i_\star+1,2j_\star} \circ v_{i_\star+1,2j_\star+1}$. (That is, the two children of $(i_\star, j_\star)$ form a non-output of C; by the previous checks, $(i_\star, j_\star)$ is the lexicographically largest such pair.)

- For every $(i,j) \leq (i_\star, j_\star)$, $v_{i,j} = \bot$.

Note that the above can be implemented with a universal ($\forall$) quantification over at most $5 \cdot (\log T + n)$ bits. First, one can see that by the definition of the correct history h (Definition 7.3.6), h passes all the checks above. Second, one can indeed see that all the conditions above *uniquely determine* h, and therefore any $\tilde{h}$ passing all the checks must equal h. □

Again, it is easy to observe that Definition 7.3.6 and Lemma 7.3.7 relativise.

Remark 7.3.8. Definition 7.3.6 and Lemma 7.3.7 *relativise*, in the sense that if C is an oracle circuit C^O for some arbitrary oracle, Definition 7.3.6 needs no modification since Algorithm 7.3.1 relativises, and Lemma 7.3.7 holds with the only modification that V now also need to take O as an oracle (since it needs to evaluate C).

7.4 Circuit Lower Bounds for $\Sigma_2\mathsf{E}$

In this section, we prove our near-maximum circuit lower bounds for $\Sigma_2\mathsf{E}$ by providing a new single-valued $\mathsf{F}\Sigma_2\mathsf{P}$ algorithm for Avoid.

Let $\{C_n \colon \{0,1\}^n \to \{0,1\}^{2n}\}_{n\in\mathbb{N}}$ be a P-uniform family of circuits. We show that there is a single-valued $\mathsf{F}\Sigma_2\mathsf{P}$ algorithm A that, on input 1^n, outputs a canonical string that is outside the range of C_n for infinitely many $n \in \mathbb{N}$.

Theorem 7.4.1. *Let $\{C_n \colon \{0,1\}^n \to \{0,1\}^{2n}\}_{n\in\mathbb{N}}$ be a P-uniform family of circuits. There is a single-valued $\mathsf{F}\Sigma_2\mathsf{P}$ algorithm A with one bit of advice such that for infinitely many $n \in \mathbb{N}$, $A(1^n)$ outputs $y_n \in \{0,1\}^{2n} \setminus \mathrm{Range}(C_n)$.*

Proof. We begin with some notation.

Notation. Let $n^{(1)}$ be a large enough power of 2, $n^{(\ell)} = 2^{2^{n^{(\ell-1)}}}$ for each integer $\ell > 1$. Let $n_0^{(\ell)} = n^{(\ell)}$ and $t^{(\ell)} = O\left(\log n_0^{(\ell)}\right)$ be parameters that we set later. For each $1 \leq i \leq t^{(\ell)}$, let $n_i^{(\ell)} := \left(n_{i-1}^{(\ell)}\right)^{10}$. To show our algorithm A works on infinitely many input lengths, we will show that for every $\ell \in \mathbb{N}$, there is an input length $n_i^{(\ell)}$ for some $i \in \{0, 1, \ldots, t^{(\ell)}\}$ such that A works.

Fix $\ell \in \mathbb{N}$. From now on, for convenience, we will use n_i and t to denote $n_i^{(\ell)}$ and $t^{(\ell)}$, respectively.

Specifying T_i and f_i. For each input length n_i, we will specify a parameter $T_i \in \mathbb{N}$ and a string $f_i \in \{0,1\}^{T_i}$. Our win-win analysis is based on whether $f_i \in \mathrm{Range}(\mathsf{GGM}_{T_i}[C_{n_i}])$ for each $i \in \{0, 1, \ldots, t\}$.

Let $T_0 := 2^{2n_0} \cdot 2n_0$ and f_0 be the concatenation of all length-$2n_0$ strings (which has length T_0). From Fact 7.3.4, we have that $f_0 \notin \mathrm{Range}(\mathsf{GGM}_{T_0}[C_{n_0}])$. For every $i \in [t]$, we define

$$f_i := \mathsf{History}(C_{n_{i-1}}, f_{i-1}).$$

From Definition 7.3.6, this also means that we have set $T_i = 5 \cdot T_{i-1}$ for every $i \in [t]$.

Let t be the first integer such that $T_{t+1} \leq 4n_{t+1}$. Note that we have $T_i = 5^i \cdot T_0 \leq 2^{3n_0 + i \cdot \log 5}$ and $n_i = (n_0)^{10^i} = 2^{\log n_0 \cdot 10^i}$. Hence, we have that $t \leq O(\log n_0)$. (Also note that $n_t^{(\ell)} < n_0^{(\ell+1)}$.)

Description of our $\mathsf{F\Sigma_2P}$ algorithm A. Now, let $k \in \{0, 1, \ldots, t\}$ be the largest integer such that $f_k \notin \mathrm{Range}(\mathsf{GGM}_{T_k}[C_{n_k}])$. Since $f_0 \notin \mathrm{Range}(\mathsf{GGM}_{T_0}[C_{n_0}])$, such a k must exist. Let $z := \mathsf{Jeřábek\text{–}Korten}(C_{n_k}, f_k)$. It follows from Lemma 7.3.3 that z is not in the range of C_{n_k}. Our single-valued $\mathsf{F\Sigma_2P}$ algorithm A computes z on input 1^{n_k} (see Definition 7.2.2). That is, for some $\ell_1, \ell_2 \leq \mathrm{poly}(n_k)$:

- There exists $\pi_1 \in \{0,1\}^{\ell_1}$ such that for every $\pi_2 \in \{0,1\}^{\ell_2}$, $V_A(1^{n_k}, \pi_1, \pi_2)$ prints z, and
- For every $\pi_1 \in \{0,1\}^{\ell_1}$, there exists some $\pi_2 \in \{0,1\}^{\ell_2}$ such that $V_A(1^{n_k}, \pi_1, \pi_2)$ prints either z or $\perp$.

In more details, if $k < t$, then V_A treats π_1 as an input to the circuit $\mathsf{GGM}_{T_{k+1}}[C_{n_{k+1}}]$, and let

$$\hat{f}_{k+1} := \mathsf{GGM}_{T_{k+1}}[C_{n_{k+1}}](\pi_1).$$

Here, the length of π_1 is $\ell_1 := n_{k+1} \leq \mathrm{poly}(n_k)$. If $k = t$, then V_A defines $\hat{f}_{k+1} := \pi_1$ and $\ell_1 := T_{t+1} \leq \mathrm{poly}(n_k)$. It is intended that $\hat{f}_{k+1} = f_{k+1} = \mathsf{History}(C_{n_k}, f_k)$ (which V_A needs to verify). Note that in the case where $k < t$, since $f_{k+1} \in \mathrm{Range}(\mathsf{GGM}_{T_{k+1}}[C_{n_{k+1}}])$, there indeed exists some π_1 such that $\hat{f}_{k+1} = f_{k+1}$.

We note that Lemma 7.3.2 provides us "random access" to the (potentially very long) string $\hat{f}_{k+1}$: given π_1 and $j \in [T_{k+1}]$, one can compute the j-th bit of $\hat{f}_{k+1}$ in $\mathrm{poly}(n_k)$ time. Also recall from Lemma 7.3.7 that for each i, $f_{i+1} = \mathsf{History}(C_{n_i}, f_i)$ contains the string f_i, which can be retrieved by the oracle algorithm in described in Item 1 of Lemma 7.3.7. Therefore, for each i from k downto 1, we can recursively define $\hat{f}_i$ such that $(\hat{f}_i)_j = \mathsf{in}^{\hat{f}_{i+1}}_{T_i, n_i}(j)$. We define $\hat{f}_0$ to be the concatenation of all length-$(2n_0)$ strings in the lexicographical order, so $\hat{f}_0 = f_0$. Applying the algorithm in recursively, we obtain an algorithm that given $i \in \{0, 1, \ldots, k\}$ and $j \in \{0, 1, \ldots, T_i - 1\}$, outputs the j-th bit of $\hat{f}_i$. Since in only makes one oracle query, this algorithm runs in $\mathrm{poly}(n_k)$ time.[23]

Then, V_A parses the second proof π_2 into $\pi_2 = (i, w)$ where $i \in \{0, 1, \ldots, k\}$ and $w \in \{0,1\}^{5(\log T_i + n_i)}$. Clearly, the length of π_2 is at most $\ell_2 := \log(k+1) + 5(\log T_k + n_k) \leq \mathrm{poly}(n_k)$. Now, let V_{History} be the oracle algorithm in Item 2 of Lemma 7.3.7, we let $V_A(1^{n_k}, \pi_1, \pi_2)$ check whether the following holds:

$$V^{\hat{f}_i, \hat{f}_{i+1}}_{\mathsf{History}}(C_{n_i}, w) = 1.^{24} \tag{7.1}$$

If this is true, then V_A outputs the string $z := \mathsf{Output}^{\hat{f}_{k+1}}_{T_k, n_k}$, where Output is the output oracle algorithm defined in Item 1 of Lemma 7.3.7. Otherwise, V_A outputs $\perp$.

The correctness of A. Before establishing the correctness of A, we need the following claim:

Claim 7.4.2. *$f_{k+1} = \hat{f}_{k+1}$ if and only if the following holds:*

- *$V^{\hat{f}_i, \hat{f}_{i+1}}_{\mathsf{History}}(C_{n_i}, w) = 1$ for every $i \in \{0, 1, \ldots, k\}$ and for every $w \in \{0,1\}^{5(\log T_i + n_i)}$.*

[23] Note that the definition of f_0 is so simple that one can directly compute the j-th bit of f_0 in $\mathrm{poly}(n_0)$ time.
[24] Here V_{History} also takes input parameters T_i and n_i. We omit them in the subscript for notational convenience.

Proof. First, assume that $f_{k+1} = \hat{f}_{k+1}$. By Item 1a of Lemma 7.3.7, we have that $\hat{f}_i = f_i$ for every $i \in \{0, 1, \ldots, k+1\}$. Recall that by definition, $f_{i+1} = \mathsf{History}(C_{n_i}, f_i)$ for every $i \in \{0, 1, \ldots, k\}$. Hence, by Item 2b of Lemma 7.3.7, we have that for every $i \in \{0, 1, \ldots, k\}$, and for every $w \in \{0,1\}^{5(\log T_i + n_i)}$, $V_{\mathsf{History}}^{\hat{f}_i, \hat{f}_{i+1}}(C_{n_i}, w) = 1$ holds.

For the other direction, suppose that for every $i \in \{0, 1, \ldots, k\}$ and $w \in \{0,1\}^{5(\log T_i + n_i)}$, we have that $V_{\mathsf{History}}^{\hat{f}_i, \hat{f}_{i+1}}(C_{n_i}, w) = 1$ holds. First recall that $f_0 = \hat{f}_0$ by definition. By an induction on $i \in [k+1]$ and (the uniqueness part of) Item 2b of Lemma 7.3.7, it follows that $f_i = \hat{f}_i$ for every $i \in \{0, 1, \ldots, k+1\}$. In particular, $f_{k+1} = \hat{f}_{k+1}$. ◇

Now we are ready to establish that A is a single-valued $\mathsf{F\Sigma_2P}$ algorithm computing z on input 1^{n_k}. We first prove the completeness of A; i.e., there is a proof π_1 such that for every π_2, $V_A(1^{n_k}, \pi_1, \pi_2)$ outputs $z = \mathsf{Jeřábek\text{–}Korten}(C_{n_k}, f_k)$. We set π_1 to be the following proof: If $k < t$, then $f_{k+1} \in \mathrm{Range}(\mathsf{GGM}_{T_{k+1}}[C_{n_{k+1}}])$, and we can set $\pi_1 \in \{0,1\}^{n_{k+1}}$ to be the input such that $f_{k+1} = \mathsf{GGM}_{T_{k+1}}[C_{n_{k+1}}](\pi_1)$; if $k = t$, then we simply set $\pi_1 = f_{k+1}$. Then, we have $f_{k+1} = \hat{f}_{k+1}$, and by Claim 7.4.2, we know that V_A will output $z = \mathsf{Jeřábek\text{–}Korten}(C_{n_k}, f_k)$ on every proof π_2.

Next, we show that for every π_1, there is some π_2 that makes V_A output either z or $\perp$. It suffices to consider π_1 such that for every π_2, $V_A(1^{n_k}, \pi_1, \pi_2) \neq \perp$. In this case, every invocation of Equation 7.1 holds, and thus by Claim 7.4.2 we know that $f_{k+1} = \hat{f}_{k+1}$. It follows that $\mathsf{Jeřábek\text{–}Korten}(C_{n_k}, f_k) = z$ and V_A will output z regardless of π_2.

Finally, we generalise A and V_A to work on all inputs 1^n. On input 1^n, V_A calculates the largest ℓ such that $n^{(\ell)} \leq n$, and also calculates the largest k' such that $n_{k'}^{(\ell)} \leq n$. If $n_{k'}^{(\ell)} \neq n$, then V_A immediately outputs $\perp$ and halts. Otherwise, V_A receives an advice bit indicating whether $k' = k^{(\ell)}$ where $k^{(\ell)}$ is the largest integer such that $f_{k^{(\ell)}}^{(\ell)} \notin \mathrm{Range}(\mathsf{GGM}_{T_k^{(\ell)}}[C_{n_k^{(\ell)}}])$. If this is the case, then V_A runs the verification procedure above; otherwise, it immediately outputs $\perp$ and halts. It is easy to see that V_A runs in $\mathrm{poly}(n)$ time, and is an infinitely-often single-valued $\mathsf{F\Sigma_2P}$ algorithm solving the range avoidance problem of $\{C_n\}_{n \in \mathbb{N}}$. □

From Remark 7.3.5 and Remark 7.3.8, one can observe that the proof above also relativises. Hence, we have the following as well.

Theorem 7.4.3 (Relativised version of Theorem 7.4.1)**.** *Let $\mathcal{O}\colon \{0,1\}^* \to \{0,1\}$ be any oracle. Let $\{C_n^{\mathcal{O}}\colon \{0,1\}^n \to \{0,1\}^{2n}\}_{n \in \mathbb{N}}$ be a P-uniform family of $\mathcal{O}$-oracle circuits. There is a single-valued $\mathsf{F\Sigma_2P}^{\mathcal{O}}$ algorithm $A^{\mathcal{O}}$ with one bit of advice such that for infinitely many $n \in \mathbb{N}$, $A^{\mathcal{O}}(1^n)$ outputs $y_n \in \{0,1\}^{2n} \setminus \mathrm{Range}(C_n^{\mathcal{O}})$.*

We omit the proof of the following corollary since it is superseded by the results in the next section.

Corollary 7.4.4. *$\Sigma_2\mathsf{E} \not\subseteq \mathsf{SIZE}[2^n/n]$ and $(\Sigma_2\mathsf{E} \cap \Pi_2\mathsf{E})/_1 \not\subseteq \mathsf{SIZE}[2^n/n]$. Moreover, these results relativise: for every oracle $\mathcal{O}$, $\Sigma_2\mathsf{E}^{\mathcal{O}} \not\subseteq \mathsf{SIZE}^{\mathcal{O}}[2^n/n]$ and $(\Sigma_2\mathsf{E}^{\mathcal{O}} \cap \Pi_2\mathsf{E}^{\mathcal{O}})/_1 \not\subseteq \mathsf{SIZE}^{\mathcal{O}}[2^n/n]$.*

7.5 Circuit Lower Bounds for $\mathsf{S_2E}$

In this section, we prove our near-maximum circuit lower bounds for $\mathsf{S_2E}/_1$ by giving a new single-valued $\mathsf{FS_2P}$ algorithm for Avoid.

7.5.1 Reed–Muller Codes

To prove maximum circuit lower bounds for $\mathsf{S}_2\mathsf{E}/_1$, we will need several standard tools for manipulating Reed–Muller (RM) codes (i.e., low-degree multi-variate polynomials).

For a polynomial $P\colon \mathbb{F}_p^m \to \mathbb{F}_p$, where $\mathbb{F}_p$ is the finite field of p elements, we use $\deg_{\max}(P)$ to denote the maximum individual degree of variables in P. Let p be a prime, $\Delta, m \in \mathbb{N}$. For a string $S \in \{0,1\}^{\Delta^m}$, we use $\mathsf{RM}_{\mathbb{F}_p,\Delta,m}(S)$ to denote its Reed–Muller encoding by extension: letting $H = \{0,1,\dots,\Delta-1\}$ and $w_1,\dots,w_{\Delta^m} \in H^m$ be the enumeration of all elements in H^m in the lexicographical order, $\mathsf{RM}_{\mathbb{F}_p,\Delta,m}(S)$ is the unique polynomial $P\colon \mathbb{F}_p^m \to \mathbb{F}_p$ such that (1) $P(w_i) = S_i$ for every $i \in [\Delta^m]$ and (2) $\deg_{\max}(P) \le \Delta - 1$.[25]

We also fix a Boolean encoding of $\mathbb{F}_p$ denoted as $\mathsf{Enc}_{\mathbb{F}_p}\colon \mathbb{F}_p \to \{0,1\}^{\lceil \log p \rceil}$. For simplicity, we can just map $z \in \{0,1,\dots,p-1\}$ to its binary encoding. In particular, $\mathsf{Enc}_{\mathbb{F}_p}(0) = 0^{\lceil \log p \rceil}$ and $\mathsf{Enc}_{\mathbb{F}_p}(1) = 0^{\lceil \log p \rceil - 1} \circ 1$.[26] Now we further define $\mathsf{BRM}_{\mathbb{F}_p,\Delta,m}(S)$ by concatenating $\mathsf{RM}_{\mathbb{F}_p,\Delta,m}(S)$ with $\mathsf{Enc}_{\mathbb{F}_p}$, thus obtaining a Boolean encoding again. Formally, letting $P = \mathsf{RM}_{\mathbb{F}_p,\Delta,m}(S)$ and $w_1,\dots,w_{p^m} \in \mathbb{F}_p^m$ be the enumeration of all elements from $\mathbb{F}_p^m$ in the lexicographic order, we define $\mathsf{BRM}_{\mathbb{F}_p,\Delta,m}(S) = \mathsf{Enc}_{\mathbb{F}_p}(P(w_1)) \circ \mathsf{Enc}_{\mathbb{F}_p}(P(w_2)) \circ \dots \circ \mathsf{Enc}_{\mathbb{F}_p}(P(w_{p^m}))$. We remark that for every $i \in [\Delta^m]$, in $\mathrm{poly}(m, \log p)$ time one can compute an index $i' \in [p^m \cdot \lceil \log p \rceil]$ such that $\mathsf{BRM}_{\mathbb{F}_p,\Delta,m}(S)_{i'} = S_i$.

We need three properties of Reed–Muller codes, which we explain below.

Self-correction for polynomials. We first need the following self-corrector for polynomials, which efficiently computes the value of P on any input given an oracle that is close to a low-degree polynomial P. (In other words, it is a *local decoder* for the Reed–Muller code.)

Lemma 7.5.1 (A self-corrector for polynomials, cf. [GS92, Sud95])**.** *There is a probabilistic oracle algorithm* PCorr *such that the following holds. Let p be a prime, $m, \Delta \in \mathbb{N}$ such that $\Delta < p/3$. Let $g\colon \mathbb{F}_p^m \to \mathbb{F}_p$ be a function such that for some polynomial P of total degree at most Δ,*

$$\Pr_{\vec{x} \leftarrow \mathbb{F}_p^m}[g(\vec{x}) \neq P(\vec{x})] \le 1/4.$$

Then for all $\vec{x} \in \mathbb{F}_p^m$, $\mathsf{PCorr}^g(p, m, \Delta, \vec{x})$ runs in time $\mathrm{poly}(\Delta, \log p, m)$ and outputs $P(\vec{x})$ with probability at least $2/3$.

Low-max-degree test. We also need the following efficient tester, which checks whether a given polynomial has maximum individual degree at most Δ or is far from such polynomials.[27]

Lemma 7.5.2 (Low-max-degree tester [BFL91, Remark 5.15])**.** *Let $n, \Delta, p \in \mathbb{N}$ be such that $p \ge 20 \cdot (\Delta+1)^2 \cdot n^2$ and p is a prime. There is a probabilistic non-adaptive oracle machine* LDT *such that the following holds. Let $g\colon \mathbb{F}_p^n \to \mathbb{F}_p$. Then for $\delta = 3n^2 \cdot (\Delta+1)/p$, it holds that*

1. *if $\deg_{\max}(g) \le \Delta$, then $\mathsf{LDT}^g(p, n, \Delta)$ accepts with probability 1,*

[25] To see the uniqueness of P, note that for every $P\colon \mathbb{F}_p^m \to \mathbb{F}_p$ with $\deg_{\max}(P) \le \Delta - 1$, the restriction of P to H^m uniquely determines the polynomial P. We can construct P using standard interpolation.

[26] This fact is useful because if we know a string $m \in \{0,1\}^{\lceil \log p \rceil}$ encodes either 0 or 1, then we can decode it by only querying the last bit of m.

[27] To obtain the theorem below, we set the parameter δ and ε from [BFL91, Remark 5.15] to be $\min\left(\frac{1}{200n^2(\Delta+1)}, 1/2p\right)$ and $\min\left(\frac{1}{400n^3(\Delta+1)}, 1/2p\right)$, respectively.

2. *if g is at least δ-far from every polynomial with maximum individual degree at most Δ, then $\mathsf{LDT}^g(p, n, \Delta)$ rejects with probability at least $2/3$, and*

3. LDT *runs in* $\mathrm{poly}(p)$ *time.*

Comparing two RM codewords. Lastly, we show an efficient algorithm that, given oracle access to two codewords of $\mathsf{RM}_{\mathbb{F}_p,\Delta,m}$, computes the lexicographically first differing point between the respective messages of the two codewords.

Lemma 7.5.3 (Comparing two RM codewords)**.** *Let p be a prime. Let $m, \Delta \in \mathbb{N}$ be such that $m \cdot \Delta < p/2$. There is a probabilistic oracle algorithm Comp that takes two polynomials $f, g \colon \mathbb{F}_p^m \to \mathbb{F}_p$ as oracles, such that if both $\deg_{\max}(f)$ and $\deg_{\max}(g)$ are at most Δ, then the following holds with probability at least $9/10$:*

- *If $f \neq g$, then $\mathsf{Comp}^{f,g}(p, m, \Delta)$ outputs the lexicographically smallest element w in H^m such that $f(w) \neq g(w)$, where $H = \{0, 1, \ldots, \Delta - 1\}$.*[28]
- *If $f = g$, then $\mathsf{Comp}^{f,g}(p, m, \Delta)$ outputs $\perp$.*
- Comp *makes at most $\mathrm{poly}(m \cdot \Delta)$ queries to both f and g, and runs in $\mathrm{poly}(m \cdot \Delta \cdot \log p)$ time.*

Proof. Our proof is similar to the proof from [Hir15], which only considers multi-linear polynomials. Our algorithm $\mathsf{Comp}^{f,g}(p, m, \Delta)$ works as follows:

1. The algorithm has m stages, where the i-th stage aims to find the i-th entry of w. At the end of the i-th stage, the algorithm obtains a length-i prefix of w.

2. For every $i \in [m]$:

   (a) Let $w_{<i} \in H^{i-1}$ be the current prefix. For every $h \in \{0, 1, \ldots, \Delta - 1\}$, we run a randomised polynomial identity test to check whether the restricted polynomial $f(w_{<i}, h, \cdot)$ and $g(w_{<i}, h, \cdot)$ are the same, with error at most $\frac{1}{10m|H|}$.[29]

   (b) We set w_i to be the smallest h such that our test above reports that $f(w_{<i}, h, \cdot)$ and $g(w_{<i}, h, \cdot)$ are distinct. If there is no such h, we immediately return $\perp$.

By a union bound, all mH polynomial identity testings are correct with probability at least $9/10$. In this case, if $f = g$, then the algorithm outputs $\perp$ in the first stage. If $f \neq g$, by induction on i, we can show that for every $i \in [m]$, $w_{\leq i}$ is the lexicographically smallest element from H^m such that $f(w_{\leq i}, \cdot)$ and $g(w_{\leq i}, \cdot)$ are distinct, which implies that the output w is also the lexicographically smallest element w in H^m such that $f(w) \neq g(w)$. □

[28] Since both f and g have max degree at most Δ, their values are completely determined by their restrictions on H^m. Hence, if $f \neq g$, then such w must exist.

[29] Note that these two polynomials have total degree at most $m \cdot \Delta < p/2$. Hence, if they are different, their values on a random element from $\mathbb{F}_p^{m-i}$ are different with probability at least $1/2$. Hence the desired error level can be achieved by sampling $O(\log m + \log \Delta)$ random points from $\mathbb{F}^{m-i}$ and checking whether $f(w_{<i}, h, \cdot)$ and $g(w_{<i}, h, \cdot)$ have the same values.

7.5.2 Encoded History and $\mathsf{S}_2\mathsf{BPP}$ Verification

Next, we define the following encoded history.

Definition 7.5.4. Let $C\colon \{0,1\}^n \to \{0,1\}^{2n}$ be a circuit, and $f \in \{0,1\}^T$ be a "hard truth table" in the sense that $f \notin \mathrm{Range}(\mathsf{GGM}_T[C])$. Let k, $(i_\star, j_\star)$, and $\{v_{i,j}\}_{i,j}$ be defined as in Algorithm 7.3.1. Let S be the concatenation of $\mathsf{enc}(v_{i,j})$ for every $i \in \{0,1,\dots,k-1\}$, $j \in \{0,1,\dots,2^i-1\}$, in the reserve lexicographical order of (i,j), padded with zeros at the end to length exactly $5T$. (Recall that $\mathsf{enc}(v_{i,j})$ was defined in Definition 7.3.6.) Let p be the smallest prime that is at least $20 \cdot \log^5 T$, and m be the smallest integer such that $(\log T)^m \geq 5 \cdot T$.

The *encoded computational history* of $\mathsf{Jeřábek–Korten}(C, f)$, denoted as

$$\widetilde{\mathsf{History}}(C, f),$$

consists of $(i_\star, j_\star)$, concatenated with $\mathsf{BRM}_{\mathbb{F}_p, \log T, m}(S)$.

The length of the encoded history is at most

$$\left\lceil \log(40 \cdot \log^5 T) \right\rceil \cdot (40 \cdot \log^5 T)^{\log(5T)/\log\log T + 1} + 2\log T \leq T^6$$

for all sufficiently large T, and for convenience we always pad zeros at the end so that its length becomes exactly T^6.[30]

Recall that the original computational history $\mathsf{History}(C, f)$ is simply the concatenation of $(i_\star, j_\star)$ and S. In the encoded version, we encode its S part by the Reed–Muller code instead. In the rest of this section, when we say history, we always mean the encoded history $\widetilde{\mathsf{History}}(C, f)$ instead of the vanilla history $\mathsf{History}(C, f)$.

We need the following lemma.

Lemma 7.5.5. *Let $n, T \in \mathbb{N}$ be such that $\log T \leq n \leq T$. Let $C\colon \{0,1\}^n \to \{0,1\}^{2n}$ be a circuit and $f \in \{0,1\}^T \setminus \mathrm{Range}(\mathsf{GGM}_T[C])$. Let $h := \widetilde{\mathsf{History}}(C, f)$ and $z := \mathsf{Jeřábek–Korten}(C, f)$.*

1. **(history contains input/output)** *There is a $\mathrm{poly}(\log T)$-time oracle algorithm in and an $O(n)$-time oracle algorithm Output, both of which have input parameters T, n and take a string $\tilde{h} \in \{0,1\}^{T^6}$ as oracle, such that the following hold:*

   (a) *$\mathsf{in}_{T,n}$ makes a single query to its oracle; when given h as the oracle, $\mathsf{in}_{T,n}$ takes an additional input $i \in \{0,1,\dots,T^6-1\}$ and outputs f_i.*

   (b) *$\mathsf{Output}_{T,n}$ makes at most $4n$ queries to its oracle; when given h as the oracle, $\mathsf{Output}_{T,n}$ outputs $z = \mathsf{Jeřábek–Korten}(C, f)$.*

2. **($\mathsf{S}_2\mathsf{BPP}$ verification of the history)** *There is a randomised oracle algorithm V with input parameters T, n such that the following hold:*

   (a) *V takes strings $\tilde{f} \in \{0,1\}^T, \pi_1, \pi_2 \in \{0,1\}^{T^6}$ as oracles, the circuit C, an integer $i \in [T^6]$, and $\varepsilon \in (0,1)$ as input, and runs in $\mathrm{poly}(n, \log \varepsilon^{-1})$ time.*

[30] For simplicity, even for T such that the length of the encoded history is longer than T^6, we will pretend its length is exactly T^6 throughout this section. This does not affect the analysis in our main theorem Theorem 7.5.7 since there we only care about sufficiently large T.

(b) For every $\pi \in \{0,1\}^{T^6}$ *and every* $i \in \{0,1,\dots,T^6-1\}$*, we have that*

$$\Pr\left[V_{T,n}^{f,\pi,h}(C,i,\varepsilon) = h_i\right] \geq 1-\varepsilon \quad \textit{and} \quad \Pr\left[V_{T,n}^{f,h,\pi}(C,i,\varepsilon) = h_i\right] \geq 1-\varepsilon.$$

Proof. Again, the algorithms $\mathsf{in}_{T,n}$ and $\mathsf{Output}_{T,n}$ can be constructed in a straightforward way.[31] So we focus on the construction of V. Let $p, m, k \in \mathbb{N}$ be as in Definition 7.5.4. We also set $\mathbb{F} = \mathbb{F}_p$ and $\Delta = \log T$ in the rest of the proof.

Our V always first *selects* one of the oracles π_1 and π_2 (say π_μ for $\mu \in \{1,2\}$), and then outputs $\pi_\mu(i)$. Hence, in the following, we say that V selects π_μ to mean that V outputs $\pi_\mu(i)$ and terminates. Given π_1 and π_2, let $g_1, g_2 \colon \mathbb{F}^m \to \mathbb{F}$ be the (potential) RM codewords encoded in π_1 and π_2, respectively.[32] From now on, we will assume that i points to an entry in the encoded history g_1 or g_2 instead of the encoded pair of integers $(i_\star, j_\star)$. We will discuss the other case at the end of the proof.

Low-max-degree test and self-correction. Let c_1 be a large enough constant, we first run $\mathsf{LDT}^{g_1}(p,m,\Delta-1)$ and $\mathsf{LDT}^{g_2}(p,m,\Delta-1)$ for c_1 times. Recall that $p \geq 20 \cdot \log^5 T$, $m = \lceil \log(5T)/\log\log T \rceil$, and $\Delta = \log T$. It follows that $p \geq 20 \cdot ((\Delta-1)+1)^2 \cdot m^2$, which satisfies the condition of Lemma 7.5.2. We also note that $3m^2 \cdot ((\Delta-1)+1)/p < 1/4$. Hence, by Lemma 7.5.2, if g_1 is $1/4$-far from all polynomials with maximum individual degree at most $\Delta-1$, then $\mathsf{LDT}^{g_1}(p,m,\Delta-1)$ rejects with probability $2/3$, and similarly for g_2.

Now, if any of the runs on $\mathsf{LDT}^{g_1}(p,m,\Delta-1)$ rejects, V selects π_2, and if any of the runs on $\mathsf{LDT}^{g_2}(p,m,\Delta-1)$ rejects, V selects π_1.[33] In other words, V first *disqualifies* the oracles that do not pass the low-max-degree test. We set c_1 to be large enough so that conditioning on the event that V does not terminate yet, with probability at least 0.99, both g_1 and g_2 are $1/4$-close to polynomials $\widetilde{g}_1 \colon \mathbb{F}_p^m \to \mathbb{F}$ and $\widetilde{g}_2 \colon \mathbb{F}_p^m \to \mathbb{F}$, respectively, where $\deg_{\max}(\widetilde{g}_1)$ and $\deg_{\max}(\widetilde{g}_2)$ are at most $\Delta-1$.

We can then use $\mathsf{PCorr}^{g_1}(p,m,m\cdot(\Delta-1),\cdot)$ and $\mathsf{PCorr}^{g_2}(p,m,m\cdot(\Delta-1),\cdot)$ to access the polynomials $\widetilde{g}_1$ and $\widetilde{g}_2$. (Note that $m\cdot(\Delta-1) < p/3$, which satisfies the condition of Lemma 7.5.1). We repeat them each $O(\log T + \log m)$ times to make sure that on a single invocation, they return the correct values of $\widetilde{g}_1$ and $\widetilde{g}_2$ respectively with probability at least $1 - 1/(mT)^{c_2}$ for a sufficiently large constant c_2. By Lemma 7.5.1, each call to $\mathsf{PCorr}^{g_1}(p,m,m\cdot(\Delta-1),\cdot)$ or $\mathsf{PCorr}^{g_2}(p,m,m\cdot(\Delta-1),\cdot)$ takes $\mathrm{polylog}(T)$ time.

Selecting the better polynomial. From now on, we **refine** what it means when V selects π_μ: now it means that V outputs the bit corresponding to i in $\widetilde{g}_\mu$ (recall that we are assuming that i points to an entry in the encoded history g_1 or g_2).

Let $\{v^1_{i,j}\}$ and $\{v^2_{i,j}\}$ be the encoded histories in $\widetilde{g}_1$ and $\widetilde{g}_2$. Then V uses $\mathsf{Comp}^{\widetilde{g}_1,\widetilde{g}_2}(p,m,\Delta-1)$ to find the lexicographically largest (i',j') such that $v^1_{i',j'} \neq v^2_{i',j'}$.[34] This requires at most $\mathrm{poly}(m\cdot\Delta)$ queries to both $\widetilde{g}_1$ and $\widetilde{g}_2$. By making c_2 large enough, we know that Comp operates

[31] To see that $\mathsf{Output}_{T,n}$ makes at most $4n$ queries: Note that Output first reads the pair $(i_\star, j_\star)$ from h, and then reads two corresponding blocks from $v_{i,j}$ encoded in h. In total, it reads at most $2\log T + 2n \leq 4n$ bits from h.

[32] Technically π_1 and π_2 are supposed to contain the RM codewords concatenated with $\mathsf{Enc}_{\mathbb{F}_p} \colon \mathbb{F}_p \to \{0,1\}^{\lceil \log p \rceil}$.

[33] As a minor detail, if both g_1 and g_2 are rejected by some runs, V selects π_2.

[34] Recall that the $\{v_{i,j}\}$ is encoded in the reverse lexicographic order (Definition 7.5.4).

correctly with probability at least 0.8. By operating correctly, we mean that (1) if $\widetilde{g}_1 \neq \widetilde{g}_2$, Comp finds the correct (i', j') and (2) If $\widetilde{g}_1 = \widetilde{g}_2$, Comp returns $\perp$.[35]

In what follows, we assume that Comp operates correctly. If Comp returns $\perp$, then V simply selects π_1. Otherwise, there are several cases:

1. $i' = k$. In this case, $\widetilde{g}_1$ and $\widetilde{g}_2$ disagree on their leaf values, which intend to encode f. V queries f to figure out which one has the correct value, and selects the corresponding oracle. (Note that at most one of them can be consistent with f. If none of them are consistent, then V selects π_1.)

   From now on, assume $i' < k$ and set $\alpha = v^1_{i'+1,2j'} \circ v^1_{i'+1,2j'+1}$. Note that by the definition of (i', j'), it holds that $\alpha = v^2_{i'+1,2j'} \circ v^2_{i'+1,2j'+1}$ as well.

2. $i' < k$ and both $v^1_{i',j'}$ and $v^2_{i',j'}$ are not $\perp$. In this case, V first checks whether both of them are in $C^{-1}(\alpha)$ (it can be checked by testing whether $C(v^1_{i',j'}) = \alpha$ and $C(v^2_{i',j'}) = \alpha$). If only one of them is contained in $C^{-1}(\alpha)$, V selects the corresponding oracle. If none of them are contained, V selects π_1. Finally, if both are contained in $C^{-1}(\alpha)$, V checks which one is lexicographically smaller, and selects the corresponding oracle.

3. $i' < k$, and one of the $v^1_{i',j'}$ and $v^2_{i',j'}$ is $\perp$. Say that $v^b_{i',j'} = \perp$ for some $b \in \{1, 2\}$, and denote $\bar{b} := 3 - b$ as the index of the other proof. In this case, let $(i_\circ, j_\circ)$ denote the predecessor of (i', j') in terms of the reverse lexicographical order (that is, the smallest pair that is lexicographically greater than (i', j')). Since Comp operates correctly, we have that $v^1_{i_\circ,j_\circ} = v^2_{i_\circ,j_\circ}$. If $v^1_{i_\circ,j_\circ} = \perp$, then $\pi_{\bar{b}}$ has to be incorrect (since by Definition 7.3.6, $\perp$'s form a contiguous suffix of the history), and V selects π_b. Otherwise, if $v^{\bar{b}}_{i',j'} \in C^{-1}(\alpha)$, then π_b is incorrect (as it claims that $C^{-1}(\alpha) = \varnothing$), and V selects $\pi_{\bar{b}}$. Otherwise, V selects π_b.

Analysis. Now we show that $\Pr\left[V^{f,h,\pi}_{T,n}(i) = h(i)\right] \geq 2/3$. (One can symmetrically prove that $\Pr\left[V^{f,\pi,h}_{T,n}(i) = h(i)\right] \geq 2/3$.) To get the desired ε error probability, one can simply repeat the above procedure $O(\log 1/\varepsilon)$ times and output the majority.

First, by Lemma 7.5.2, $\mathsf{LDT}^{g_1}(p, m, \Delta - 1)$ passes with probability 1. If some of the runs of $\mathsf{LDT}^{g_2}(p, m, \Delta - 1)$ rejects, then V selects h. Otherwise, we know that with probability at least 0.99, $\mathsf{PCorr}^{g_1}(p, m, m \cdot (\Delta - 1), \cdot)$ and $\mathsf{PCorr}^{g_2}(p, m, m \cdot (\Delta - 1), \cdot)$ provide access to polynomials $\widetilde{g}_1$ and $\widetilde{g}_2$ with maximum individual degree at most $\Delta - 1$, where $\widetilde{g}_1$ encodes the correct history values $\{v_{i,j}\}_{i,j}$ of $\mathsf{Jeřábek–Korten}(C, f)$.

Then, assuming Comp operates correctly (which happens with probability at least 0.8), if $\widetilde{g}_1 = \widetilde{g}_2$, then the selection of V does not matter. Now we assume $\widetilde{g}_1 \neq \widetilde{g}_2$.

We will verify that in all three cases above, h (as the first oracle) is selected by V. In the first case, by definition, all leaf values in h are consistent with f, and hence h is selected. In the second case, since h contains the correct history values, we know that $v^1_{i',j'}$ must be the smallest

[35] From Lemma 7.5.3, $\mathsf{Comp}^{\widetilde{g}_1,\widetilde{g}_2}(p, m, \Delta - 1)$ itself operates correctly with probability at least 0.9. But the access to $\widetilde{g}_1$ (similarly to $\widetilde{g}_2$) is provided by $\mathsf{PCorr}^{g_1}(p, m, m \cdot (\Delta - 1), \cdot)$, which may err with probability at most $1/(mT)^{c_2}$. Hence, we also need to take a union bound over all the bad events that a query from Comp to $\widetilde{g}_1$ or $\widetilde{g}_2$ is incorrectly answered.

element from $C^{-1}(\alpha)$, so again h is selected. In the last case: (1) if $v^1_{i_\circ,j_\circ} = \perp$, then $v^1_{i',j'}$ has to be $\perp$ as well, thus h is selected; (2) if $v^1_{i_\circ,j_\circ} \neq \perp$ and $v^1_{i',j'} = \perp$, then $C^{-1}(\alpha) = \varnothing$, and since the other proof π claims some element $v^2_{i',j'} \in C^{-1}(\alpha)$, h is selected; and (3) if $v^1_{i_\circ,j_\circ} \neq \perp$ and $v^1_{i',j'} \neq \perp$, then π claims that $C^{-1}(\alpha) = \varnothing$ and we can check that $v^1_{i',j'} \in C^{-1}(\alpha)$, therefore h is selected as well.

The remaining case: i points to the location of $(i_\star, j_\star)$. In this case, V still runs the algorithm described above to make a selection. Indeed, if Comp does not return $\perp$, V operates exactly the same. But when Comp returns $\perp$, V cannot simply select π_1 since we need to make sure that V selects the oracle corresponding to h (it can be either π_1 or π_2). Hence, in this case, V first reads $(i^1_\star, j^1_\star)$ and $(i^2_\star, j^2_\star)$ from π_1 and π_2. If they are the same, V simply selects π_1. Otherwise, for $b \in [2]$, V checks whether $v^b_{i^b_\star,j^b_\star} = \perp$, and select the one that satisfies this condition. (If none of the $v^b_{i^b_\star,j^b_\star}$ are, then V selects π_1). If both of $v^b_{i^b_\star,j^b_\star}$ are $\perp$, V selects the $\mu \in [2]$ such that $(i^\mu_\star, j^\mu_\star)$ is larger.

Now, we can verify that $V^{f,h,\pi}_{T,n}$ selects h with high probability as well. (To see this, note that in the correct history, $(i_\star, j_\star)$ points to the lexicographically largest all-zero block.)

Finally, the running time bound follows directly from the description of V. □

A remark on relativisation

Perhaps surprisingly, although Lemma 7.5.5 heavily relies on arithmetisation tools such as Reed–Muller encoding and low-degree tests, it in fact also relativises. To see this, the crucial observation is that, similarly to Lemma 7.3.7, the verifier V from Lemma 7.5.5 only needs *black-box access* to the input circuit C, meaning that it only needs to evaluate C on certain chosen inputs. Hence, when C is actually an oracle circuit $C^{\mathcal{O}}$ for some arbitrary oracle $\mathcal{O}$, the only modification we need is that V now also takes $\mathcal{O}$ as an oracle.

Remark 7.5.6. Definition 7.5.4 and Lemma 7.5.5 *relativise*, in the sense that if C is an oracle circuit $C^{\mathcal{O}}$ for some arbitrary oracle, Definition 7.5.4 needs no modification since Definition 7.3.6 relativises, and Lemma 7.5.5 holds with the only modification that V now also needs to take $\mathcal{O}$ as an oracle (since it needs to evaluate C).

Indeed, the remark above might sound strange at first glance: arguments that involve PCPs often do not *relativise*, and the encoded history $\widetilde{\mathsf{History}}(C, f)$ looks similar to a PCP since it enables V to perform a probabilistic local verification. However, a closer inspection reveals a key difference: the circuit C is always treated as a black box—both in the construction of history (Definition 7.3.6) and in the construction of the encoded history (Definition 7.5.4). That is, the arithmetisation in the encoded history *does not arithmetise* the circuit C itself.

7.5.3 Lower Bounds for $\mathsf{S_2E}$

Let $\{C_n : \{0,1\}^n \to \{0,1\}^{2n}\}$ be a P-uniform family of circuits. We show that there is a single-valued $\mathsf{FS_2P}$ algorithm $\mathcal{A}$ such that for infinitely many $n \in \mathbb{N}$, on input 1^n, $\mathcal{A}(1^n)$ outputs a canonical string that is outside the range of C_n.

Theorem 7.5.7. *Let $\{C_n\colon \{0,1\}^n \to \{0,1\}^{2n}\}_{n\in\mathbb{N}}$ be a P-uniform family of circuits. There is a sequence of valid outputs $\{y_n \in \{0,1\}^{2n} \setminus \mathrm{Range}(C_n)\}_{n\in\mathbb{N}}$ and a single-valued $\mathsf{FS_2P}$ algorithm A with one bit of advice, such that for infinitely many $n \in \mathbb{N}$, $A(1^n)$ outputs y_n.*

Proof. Our proof proceeds similarly to the proof of the previous Theorem 7.4.1. We will follow the same notation.

Notation. Let $n^{(1)}$ be a large enough power of 2, $n^{(\ell)} = 2^{2^{n^{(\ell-1)}}}$ for each integer $\ell > 1$. Let $n_0^{(\ell)} = n^{(\ell)}$ and $t^{(\ell)} = O\left(\log n_0^{(\ell)}\right)$ be parameters that we set later. For each $1 \le i \le t^{(\ell)}$, let $n_i^{(\ell)} := \left(n_{i-1}^{(\ell)}\right)^{10}$. To show our algorithm A works on infinitely many input lengths, we will show that for every $\ell \in \mathbb{N}$, there is an input length $n_i^{(\ell)}$ for some $i \in \left[t^{(\ell)}\right]$ such that A works.

Fix $\ell \in \mathbb{N}$. From now on, for convenience, we will use n_i and t to denote $n_i^{(\ell)}$ and $t^{(\ell)}$, respectively.

Specifying T_i and f_i. For each input length n_i, we will specify a parameter $T_i \in \mathbb{N}$ and a string $f_i \in \{0,1\}^{T_i}$. Our win-win analysis is based on whether $f_i \in \mathrm{Range}(\mathsf{GGM}_{T_i}[C_{n_i}])$ for each $i \in \{0,1,\dots,t\}$.

Let $T_0 := 2^{2n_0} \cdot 2n_0$ and f_0 be the concatenation of all length-$2n_0$ strings (which has length T_0). From Fact 7.3.4, we have that $f_0 \notin \mathrm{Range}(\mathsf{GGM}_{T_0}[C_{n_0}])$. For every $i \in [t]$, we define

$$f_i = \widetilde{\mathsf{History}}(C_{n_{i-1}}, f_{i-1}).$$

From Definition 7.5.4, this also means that we have set $T_i = T_{i-1}^6$ for every $i \in [t]$.

Let t be the first integer such that $T_{t+1} \le n_{t+1}$. Note that we have $T_i = (T_0)^{6^i} \le 2^{3n_0 \cdot 6^i}$ and $n_i = (n_0)^{10^i} = 2^{\log n_0 \cdot 10^i}$. Hence, we have that $t \le O(\log n_0)$. (Also note that $n_t^{(\ell)} < n_0^{(\ell+1)}$.)

Description of our $\mathsf{FS_2P}$ algorithm A. Now, let $k \in \{0,1,\dots,t\}$ be the largest integer such that $f_k \notin \mathrm{Range}(\mathsf{GGM}_{T_k}[C_{n_k}])$. Since $f_0 \notin \mathrm{Range}(\mathsf{GGM}_{T_0}[C_{n_0}])$, such a k must exist. Let $z := \mathsf{Jeřábek\text{–}Korten}(C_{n_k}, f_k)$, it follows from Lemma 7.3.3 that z is not in the range of C_{n_k} (i.e., $z \in \{0,1\}^{2n_k} \setminus \mathrm{Range}(C_{n_k})$). Our single-valued $\mathsf{FS_2P}$ algorithm A computes z on input 1^{n_k} (see Definition 7.2.2).

We will first construct an $\mathsf{S_2BPP}$ verifier V that computes z in polynomial time on input 1^{n_k}, and then use the fact that all $\mathsf{S_2BPP}$ verifiers can be turned into equivalent $\mathsf{S_2P}$ verifiers with a polynomial-time blow-up [Can96, RS98], from which we can obtain the desired verifier V_A for A.

Description of an $\mathsf{S_2BPP}$ verifier V computing z. Formally, V is a randomised polynomial-time algorithm that takes 1^{n_k} and two witnesses $\pi_1, \pi_2 \in \{0,1\}^{n_{k+1}}$ as input, and we aim to establish the following:

There exists $\omega \in \{0,1\}^{n_{k+1}}$ such that for every $\pi \in \{0,1\}^{n_{k+1}}$, we have

$$\Pr[V(1^{n_k}, \omega, \pi) = z] \ge 2/3 \qquad \text{and} \qquad \Pr[V(1^{n_k}, \pi, \omega) = z] \ge 2/3,$$

where the probabilities are over the internal randomness of V.

In more detail, if $k < t$, then V treats π_1 and π_2 as inputs to the circuit $\mathsf{GGM}_{T_{k+1}}[C_{n_{k+1}}]$, and let

$$\hat{f}_{k+1} := \mathsf{GGM}_{T_{k+1}}[C_{n_{k+1}}](\pi_1) \quad \text{and} \quad \hat{g}_{k+1} := \mathsf{GGM}_{T_{k+1}}[C_{n_{k+1}}](\pi_2).$$

Here, the lengths of π_1 and π_2 are $\ell := n_{k+1} \leq \mathrm{poly}(n_k)$. If $k = t$, then V defines $\hat{f}_{k+1} := \pi_1$, $\hat{g}_{k+1} := \pi_2$, and their lengths are $\ell := T_{t+1} \leq n_{k+1} \leq \mathrm{poly}(n_k)$. It is intended that one of the $\hat{f}_{k+1}$ and $\hat{g}_{k+1}$ is $f_{k+1} = \widetilde{\mathsf{History}}(C_{n_k}, f_k)$ (V needs to figure out which one).

We now specify the intended proof $\omega \in \{0,1\}^{n_{k+1}}$. When $k < t$, since f_{k+1} is in the range of $\mathsf{GGM}_{T_{k+1}}[C_{n_{k+1}}]$, we can set ω so that $\mathsf{GGM}_{T_{k+1}}[C_{n_{k+1}}](\omega) = f_{k+1}$. When $k = t$, we simply set $\omega = f_{k+1}$.

Note that Lemma 7.3.2 provides us "random access" to the (potentially very long) strings $\hat{f}_{k+1}$ and $\hat{g}_{k+1}$: (take $\hat{f}_{k+1}$ as an example) given π_1 and $j \in \{0, 1, \ldots, T_{k+1} - 1\}$, one can compute the j-th bit of $\hat{f}_{k+1}$ in $\mathrm{poly}(n_k)$ time. Also recall from Lemma 7.5.5 that for each i, $f_{i+1} = \widetilde{\mathsf{History}}(C_{n_i}, f_i)$ contains the string f_i, which can be retrieved by the oracle algorithm in described in Item 1 of Lemma 7.5.5. Therefore, for each i from k downto 1, we can recursively define $\hat{f}_i$ such that $(\hat{f}_i)_j = \mathsf{in}^{\hat{f}_{i+1}}_{T_i,n_i}(j)$ (similarly for $\hat{g}_i$). We also define $\hat{f}_0$ and $\hat{g}_0$ to be the concatenation of all length-$(2n_0)$ strings in the lexicographical order, so $\hat{f}_0 = \hat{g}_0 = f_0$.

Applying the algorithm in recursively, we obtain two algorithms F and G (depending on π_1 and π_2, respectively) that given $i \in \{0, 1, \ldots, k+1\}$ and $j \in \{0, 1, \ldots, T_i - 1\}$, output the j-th bit of $\hat{f}_i$ or $\hat{g}_i$, respectively. Since in only makes one oracle query, these algorithms run in $\mathrm{poly}(n_k)$ time.

We are now ready to formally construct V. We first recursively define a series of procedures $V_0, \ldots, V_{k+1}$, where each V_i takes an input j and outputs (with high probability) the j-th bit of f_i. Let V_0 be the simple algorithm that, on input j, computes the j-th bit of f_0. For every $i \in [k+1]$ and for some $\varepsilon_i \in [0, 1)$ to be specified later, we define

$$V_i(\alpha) := \mathsf{Select}^{V_{i-1},\hat{f}_i,\hat{g}_i}_{T_{i-1},n_{i-1}}(C_{n_{i-1}}, \alpha, \varepsilon_i),$$

where Select is the algorithm in Item 2 of Lemma 7.5.5. We note that since V_{i-1} is a randomised algorithm, when V_i calls V_{i-1}, it also draws *independent* random coins used by the execution of V_{i-1}. Moreover, all calls to $\hat{f}_i$ and $\hat{g}_i$ in V_i can be simulated by calling our algorithms F and G. Jumping ahead, we remark that V_i is supposed to compute f_i when at least one of $\hat{f}_i$ or $\hat{g}_i$ is f_i. We then set

$$V(1^{n_k}, \pi_1, \pi_2) := \mathsf{Output}^{V_{k+1}}_{T_k,n_k}$$

(note that V_{k+1} is defined from $\hat{f}_{k+1}$ and $\hat{g}_{k+1}$, which are in turn constructed from π_1 and π_2), where $\mathsf{Output}_{T_k,n_k}$ is the algorithm from Item 1 of Lemma 7.5.5.

Correctness of V. Let $\tau \in \mathbb{N}$ be a large constant such that $\mathsf{Select}_{T,n}$ runs in $(n \cdot \log 1/\varepsilon)^\tau$ time. In particular, on any input α, $\mathsf{Select}^{V_{i-1},\hat{f}_i,\hat{g}_i}_{T_{i-1},n_{i-1}}(C_{n_{i-1}}, \alpha, \varepsilon_i)$ makes at most $(n_{i-1} \cdot \log 1/\varepsilon_i)^\tau$ many queries to V_{i-1}.

We say $\mathsf{Select}^{f,\pi_1,\pi_2}_{T,n}(C, \alpha, \varepsilon_i)$ makes an error if the following statements is true (here $h =$

$\widetilde{\mathsf{History}}(C, f)$ from Lemma 7.5.5):[36]

$$[\pi_1 = h \quad \mathsf{OR} \quad \pi_2 = h] \quad \mathsf{AND} \quad \left[\mathsf{Select}_{T,n}^{f,\pi_1,\pi_2}(C_{n_{i-1}}, \alpha, \varepsilon_i) \neq h_\alpha\right].$$

Similarly, we say that $\mathsf{Select}_{T_{i-1},n_{i-1}}^{V_{i-1},\hat{f}_i,\hat{g}_i}(C_{n_{i-1}}, \alpha, \varepsilon_i)$ makes an error if either (1) one of the queries to V_{i-1} are incorrectly answered (i.e., the answer is not consistent with f_{i-1}) or (2) all queries are correctly answered but $\mathsf{Select}_{T_{i-1},n_{i-1}}^{f_{i-1},\hat{f}_i,\hat{g}_i}(C_{n_{i-1}}, \alpha, \varepsilon_i)$ makes an error. Note that (2) happens with probability at most ε_i from Item 2 of Lemma 7.5.5.

Now we are ready to specify the parameter ε_i. We set $\varepsilon_{k+1} = 1/(100 \cdot n_{k+1})$, and for every $i \in \{0, 1, \ldots, k\}$, we set

$$\varepsilon_i = \frac{\varepsilon_{i+1}}{4 \cdot (n_i \cdot \log 1/\varepsilon_{i+1})^\tau}.$$

To show the correctness of V, we prove the following claim by induction.

Claim 7.5.8. *Assume either $\hat{f}_{k+1} = f_{k+1}$ or $\hat{g}_{k+1} = f_{k+1}$. For every $i \in \{0, 1, \ldots, k+1\}$ and $\alpha \in [|f_i|]$, $V_i(\alpha)$ outputs $f_i(\alpha)$ with probability at least $1 - 2\varepsilon_i$.*

Proof. The claim certainly holds for V_0. Now, for $i \in [k+1]$, assuming it holds for V_{i-1}, it follows that $\mathsf{Select}_{T_{i-1},n_{i-1}}^{V_{i-1},\hat{f}_i,\hat{g}_i}(C_{n_{i-1}}, \alpha, \varepsilon_i)$ makes an error with probability at most

$$\varepsilon_i + (n_{i-1} \cdot \log 1/\varepsilon_i)^\tau \cdot 2\varepsilon_{i-1} \leq 2\varepsilon_i.$$

By the definition of making an error and our assumption that either $\hat{f}_{k+1} = f_{k+1}$ or $\hat{g}_{k+1} = f_{k+1}$ (from which we know either $\hat{f}_i = f_i$ or $\hat{g}_i = f_i$), it follows that $V_i(\alpha)$ outputs $f_i(\alpha)$ with probability at least $1 - 2\varepsilon_i$. ◇

Note that $\mathsf{Output}_{T_k,n_k}^{V_{k+1}}$ makes at most $4n_k$ queries to V_{k+1}. It follows from Claim 7.5.8 that when either $\hat{f}_{k+1} = f_{k+1}$ or $\hat{g}_{k+1} = f_{k+1}$, we have that $V(1^{n_k}, \pi_1, \pi_2)$ outputs z with probability at least $1 - (4n_k) \cdot 1/(100n_{k+1}) \geq 2/3$. The correctness of V then follows from our choice of ω.

Running time of V. Finally, we analyse the running time of V, for which we first need to bound $\log \varepsilon_i^{-1}$. First, we have

$$\log \varepsilon_{k+1}^{-1} = \log n_{k+1} + \log 100.$$

By our definition of ε_i and the fact that τ is a constant, we have

$$\begin{aligned}\log \varepsilon_i^{-1} &= \log \varepsilon_{i+1}^{-1} + \log 4 + \tau \cdot \left(\log n_i + \log \log \varepsilon_{i+1}^{-1}\right) \\ &\leq 2 \log \varepsilon_{i+1}^{-1} + O(\log n_i).\end{aligned}$$

Expanding the above and noting that $k \leq t \leq O(\log n_0)$, for every $i \in [k+1]$ we have that

$$\log \varepsilon_i^{-1} \leq 2^k \cdot O\left(\sum_{\ell=0}^{k} \log n_\ell\right) \leq \mathrm{poly}(n_0) \cdot \log n_k.$$

[36] The condition below only applies when at least one of π_1 and π_2 is h. If neither of them are h, then Select by definition never errs.

Now we are ready to bound the running time of the V_i. First V_0 runs in $T_0 = \mathrm{poly}(n_0)$ time. For every $i \in [k+1]$, by the definition of V_i, we know that V_i runs in time

$$T_i = O\Big((n_{i-1} \cdot \log 1/\varepsilon_i)^\tau\Big) \cdot (T_{i-1} + n_k^\beta + 1),$$

where β is a sufficiently large constant and n_k^β bounds the running time of answering each query $\mathsf{Select}^{V_{i-1},\hat{f}_i,\hat{g}_i}_{T_{i-1},n_{i-1}}(C_{n_{i-1}}, \alpha, \varepsilon_i)$ makes to $\hat{f}_i$ or $\hat{g}_i$, by running F or G, respectively.

Expanding out the bound for T_k, we know that V_{k+1} runs in time

$$2^{O(k)} \cdot (\mathrm{poly}(n_0) \cdot \log n_k)^{O(k\cdot\tau)} \cdot n_k^\beta \cdot \prod_{i=1}^{k+1} n_{i-1}^\tau.$$

Since $n_k = n_0^{10^k}$ and $k \le O(\log n_0)$, the above can be bounded by $\mathrm{poly}(n_k)$. This also implies that V runs in $\mathrm{poly}(n_k)$ time as well, which completes the analysis of the $\mathsf{S_2BPP}$ verifier V.

Derandomisation of the $\mathsf{S_2BPP}$ verifier V into the desired $\mathsf{S_2P}$ verifier V_A. Finally, we use the underlying proof technique of $\mathsf{S_2BPP} = \mathsf{S_2P}$ [Can96, RS98] to derandomise V into a deterministic $\mathsf{S_2P}$ verifier V_A that outputs z.

By repeating V $\mathrm{poly}(n_k)$ times and outputting the majority among all the outputs, we can obtain a new $\mathsf{S_2BPP}$ verifier $\widetilde{V}$ such that

- There exists $\omega \in \{0,1\}^{n_{k+1}}$ such that for every $\pi \in \{0,1\}^{n_{k+1}}$, we have

$$\Pr[\widetilde{V}(1^{n_k}, \omega, \pi) = z] \ge 1 - 2^{-n_k} \qquad \text{and} \qquad \Pr[\widetilde{V}(1^{n_k}, \pi, \omega) = z] \ge 1 - 2^{-n_k}. \tag{7.2}$$

Let $\ell = \mathrm{poly}(n_k)$ be an upper bound on the number of random coins used by $\widetilde{V}$. We also let $m := \mathrm{poly}(\ell, n_{k+1}) \le \mathrm{poly}(n_k)$ and use $\widetilde{V}(1^{n_k}, \pi_1, \pi_2; r)$ to denote the output of $\widetilde{V}$ given randomness r. Now, we define V_A as follows: It takes two vectors $\vec{\pi}_1, \vec{\pi}_2 \in \{0,1\}^{n_{k+1}} \times \big(\{0,1\}^\ell\big)^m$ as proofs. For $\vec{\pi}_1 = (\alpha, u_1, u_2, \ldots, u_m)$ and $\vec{\pi}_2 = (\beta, v_1, v_2, \ldots, v_m)$, V_A outputs the majority of the multi-set

$$\{\widetilde{V}(1^{n_k}, \alpha, \beta; u_i \oplus v_j)\}_{(i,j)\in[m]^2},$$

where $u_i \oplus v_j$ denotes the bit-wise XOR of u_i and v_j (if no strings occur more than $m^2/2$ times in the set above, then V_A simply outputs $\perp$).

We will show there exists $\vec{\omega} = (\gamma, r_1, \ldots, r_m)$ such that for every $\vec{\pi} \in \{0,1\}^{n_{k+1}} \times \big(\{0,1\}^\ell\big)^m$,

$$\Pr[V_A(1^{n_k}, \vec{\omega}, \vec{\pi}) = z] \quad \text{and} \quad \Pr[V_A(1^{n_k}, \vec{\pi}, \vec{\omega}) = z].$$

We first claim that there exist $r_1, \ldots, r_m \in \{0,1\}^\ell$ such that for every $u \in \{0,1\}^\ell$ and for every $\pi \in \{0,1\}^{n_{k+1}}$, it holds that (1) for at least a $2/3$ fraction of $i \in [m]$, we have $\widetilde{V}(1^{n_k}, \omega, \pi; r_i \oplus u) = z$ and (2) for at least a $2/3$ fraction of $i \in [m]$, we have $\widetilde{V}(1^{n_k}, \pi, \omega; r_i \oplus u) = z$.

To see this, for every fixed $u \in \{0,1\}^\ell$ and $\pi \in \{0,1\}^{n_{k+1}}$, by a simple Chernoff bound, the probability, over m independently uniformly drawn $r_1, \ldots, r_m$, that more than a $1/3$ fraction of $i \in [m]$ satisfies $\widetilde{V}(1^{n_k}, \omega, \pi; r_i \oplus u) \ne z$ is at most $2^{-\Omega(m)}$, and the same probability upper bound holds for the corresponding case of $\widetilde{V}(1^{n_k}, \pi, \omega; r_i \oplus u) \ne z$ as well. Our claim then just

follows from a simple union bound over all $u \in \{0,1\}^\ell$ and $\pi \in \{0,1\}^{n_{k+1}}$.

Now, let γ be the proof ω such that the condition (7.2) holds, we set $\vec{\omega} = (\gamma, r_1, \ldots, r_m)$. From our choice of γ and $r_1, \ldots, r_m$, it then follows that for every $v_1, \ldots, v_m \in \{0,1\}^\ell$ and $\pi \in \{0,1\}^{n_{k+1}}$, at least a $2/3$ fraction of $\widetilde{V}(1^{n_k}, \gamma, \pi; r_i \oplus v_j)$ equals z, and similarly for $\widetilde{V}(1^{n_k}, \pi, \gamma; r_i \oplus v_j)$. This completes the proof.

Wrapping up. Finally, we generalise A and V_A to work on all inputs 1^n. On input 1^n, V_A calculates the largest ℓ such that $n^{(\ell)} \leq n$, and also calculates the largest k' such that $n^{(\ell)}_{k'} \leq n$. If $n^{(\ell)}_{k'} \neq n$, then V_A immediately outputs $\perp$ and halts. Otherwise, V_A receives an advice bit indicating whether $k' = k^{(\ell)}$, where $k^{(\ell)}$ is the largest integer such that $f^{(\ell)}_{k^{(\ell)}} \notin \mathrm{Range}(\mathsf{GGM}_{T^{(\ell)}_k}[C_{n^{(\ell)}_k}])$. If this is the case, then V_A runs the verification procedure above; otherwise, it immediately outputs $\perp$ and halts. It is easy to see that V_A runs in $\mathrm{poly}(n)$ time, and is an infinitely-often single-valued $\mathsf{FS_2P}$ algorithm solving the range avoidance problem of $\{C_n\}$. □

Moreover, observe that in the proof of Lemma 7.5.5, all considered input lengths (the $n^{(\ell)}_i$) are indeed powers of 2. So we indeed have the following slightly stronger result.

Corollary 7.5.9. *Let $\{C_n \colon \{0,1\}^n \to \{0,1\}^{2n}\}_{n\in\mathbb{N}}$ be a P-uniform family of circuits. There is a single-valued $\mathsf{FS_2P}$ algorithm A with one bit of advice such that for infinitely many $r \in \mathbb{N}$, letting $n = 2^r$, $A(1^n)$ outputs $y_n \in \{0,1\}^{2n} \setminus \mathrm{Range}(C_n)$.*

We need the following reduction from Korten, which reduces solving range avoidance with one-bit stretch to solving range avoidance with doubling stretch.

Lemma 7.5.10 ([Kor21, Lemma 3])**.** *Let $n \in \mathbb{N}$. There is a polynomial time algorithm A and an $\mathsf{FP}^{\mathsf{NP}}$ algorithm B such that the following hold:*

1. *Given a circuit $C \colon \{0,1\}^n \to \{0,1\}^{n+1}$, $A(C)$ outputs a circuit $D \colon \{0,1\}^n \to \{0,1\}^{2n}$.*
2. *Given any $y \in \{0,1\}^{2n} \setminus \mathrm{Range}(D)$, $B(C, y)$ outputs a string $z \in \{0,1\}^{n+1} \setminus \mathrm{Range}(C)$.*

The following corollary then follows by combining Lemma 7.5.10 and Theorem 7.2.3.

Corollary 7.5.11. *Let $\{C_n \colon \{0,1\}^n \to \{0,1\}^{n+1}\}_{n\in\mathbb{N}}$ be a P-uniform family of circuits. There is a single-valued $\mathsf{FS_2P}$ algorithm A with one bit of advice such that for infinitely many $r \in \mathbb{N}$, letting $n = 2^r$, $A(1^n)$ outputs $y_n \in \{0,1\}^{n+1} \setminus \mathrm{Range}(C_n)$.*

The following corollary follows from Fact 7.2.4 and Corollary 7.5.11.

Corollary 7.5.12. $\mathsf{S_2E}/_1 \not\subset \mathsf{SIZE}[2^n/n]$.

Finally, we also note that by letting C_n be a universal Turing machine mapping n bits to $n+1$ bits in $\mathrm{poly}(n)$ time, we have the following strong lower bounds for $\mathsf{S_2E}/_1$ against non-uniform time complexity classes with maximum advice.

Corollary 7.5.13. *For every $\alpha(n) \geq \omega(1)$ and any constant $k \geq 1$, $\mathsf{S_2E}/_1 \not\subset \mathsf{TIME}[2^{kn}]/_{2^n - \alpha(n)}$.*

From Remark 7.5.6 and noting that the derandomisation of $\mathsf{S_2BPP}$ verifier V to $\mathsf{S_2P}$ verifier A_V also relativises, we can see that all the results above relativise as well.

7.5.4 Infinitely-Often Single-Valued $\mathsf{FS}_2\mathsf{P}$ Algorithms for Arbitrary Range Avoidance

Theorem 7.5.7 and Corollary 7.5.11 only give single-valued $\mathsf{FS}_2\mathsf{P}$ algorithms for solving range avoidance for P-uniform families of circuits. Applying the Jeřábek–Korten reduction (again), we show that it can be strengthened into a single-valued infinitely-often $\mathsf{FS}_2\mathsf{P}$ algorithm solving range avoidance given an arbitrary input circuit.

Theorem 7.5.14. *There is a single-valued $\mathsf{FS}_2\mathsf{P}$ algorithm A with one bit of advice such that for infinitely many $s \in \mathbb{N}$, for all s-size circuits $C\colon \{0,1\}^n \to \{0,1\}^{n+1}$ where $n \leq s$, $A(C)$ outputs $y_C \in \{0,1\}^{n+1} \setminus \operatorname{Range}(C)$.*

Proof Sketch. By Corollary 7.5.11, there is a single-valued $\mathsf{FS}_2\mathsf{P}$ algorithm W with one bit of advice such that for infinitely many $n \in \mathbb{N}$, $W(1^{2^n})$ outputs a string $f_n \in \{0,1\}^{2^n}$ with $\mathsf{SIZE}(f_n) \geq 2^n/n$.

Now we construct our single-valued $\mathsf{FS}_2\mathsf{P}$ algorithm A with one bit of advice as follows: given an s-size circuit $C\colon \{0,1\}^n \to \{0,1\}^{n+1}$ with $n \leq s$ as input; let $m = \lceil \log s^3 \rceil$ and $f_m = W(1^{2^m})$; output $\mathsf{Jeřábek\text{–}Korten}(C, f_m)$. It follows from Theorem 7.2.3 that A is a single-valued $\mathsf{FS}_2\mathsf{P}$ algorithm with one bit of advice (the advice of A is given to W). □

Finally, $\mathsf{S}_2\mathsf{P} \subseteq \mathsf{ZPP}^{\mathsf{NP}}$ [Cai07] implies that every single-valued $\mathsf{FS}_2\mathsf{P}$ algorithm can also be implemented as a single-valued $\mathsf{FZPP}^{\mathsf{NP}}$ algorithm with polynomial overhead. Therefore, the above theorem also implies an infinitely often $\mathsf{FZPP}^{\mathsf{NP}}$ algorithm for range avoidance.

Reminder of Theorem 7.1.5. *There is a single-valued $\mathsf{FZPP}^{\mathsf{NP}}$ algorithm A with one bit of advice such that for infinitely many $s \in \mathbb{N}$, for all s-size circuits $C\colon \{0,1\}^n \to \{0,1\}^{n+1}$ where $n \leq s$, $A(C)$ outputs $y_C \in \{0,1\}^{n+1} \setminus \operatorname{Range}(C)$. That is, for all those s, there is a string $y_C \in \{0,1\}^{n+1} \setminus \operatorname{Range}(C)$ such that $A(C)$ either outputs y_C or $\perp$, and the probability (over the inner randomness of A) that $A(C)$ outputs y_C is at least $2/3$.*

Chapter 8

The Complexity of Avoiding Heavy Elements

8.1 Introduction

Let C be a Boolean circuit sampling a distribution D on N-bit strings. Say that an N-bit string y is δ-heavy in D if y occurs with probability at least δ in D. Assuming that some 2δ-heavy string exists, for $\delta \geq 1/\text{poly}(N)$, how hard is it to find a δ-heavy string given C as input?

We call this natural search problem the *heavy element finding* (Heavy Find) problem. It is not difficult to see that the complexity of Heavy Find is closely related to the complexity of derandomisation. There is a simple randomised polynomial-time algorithm for Heavy Find: we use C to draw $O(N/\delta^2)$ independent samples from D and output the string that occurs with the greatest multiplicity in the multiset of samples. A standard application of Chernoff–Hoeffding bounds shows that assuming that a 2δ-heavy string exists, the output of the algorithm will be a string that is δ-heavy in D with high probability.

Moreover, a deterministic polynomial-time algorithm for Heavy Find implies $\mathsf{BPP} = \mathsf{P}$. Indeed, let M be a probabilistic polynomial-time Turing machine with error bounded by $1/4$ and x be an input to M. We can define a circuit sampler C_x which interprets its input as randomness r for the computation of M on x, outputting 1^N if M accepts on x using randomness r and 0^N otherwise. Observe that if M accepts x, the unique solution to Heavy Find on input C_x with parameter $\delta = 1/3$ is 1^N, and if M rejects x, the unique solution to Heavy Find on input C_x with parameter $1/3$ is 0^N. Thus, a deterministic polynomial-time algorithm for Heavy Find allows us to decide if M accepts x, also in deterministic polynomial time.[1]

We now turn our original question on its head: given C as input, how hard is it to find a string that is *not* δ-heavy? We call this the *heavy element avoidance* (Heavy Avoid) problem. Heavy Avoid is the complementary search problem to Heavy Find: a string $y \in \{0,1\}^N$ is a solution to Heavy Avoid if and only if it is not a solution to Heavy Find. The complexity of Heavy Avoid is the primary focus of this chapter.

[1] Readers who are familiar with derandomisation might already see that the derandomisation also holds for the *promise* version of BPP (prBPP). In fact, it is not hard to show that Heavy Find can be solved in deterministic polynomial time if and only if prBPP collapses to prP, the promise version of P.

Superficially, Heavy Avoid seems to be a much simpler problem to solve than Heavy Find. First, when $\delta > 2^{-N}$, Heavy Avoid is a *total* search problem, i.e., the promise that a non-heavy N-bit string exists is automatically satisfied. In this chapter, we mainly focus on the regime where $\delta \geq 1/\mathrm{poly}(N)$, hence this is always true if N is large enough. Second, there is a trivial algorithm that *list*-solves Heavy Avoid: Since the number of δ-heavy strings is at most $1/\delta$, at least one of the lexicographically first $\lceil 1/\delta \rceil + 1$ strings of length N is guaranteed to be a solution to Heavy Avoid. Third, there is a very efficient randomised algorithm for Heavy Avoid with overwhelming success probability: output a uniformly random string of length N. Note that by the previous observation that the number of δ-heavy strings is at most $1/\delta$, this randomised algorithm fails on at most $1/\delta \leq \mathrm{poly}(N)$ of its random choices.

Our main contribution is to introduce Heavy Avoid as a natural search problem of interest, and show that despite its seeming simplicity, Heavy Avoid has applications to several frontier questions in complexity theory regarding *uniform randomised lower bounds* and *derandomisation*. Indeed, we show that in many settings the existence of algorithms for Heavy Avoid is *equivalent* to a complexity lower bound. The study of Heavy Avoid also illuminates recent *almost-all-inputs-hardness* assumptions in the theory of derandomisation [CT21a], and leads to novel *white-box* reductions in settings where black-box reductions are hard to show.

8.1.1 Results

Our results are twofold.

- First, we present algorithmic characterisations of lower bounds against uniform probabilistic circuits via Heavy Avoid. That is, deterministic algorithms for Heavy Avoid (in certain settings and with certain parameters) are *equivalent* to such lower bounds. In fact, we obtain very general characterisations that hold for classes such as $\mathsf{EXP}, \mathsf{PSPACE}, \mathsf{EXP}^{\mathsf{NP}}$ and NP, against uniform randomised circuit classes such as ACC^0, TC^0, or $\mathsf{SIZE}[\mathrm{poly}]$. This suggests that the analysis of Heavy Avoid could be useful in attacking frontier open questions such as $\mathsf{EXP} \not\subseteq \mathsf{BP\text{-}ACC}^0$ and $\mathsf{EXP}^{\mathsf{NP}} \not\subseteq \mathsf{BPP}$.
- Then, we give applications of Heavy Avoid to derandomisation, including novel *white-box* reductions from promise problems that are hard for prRP or prBPP to Heavy Avoid, as well as connections to "almost-all-inputs-hardness" assumptions that have been explored in recent work on derandomisation.

We consider both *uniform* and *non-uniform* versions of Heavy Avoid. In the uniform version, the search algorithm is given N in unary, and needs to find a δ-light[2] element in D_N, where $\mathcal{D} = \{D_N\}_{N \in \mathbb{N}}$ is an ensemble of distributions over N-bit strings that are sampled by some *uniform* sequence of circuits from a circuit class. Since $\mathcal{D}$ is sampled by a uniform sequence of circuits, we do not need to give the circuit sampler explicitly to the search algorithm—the search algorithm can compute the circuit sampler by itself. In this uniform variant of the problem, fix a parameter $\delta \colon \mathbb{N} \to [0,1]$, $(\mathcal{D}, \delta)$-`Heavy-Avoid` is the problem of finding a $\delta(N)$-light element in D_N, given 1^N as input.

[2] A δ-light element is one that is not δ-heavy.

In the non-uniform variant of the problem, the search algorithm is given as input a circuit sampler C from some circuit class $\mathcal{C}$, and needs to output a δ-light element in the distribution sampled by C.

There are also two kinds of samplability we consider: *implicit* and *explicit*. In the implicit version, our sampler C is Boolean: given randomness r as input together with an index $i \in [N]$, it outputs the i-th bit of the string sampled on randomness r. In this setting, the circuit size is typically less than N. In the explicit version, the circuit C is given randomness r as input and has N output bits: it outputs the string sampled on randomness r. In this setting, the circuit size is at least N, since there are N output bits. Note that when we show an implication from solving Heavy Avoid to proving lower bounds, the implication is *stronger* when we consider implicit solvers, since the algorithmic problem is *easier* to solve for implicit samplers.[3] An implicit solver $C(r, i)$ can easily be converted to an equivalent explicit solver

$$C_{\mathsf{Explicit}}(r) := C(r, 1)C(r, 2) \dots C(r, N).$$

Equivalences Between Complexity Separations and Algorithms for Heavy Avoid

It is a long-standing open question to prove lower bounds against non-uniform circuits – we still have not ruled out the possibility that every language computable in exponential time with an NP oracle ($\mathsf{EXP}^{\mathsf{NP}}$) has polynomial-size circuits. What is more embarrassing is our inability to separate $\mathsf{EXP}^{\mathsf{NP}}$ from BPP (see, e.g., [Wil13b, Wil19] for discussions), despite the belief shared by many researchers that $\mathsf{BPP} = \mathsf{P}$ [NW94, IW97].[4] Moreover, the state of affairs is the same regarding lower bounds against uniform probabilistic circuits from *restricted* circuit classes: for example, it is open whether EXP can be simulated by $\mathsf{DLOGTIME}$-uniform probabilistic ACC^0 circuits or $\mathsf{EXP}^{\mathsf{NP}}$ can be simulated by $\mathsf{DLOGTIME}$-uniform probabilistic TC^0 circuits.[5]

Our first set of results gives *equivalences* between such explicit lower bounds against uniform probabilistic circuits and efficient deterministic algorithms for Heavy Avoid. The equivalences work in a wide variety of settings, for a range of circuit classes including $\mathsf{ACC}^0, \mathsf{TC}^0, \mathsf{NC}^1$ and general Boolean circuits, and for explicit lower bounds in several standard complexity classes of interest such as $\mathsf{EXP}, \mathsf{EXP}^{\mathsf{NP}}, \mathsf{PSPACE}$ and NP. Notably, these results give new *algorithmic characterisations* of uniform lower bound questions by the existence of efficient algorithms for a natural search problem. Thus, they could potentially be useful in attacking frontier open questions such as the EXP vs (uniform probabilistic) ACC^0 question, or the $\mathsf{EXP}^{\mathsf{NP}}$ vs BPP question.

We use $\mathsf{BP}\mathcal{C}$ to denote the set of languages computed by $\mathsf{DLOGTIME}$-uniform probabilistic $\mathcal{C}$-circuits.

Theorem 8.1.1 (Informal)**.** *Let $\mathcal{C}$ be a nice*[6] *class of Boolean circuits. The following equivalences*

[3] We measure the complexity of solving the search problem as a function of N, even in the implicit-sampler setting.

[4] Since BPP is strictly contained in $\mathsf{SIZE}[\mathrm{poly}]$ [Adl78], the open problem of separating $\mathsf{EXP}^{\mathsf{NP}}$ from BPP is *more* embarrassing than separating $\mathsf{EXP}^{\mathsf{NP}}$ from $\mathsf{SIZE}[\mathrm{poly}]$! See also [Wil19, Table 1] for a related perspective.

[5] It follows from $\mathsf{EXP}^{\mathsf{NP}} \not\subseteq \mathsf{ACC}^0$ [Wil14, CLW20], which is a *non-uniform* circuit lower bound, that $\mathsf{EXP}^{\mathsf{NP}}$ cannot be simulated by $\mathsf{DLOGTIME}$-uniform probabilistic ACC^0 circuits. (Note that we do not know how to prove such lower bounds by exploiting the circuit *uniformity* condition.)

[6] In brief, a nice circuit class is one that contains $\mathsf{AC}^0[\oplus]$, is closed under composition, and admits universal circuits for the corresponding class.

hold:

(*i*) $\mathsf{EXP} \not\subseteq \mathsf{BP}\text{-}\mathcal{C}$ *if and only if* $(\mathcal{D}, \delta)$-`Heavy-Avoid` *with* $\delta(N) = 1/\mathrm{polylog}(N)$ *can be solved in deterministic polynomial time on infinitely many input lengths for any* $\mathcal{D}$ *that admits implicit* DLOGTIME*-uniform* $\mathcal{C}$*-samplers of size* $\mathrm{polylog}(N)$.

(*ii*) $\mathsf{EXP}^{\mathsf{NP}} \not\subseteq \mathsf{BP}\text{-}\mathcal{C}$ *if and only if* $(\mathcal{D}, \delta)$-`Heavy-Avoid` *with* $\delta(N) = 1/\mathrm{polylog}(N)$ *can be solved in deterministic polynomial time with an* NP *oracle on infinitely many input lengths for any* $\mathcal{D}$ *that admits implicit* DLOGTIME*-uniform* $\mathcal{C}$*-samplers of size* $\mathrm{polylog}(N)$.

(*iii*) $\mathsf{PSPACE} \not\subseteq \mathsf{BP}\text{-}\mathcal{C}$ *if and only if* $(\mathcal{D}, \delta)$-`Heavy-Avoid` *with* $\delta(N) = 1/\mathrm{polylog}(N)$ *can be solved in deterministic logarithmic space on infinitely many input lengths for any* $\mathcal{D}$ *that admits implicit* DLOGTIME*-uniform* $\mathcal{C}$*-samplers of size* $\mathrm{polylog}(N)$.

(*iv*) $\mathsf{NP} \not\subseteq \mathsf{BP}\text{-}\mathcal{C}$ *if and only if* $(\mathcal{D}, \delta)$-`Heavy-Avoid` *with* $\delta(N) = 1/\mathrm{polylog}(N)$ *can be solved by* DLOGTIME*-uniform unbounded fan-in circuits of quasi-polynomial size and constant depth on infinitely many input lengths for any* $\mathcal{D}$ *that admits implicit* DLOGTIME*-uniform* $\mathcal{C}$*-samplers of size* $\mathrm{polylog}(N)$.

For the PSPACE lower bounds, analogous algorithmic characterisations hold for *almost everywhere* uniform lower bounds and for lower bounds against uniform randomised *subexponential* size circuits. Perhaps interestingly, it follows from our arguments that the existence of efficient algorithms for $(\mathcal{D}, \delta)$-`Heavy-Avoid` in the settings considered in Theorem 8.1.1 is *robust* with respect to the threshold parameter $\delta(N)$: the existence of algorithms for any $\delta(N) = o(1)$ yields the existence of algorithms of similar complexity for $\delta(N) = 1/\mathrm{polylog}(N)$.

Theorem 8.1.1 has direct corollaries that characterise frontier open questions in complexity theory.

Corollary 8.1.2 (Informal)**.** *The following results hold:*

(*i*) $\mathsf{EXP}^{\mathsf{NP}} \not\subseteq \mathsf{BP}\text{-}\mathsf{TC}^0$ *if and only if* Heavy-Avoid *for implicit* DLOGTIME*-uniform* TC^0*-samplers can be solved in deterministic polynomial time with access to and* NP *oracle on infinitely many input lengths.*

(*ii*) $\mathsf{PSPACE} \not\subseteq \mathsf{BP}\text{-}\mathsf{ACC}^0$ *if and only if* Heavy-Avoid *for implicit* DLOGTIME*-uniform* ACC^0*-samplers can be solved in logarithmic space on infinitely many input lengths.*

Previously, algorithmic characterisations of *non-uniform* lower bounds were known for classes such as NEXP [IKW02, Wil16] and $\mathsf{EXP}^{\mathsf{NP}}$ [Kor21, RSW22], and such characterisations for uniform randomised lower bounds against *general* circuits (that is, against BPP) were known for EXP [IW01] and NEXP [Wil16]. We are not aware of any previous algorithmic characterisation of super-polynomial non-uniform or uniform randomised lower bounds for NP.

Connections to Derandomisation

We also explore relations between the complexity of Heavy Avoid and fundamental questions in derandomisation. We consider the *non-uniform* variant of Heavy Avoid, where a Boolean circuit sampler is given as input to the algorithm solving Heavy Avoid. For $\delta\colon \mathbb{N} \to [0,1]$,

`Implicit-δ-Heavy-Avoid` is the problem where we are given as input a circuit C implicitly sampling a distribution on N bits (as explained at the beginning of Section 8.1.1), and would like to output a δ-light element in the distribution.

Our first result shows that the existence of efficient deterministic algorithms for Heavy Avoid that, in addition, can be implemented by uniform circuits of sub-polynomial depth leads to a complete derandomisation of prBPP. Note that in this result, to obtain the desired conclusion, it is sufficient for this algorithm to solve the problem for implicit samplers.

Theorem 8.1.3. *Let $\delta(N) = o(1)$ be any function. Suppose there is a constant $\varepsilon > 0$ and a deterministic algorithm $\mathcal{A}$ that solves the* `Implicit-δ-Heavy-Avoid` *problem on implicit samplers of size N^{ε}. Moreover, assume that $\mathcal{A}$ can be implemented as a logspace-uniform circuit of size* $\mathrm{poly}(N)$ *and depth $N^{o(1)}$. Then* prBPP = prP.

If we could eliminate the circuit depth constraint from the statement of Theorem 8.1.3, it would be possible to establish an equivalence between the derandomisation of prBPP and algorithms for Heavy Avoid (in both the implicit and explicit settings). While obtaining this strong characterisation remains elusive, in the next result, we obtain a non-trivial derandomisation consequence from the existence of an efficient algorithm for Heavy Avoid without assuming a circuit depth bound.

Let Gap-SAT denote the promise problem where YES instances are Boolean circuits with at least half of the assignments being satisfying, and NO instances are unsatisfiable Boolean circuits. It is well known that Gap-SAT is complete for the promise version of RP.

Theorem 8.1.4 (Informal)**.** *Let $\delta(N) = o(1)$ be any function. Suppose there is an algorithm for* `Implicit-δ-Heavy-Avoid` *on maps $G\colon \{0,1\}^{\mathrm{poly}(n)} \to \{0,1\}^N$ (where $N = 2^{n^{\varepsilon}}$) implicitly computable by an input circuit of size* $\mathrm{poly}(n)$*, where the Heavy Avoid algorithm runs in* $\mathrm{poly}(N)$ *time and is infinitely-often correct. Then there is an algorithm for* Gap-SAT *that runs in subexponential time and is infinitely-often* correct.*[7]

Theorem 8.1.3 and Theorem 8.1.4 are both established using *non-black-box reductions* that make use of recent hardness-randomness trade-offs. In more detail, as explained in Section 8.1.2 below, Theorem 8.1.3 crucially relies on the instance-wise hardness-randomness trade-off for low-depth circuits of Chen and Tell [CT21a], while Theorem 8.1.4 combines the framework of [CT21a] and the "leakage resilient" hardness-randomness framework of Liu and Pass [LP23]. In contrast to the non-black-box nature of the proofs given for these two results, we show that it will be quite difficult to obtain them using *black-box* reductions. In particular, we show that improving Theorem 8.1.4 to a polynomial-time Levin reduction [Lev73] would derandomise prBPP.[8] Stated more precisely, if there is an efficient black-box Levin reduction from the search

[7] In Theorem 8.1.4, we only obtain Gap-SAT algorithms satisfying a technical condition called infinitely-often* correctness, which is a nonstandard variant of infinitely-often correctness. The crucial difference is that, for a sequence of inputs $\{x_n\}_{n\in\mathbb{N}}$, given 1^n, the algorithm is allowed to inspect every input $x_1, x_2, \ldots, x_{\mathrm{poly}(n)}$, and needs to provide a solution for x_n. In other words, the algorithm is correct *infinitely-often** if it outputs the correct answer on infinitely many input lengths n while having access to all input strings from the sequence that have length polynomial in n. We refer the reader to Definition 8.4.7 and to the proof of Theorem 8.4.8 for more details.

[8] Recall that in a Levin reduction between search problems we have a pair (f, g) of functions, where f maps to an instance of the other problem while g converts a given solution into a solution to the original problem.

version of Gap-SAT to Heavy Avoid (even with respect to non-uniform explicit samplers), then prBPP = prP holds *unconditionally*. We refer to Section 8.4.3 for more details.

Finally, we establish a deeper connection between the implicit non-uniform variant of Heavy Avoid considered in this section and the recent paradigm of *instance-wise* hardness-randomness trade-offs alluded to above [CT21a, LP22, LP23, CTW23]. Roughly speaking, in this paradigm, we convert a hard function $f : \{0,1\}^n \to \{0,1\}^{\mathrm{poly}(n)}$ with multiple output bits into pseudorandomness, where the obtained derandomisation is *instance-wise*: for every $x \in \{0,1\}^n$, if f is hard to compute on x, then the derandomisation of the corresponding computation over input x succeeds. Naturally, the derandomisation assumptions used in these results need *almost-all-inputs hardness*, meaning that f is hard on *all but finitely many* inputs (instead of input lengths).[9] In Section 8.4.4, we prove that the existence of efficient deterministic algorithms for Heavy Avoid in the implicit non-uniform setting is *equivalent* to the existence of functions f with multiple output bits that are easy to compute deterministically but are hard against fixed polynomial-size randomised algorithms. This result sheds light on the relevance of the techniques that we employ to prove Theorem 8.1.3 and Theorem 8.1.4, and suggests that developing further connections between Heavy Avoid and these modern hardness-randomness trade-offs paradigms could be a fruitful research direction.

8.1.2 Techniques

We now discuss the proofs of Theorem 8.1.1, Theorem 8.1.3, and Theorem 8.1.4. We make use of a variety of techniques to establish these results:

- The proof of Theorem 8.1.1 Item (*iii*) relies on extremely efficient *instance checkers* for a special PSPACE-complete problem investigated in [Che23]. This allows us to establish equivalences for very weak circuit classes $\mathcal{C}$ at the frontier of existing separations. Extending the equivalence result to NP, EXP, and $\mathsf{EXP}^{\mathsf{NP}}$ in the context of weak circuit classes poses some additional challenges that we address through different ideas and techniques.

- The proof of Theorem 8.1.3 relies on a novel application of the Chen–Tell *non-black-box hitting set generator* construction from [CT21a, CLO+23]. In contrast to previous applications, here the *reconstruction procedure* of the generator itself, as well as the assumed algorithm for Heavy Avoid, plays a key role in the specification of a "hard" function.

- Finally, the proof of Theorem 8.1.4 builds on the proof of Theorem 8.1.3. It combines for the first time the Chen–Tell derandomisation framework [CT21a] with the *leakage resilience* derandomisation framework of [LP23], using a win-win analysis. We show that either the Heavy Avoid algorithm is leakage resilient, allowing us to apply the framework of [LP23], or it can be implemented by a low-depth circuit, allowing us to apply the framework of [CT21a]. This enables us to derive a non-trivial derandomisation consequence without the circuit depth constraint present in the hypothesis of Theorem 8.1.3.

Next, we describe some of our proofs and techniques in more detail.

[9] Compared with classical hardness-randomness frameworks such as [NW94, IW97, STV01], the advantage of the new paradigm is that lower bounds against *uniform algorithms* (instead of non-uniform circuits) suffice for *worst-case* derandomisation.

Sketch of the Proof of Theorem 8.1.1. We first explain the proof of Item (*iii*), i.e., the equivalence between the complexity separation $\mathsf{PSPACE} \not\subseteq \mathsf{BP}\text{-}\mathcal{C}$ and the existence of (infinitely often) logarithmic-space algorithms for Heavy-Avoid over implicit DLOGTIME-uniform $\mathcal{C}$-samplers.

First, we show how to obtain the separation using algorithms for the implicit heavy avoid problem. Using standard arguments, it suffices to show that for every choice of $k \geq 1$, there is $L \in \mathsf{DSPACE}[n^2]$ such that L cannot be computed by $\mathsf{DTIME}[k \cdot \log n]$-uniform randomised $\mathcal{C}$-circuits of size n^k.

Let $N = 2^n$. We consider a map $G_N \colon \{0,1\}^{n^{O(k)}} \to \{0,1\}^N$ that views its input string x as a pair (M, r), where M is a short encoding (say, $\log n$ bits) of a clocked machine running in time $10k \cdot \log n$, and r is a random string. Let D_M be the $\mathcal{C}$-circuit of size at most n^{2k} whose direct connection language is encoded by the machine M. For $i \in [N]$, we define the i-th output bit of $G_N(x)$ as $D_M(r, i)$. Due to its running time, the computation of M can be uniformly converted into an AC^0 circuit of size at most n^{10k}. Using that $\mathcal{C}$ is a nice circuit class, G_N can be implicitly computed by a DLOGTIME-uniform probabilistic $\mathcal{C}$-circuit C_N of size at most $n^{O(k)}$.

Let $B(1^N)$ be an algorithm of space complexity $O(\log N)$ that solves $\mathcal{C}$-`Implicit-`δ-`Heavy-Avoid` on infinitely many values of N for the sequence G_N, and let L_B be the language defined by B. Note that L_B is in $\mathsf{DSPACE}[O(n)] \subseteq \mathsf{DSPACE}[n^2]$. To argue that L_B cannot be computed by $\mathsf{DTIME}[k \cdot \log n]$-uniform randomised $\mathcal{C}$-circuits of size n^k, it is enough to show that for every language L computed by such circuits, each string in the sequence $\{y_N^L\}_N$ of truth-tables obtained from L is δ-heavy in $G_N(\mathcal{U}_{m(N)})$. Since B solves $\mathcal{C}$-`Implicit-`δ-`Heavy-Avoid` for the sequence $\{G_N\}$, it follows that $L_B \neq L$.

The proof that the sequence $\{y_N^L\}_N$ of truth-tables obtained from L is δ-heavy in $G_N(\mathcal{U}_{m(N)})$ relies on the definition of G_N. In more detail, under the assumption that L admits $\mathsf{DTIME}[k \cdot \log n]$-uniform randomised $\mathcal{C}$-circuits of size n^k, it is not hard to show that its truth-table is produced with probability comparable to $2^{-|M|}$. However, this probability is sufficiently large under the assumption that the encoding length $|M|$ is small in the definition of G_N.

The proof of the other direction in Theorem 8.1.1 is more interesting. We establish the contrapositive. Suppose that for some $G_N \colon \{0,1\}^{\mathrm{poly}(n)} \to \{0,1\}^N$ implicitly computed by DLOGTIME-uniform $\mathcal{C}$-circuits of size $\mathrm{poly}(n)$, every algorithm $A(1^N)$ running in space $O(\log N)$ fails to solve $\mathcal{C}$-`Implicit-`δ-`Heavy-Avoid` on every large enough input length N. We employ this assumption to show that $\mathsf{PSPACE} \subseteq \mathsf{BP}\text{-}\mathcal{C}$. For this, we recall the notion of *instance checkers*. Let $L \subseteq \{0,1\}^*$ be a language, and let $\{C_n^{(-)}(x, z)\}_{n \in \mathbb{N}}$ be a family of probabilistic oracle circuits. We say that C is an *instance checker* for L if for every input $x \in \{0,1\}^*$:

1. $\Pr_z[C^L_{|x|}(x, z) = L(x)] = 1$, and
2. for every oracle $\mathcal{O}$, $\Pr_z[C^{\mathcal{O}}_{|x|}(x, z) \notin \{L(x), \perp\}] \leq 1/2^n$.

We will rely on an appropriate PSPACE-complete language $L^\star$ that admits highly efficient instance checkers computable in any nice circuit class. This is a consequence of a result from [Che23], as explained in Section 8.5.

We then consider a candidate algorithm $A(1^N)$ that computes as follows. On input 1^N, define tt_N to be the truth table of $L^\star$ on n-bit inputs; we simply output tt_N. It is possible to

show that A computes in space $O(\log N)$ after an appropriate scaling of parameters, which we omit here for simplicity. Therefore, A fails to solve $\mathcal{C}$-$\texttt{Implicit}$-δ-$\texttt{Heavy-Avoid}$ on every large enough input length N. This means that for every large enough N, the probability of tt_N under the distribution $G_N(\mathcal{U}_{\mathrm{poly}(n)})$ from above is at least $\delta = 1/(\log N)^{O(1)} = 1/\mathrm{poly}(n)$.

To explain how we compute $L^\star$ on an input $x \in \{0,1\}^n$, assume for simplicity that the oracle instance checker circuit (call it IC) only queries its oracle on input length n. We sample $v := n^{O(1)}$ strings $z_1, \ldots, z_v \in \{0,1\}^{\mathrm{poly}(n)}$ uniformly and independently at random, and for each string z_i, we define an oracle $\mathcal{O}_i$ whose truth table is the string $G_N(z_i) \in \{0,1\}^N$. We run IC in parallel and obtain $b_i := \mathsf{IC}_n^{\mathcal{O}_i}(x)$ for each $i \in [v]$. We output 1 if at least one bit among $b_1, \ldots, b_v$ is 1, and 0 otherwise.

Next, we argue that A computes $L^\star$ with high probability. Let tt_N denote the truth table of $L^\star$ on input length n. By our choice of v, with high probability the string tt_N appears among the strings $G_N(z_1), \ldots, G_N(z_v)$, meaning that one of the oracles $\mathcal{O}_i$ computes $L^\star$ on inputs of length n. Consequently, in this case, if $L^\star(x) = 1$ then at least one bit $b_i = 1$, and the procedure outputs 1. On the other hand, if $L^\star(x) = 0$, then by a union bound over the internal randomness of IC, with high probability every bit $b_i \in \{0, \perp\}$. In this case, the procedure outputs 0. This establishes the correctness of A. Using the efficiency of the instance checker and that $\mathcal{C}$ is a nice circuit class, it is also possible to upper bound the circuit complexity of A and to analyse the uniformity of the corresponding circuits. This implies that $L^\star \in \mathsf{BP}\text{-}\mathcal{C}$. Since $L^\star$ is PSPACE-complete under $\mathsf{DLOGTIME}$-uniform projection reductions, we get that $\mathsf{PSPACE} \subseteq \mathsf{BP}\text{-}\mathcal{C}$, as desired.

We now briefly comment on the additional ideas needed for the proofs of the other items in Theorem 8.1.1. The proof of Item (*ii*) requires a different approach, since instance checkers for $\mathsf{EXP}^{\mathsf{NP}}$-complete languages are not known. We provide two different proofs in this case. In more detail, the result for $\mathsf{EXP}^{\mathsf{NP}}$ can be obtained using a win-win argument and a reduction to Item (*iii*), or through the use of *selectors* for $\mathsf{EXP}^{\mathsf{NP}}$-complete languages [Hir15]. These two approaches provide different extensions of the result, which we discuss in detail in Section 8.3.3. On the other hand, the proof of Item (*iv*) relies on a randomised depth-efficient version of the search-to-decision reduction for SAT based on the Valiant-Vazirani Isolation Lemma [VV86], as well as the equivalence between the polynomial hierarchy and $\mathsf{DLOGTIME}$-uniform constant-depth circuits of exponential size [BIS90].

Sketch of the Proof of Theorem 8.1.3. Using existing results [BF99], in order to derandomise prBPP it is sufficient to describe an algorithm that, given an input circuit $D\colon \{0,1\}^M \to \{0,1\}$ of size $O(M)$ with the promise that $\Pr_y[D(y) = 1] \geq 1/2$, runs in deterministic time $\mathrm{poly}(M)$ and outputs a positive input of D. To achieve this, we will rely on a novel application of the Chen–Tell generator [CT21a] (with the improved parameters from [CLO+23]). In more detail, given a function $f\colon \{0,1\}^n \to \{0,1\}^{T(n)}$ computed by logspace uniform circuits of size $T(n)$ and depth $d(n)$, and a parameter $M(n)$ such that $c \cdot \log T \leq M \leq T^{1/c}$ (for a constant c), [CT21a, CLO+23] provides algorithms HSG_f and Recon_f depending on f such that:

- The algorithm $\mathsf{HSG}_f(x)$ runs in deterministic T^c time and outputs a set of M-bit strings.
- Given $x \in \{0,1\}^n$ and $i \in [T]$ as inputs, and oracle access to a candidate distinguisher

$D\colon \{0,1\}^M \to \{0,1\}$, $\mathsf{Recon}_f^D(x,i)$ runs in randomised $(dnM)^c$ time. If D is dense and avoids $\mathsf{HSG}_f(x)$, then with probability $\geq 1 - 2^{-M}$, $\mathsf{Recon}_f^D(x,i)$ outputs the i-th bit of $f(x)$.

We consider an appropriate function $f'\colon \{0,1\}^{\widetilde{O}(M)} \to \{0,1\}^N$, where $N = M^{C_1}$ for a large enough constant C_1. We view the input of f' as the description of an arbitrary circuit $D\colon \{0,1\}^M \to \{0,1\}$ of size $O(M)$. In this construction, the parameter $T = M^{C_2}$ for a large enough constant $C_2 > C_1$, while $d = M^{o(1)} = N^{o(1)}$. Moreover, f' will be computed by a logspace-uniform family of circuits. We then show that $\mathsf{HSG}_{f'}(D)$ hits D if D is a dense circuit. Note that the generator runs in time $\mathrm{poly}(T) = \mathrm{poly}(M)$ by our choice of parameters.

The function f' makes use of the algorithm $\mathcal{A}$ that solves the `Implicit-`δ`-Heavy-Avoid` problem on instances $G\colon \{0,1\}^{N^\varepsilon} \to \{0,1\}^N$ that are implicitly computable in N^ε size. In more detail, we let $f'(D) = \mathcal{A}(C_D)$, where C_D is an implicit (non-uniform) sampler of size N^ε for a map $G_D\colon \{0,1\}^{N^\varepsilon} \to \{0,1\}^N$ described next.

First, we make a simplifying assumption: The sampler G_D has access to the code of a machine $M_{f'}$ that serves as a logspace-uniform description of a circuit family that computes f'. (Observe that this is self-referential, since we have defined $f'(D) = \mathcal{A}(C_D)$ above, while we will also use f' to define C_D. We will handle this issue later.)

The sampler G_D stores D as advice. This is possible because D is of size M, and if C_1 is large enough, then $M \ll N^\varepsilon$. The implicit sampler $C_D(r,i)$ for G_D then uses its random input string r of length N^ε and $i \in [\log N]$ to compute $\mathsf{Recon}_{f'}^D(D,i,r)$, where we have made explicit the random string r used by $\mathsf{Recon}_{f'}^D$. Since $d = M^{o(1)}$ and C_1 is large enough, we get that $\mathsf{Recon}_{f'}^D(D,i,r)$ can be computed in time $(d \cdot M^{1+o(1)} \cdot M)^c \leq M^{c+o(1)} \leq N^\varepsilon$. This completes the definition of $f'(D)$ and of $\mathsf{HSG}_{f'}(D)$. We note that to establish the size, depth, and logspace-uniformity of the sequence of circuits computing f' we can rely on the fact that f' only needs to produce the *code* of C_D.[10]

Next, we argue that $\mathsf{HSG}_{f'}(D)$ hits any dense circuit D. Assume this is not the case. Then, since D avoids the generator, $\mathsf{Recon}_f^D(D,i)$ outputs the i-th bit of $f'(D)$ with probability at least $1 - 2^{-M}$. Consequently, by a union bound over $i \in [N]$, it follows that the string $\mathcal{A}(C_D) = f'(D) \in \{0,1\}^N$ is output by $\mathsf{Recon}_f^D(D,\cdot)$ with probability $1 - o(1)$. In other words, the string $f'(D)$ is sampled with high probability by the sampler G_D encoded by C_D. On the other hand, since $f'(D) = \mathcal{A}(C_D)$ and $\mathcal{A}$ solves the heavy avoid problem for G_D, we get that the string $f'(D)$ has probability $o(1)$ under G_D. This contradiction implies that $\mathsf{HSG}_{f'}(D)$ indeed hits D.

It remains to explain how to fix the self-referential nature of the definition of G_D via the implicit sampler C_D, which depends on f' (and which in turn depends on C_D). In more detail, the construction is self-referential due to the use of the routine $\mathsf{Recon}_{f'}^D$, which depends on f'. To patch the argument, we combine the following key points:

- There is a deterministic algorithm that, given the Turing machine $M_{f'}$ that prints the circuit for f' in logspace, outputs the description of $\mathsf{Recon}_{f'}$ in $\mathrm{poly}(|\langle M_{f'}\rangle|)$ time.

[10] We make a brief comment about the novelty of this argument. In order to define the "hard" function f', here we make use of the *reconstruction procedure* of the generator. This is different from an application of this generator in [CLO+23], where the code of the *hitting set procedure* plays a key role in the definition of the hard function.

- We can combine $O(1)$ many samplers into a single sampler that produces the convex combination of the corresponding distributions. A string with weight $o(1)$ under the new distribution must have weight $o(1)$ under each original sampler.

Therefore, we can change the description of G_D so that it interprets a small prefix of its random input string as the description of a Turing machine M_f that prints a circuit of the expected size using logarithmic uniformity, then use the first bullet above to produce the procedure Recon_f corresponding to f. Notice that with this change, the sampler G_D no longer depends on f'. Moreover, since f' is encoded by some finite machine $M_{f'}$, using the second bullet the argument described above to reach a contradiction and establish the correctness of the hitting set generator still holds: When D avoids $\mathsf{HSG}_{f'}(D)$ the modified sampler G_D outputs the string $f'(D) = \mathcal{A}(C_D)$ with constant probability, while as a solution to the heavy avoid problem for G_D this string has probability $o(1)$. This completes the sketch of the argument.[11]

Sketch of the Proof of Theorem 8.1.4. Since this is a more sophisticated construction, we only provide a brief sketch of the idea. As alluded to above, the argument combines the two instance-wise hardness-randomness trade-offs introduced by Chen and Tell [CT21a] and by Liu and Pass [LP23], respectively.

We employ a win-win analysis based on whether the assumed algorithm for `Implicit-`δ`-Heavy-Avoid` (call it Avoid) is "*leakage resilient*" hard. In more detail, let $f\colon \{0,1\}^n \to \{0,1\}^T$ be a function, A be a randomised algorithm, and $x \in \{0,1\}^n$ be an input of f. We say that $f(x)$ is ℓ-*leakage resilient hard* against A if for every "leakage string" $\mathsf{leak} \in \{0,1\}^\ell$, there is some $i \in [T]$ such that $\Pr[A(x, \mathsf{leak}, i) = f(x)_i] \leq 2/3$, where the probability is taken over the internal randomness of A. Liu and Pass [LP23] showed that leakage resilient hardness can be used for derandomisation.

We can now explain the main idea behind the win-win analysis. If Avoid is leakage resilient hard, we use the hardness-randomness trade-offs in [LP23]. If this is not the case, we show that Avoid can actually be implemented by a low-depth circuit. We can then use the hardness-randomness trade-offs in [CT21a], which requires the hard function to be computed by a low-depth circuit family.

Implementing this plan turns out to require a delicate construction and the notion of infinitely-often* correctness appearing in the statement of Theorem 8.1.4. We refer to Section 8.4.2 for more details.

8.2 Preliminaries

8.2.1 Notation

We use $\mathcal{U}_n$ to denote the uniform distribution over $\{0,1\}^n$. For a distribution D and an element x, we use $D(x)$ to denote the probability of x under D.

We say that a probability distribution D contains a δ-*heavy element* if there is x in the support of D such that $D(x) \geq \delta$. Any such element x is said to be δ-*heavy*. In this case, we also say that the distribution D is δ-heavy. If an element x is not δ-heavy, then we say it is δ-*light*.

[11] We note that this argument is non-black-box. The code of a machine $M_{f'}$ that describes a uniform circuit family for f' is needed to instantiate the Chen-Tell generator. In the aforementioned construction, this means that black-box access to the algorithm $\mathcal{A}$ is not enough.

We will often consider a distribution ensemble $\mathcal{D} = \{D_n\}_{n\geq 1}$, where each D_n is a distribution supported over $\{0,1\}^n$. For convenience, we might simply refer to $\mathcal{D}$ as a distribution. We let PSAMP denote the set of polynomial-time samplable distributions.

We say a probabilistic algorithm $\mathcal{A}$ for a search problem $\mathcal{P}$ is *pseudodeterministic* [GG11], if for every input x, there is a *canonical* $\mathcal{P}$-solution y of x such that $\mathcal{A}(x)$ outputs y with probability $\geq 2/3$. It is easy to see that the success probability can be amplified to $1 - \exp(-n)$ by parallel repetition.

8.2.2 The Heavy Avoid Problem

In general, given a distribution $\mathcal{D}$ over $\{0,1\}^n$ and a parameter $\delta \in (0,1)$, the Heavy Avoid problem asks to find a string $x \in \{0,1\}^n$ such that $\mathcal{D}(x) < \delta$. It is easy to see that such a string always exists as long as $2^n > 1/\delta$. Consequently, the Heavy Avoid problem is a *total* search problem. We mainly focus on the regime where $1/\delta$ is significantly smaller than 2^n, such as $\delta = 1/\mathrm{poly}(n)$ or even just $\delta \approx 1/\log n$.

We consider the Heavy Avoid problem in different settings, depending on whether the sampler for $\mathcal{D}$ *implicitly* samples the distribution and whether it is computed *uniformly*.[12]

- We say a distribution $\mathcal{D}$ over $\{0,1\}^n$ is *implicit* (or, *locally-samplable*) if there is an efficient procedure that given an integer i and the randomness r used by the sampler, outputs the i-th bit of the sample according to r. In the typical parameter regime, $\mathcal{D}$ runs in time $t \approx \mathrm{poly}(\log n, |r|)$ which is much smaller than n. Depending on the context, "efficient procedure" could either mean Turing machines or circuits, as will be addressed in the next bullet. Note that the random string r is also short and the sampler has sequential access (instead of random access) to r.

  We use the word *explicit* to describe samplers that take $\mathrm{poly}(n)$ time, as opposed to implicit samplers.

- We say a family of distributions $\mathcal{D} = \{D_n\}_{n\in\mathbb{N}}$ is *uniformly samplable* if there is a Turing machine M that given 1^n (and access to uniformly random bits), samples from D_n. (Similarly, we often consider a Turing machine $M(1^n)$ that prints a circuit that samples D_n.) On the other hand, if we only have a (non-uniform) family of circuits $\{C_n\}$, where each C_n samples from D_n, then we say the distribution is *non-uniformly samplable*.

  We will also say that uniformly samplable distributions are sampled in *time* t, while non-uniformly samplable distributions are sampled in *size* t.

Explicit Maps

Definition 8.2.1 (Uniform Heavy Avoid)**.** Let $\mathcal{D} = \{D_n\} \in \mathsf{PSAMP}$, where each D_n is supported over $\{0,1\}^n$, and let $\delta(n) \in [0,1]$. In the $(\mathcal{D},\delta)$-`Heavy-Avoid` problem, given 1^n the goal is to output an element $x \in \{0,1\}^n$ such that $D_n(x) < \delta(n)$.

[12] Do not confuse the uniformity of the sampler with the distribution D_n, which most often in this work will not be the uniform distribution.

We say that the $(\mathcal{D}, \delta)$-Heavy-Avoid problem can be solved in polynomial time if there is a deterministic algorithm $A(1^n)$ that runs in polynomial time and solves $(\mathcal{D}, \delta)$-Heavy-Avoid. Similarly, the $(\mathcal{D}, \delta)$-Heavy-Avoid problem can be solved in pseudodeterministic polynomial time if there is a pseudodeterministic algorithm $A(1^n)$ that runs in polynomial time and solves $(\mathcal{D}, \delta)$-Heavy-Avoid.

Definition 8.2.2 (Non-Uniform Heavy Avoid)**.** Let $C\colon \{0,1\}^m \to \{0,1\}^n$ be a Boolean circuit, and let D_C be the distribution induced by $C(\mathcal{U}_m)$. Let $\delta \in [0,1]$. In the Non-Uniform-Heavy-Avoid problem, given C and δ, the goal is to output an element $x \in \{0,1\}^n$ such that $D_C(x) < \delta$.

We may also represent $1/\delta$ in unary when we want to emphasise that we consider the regime where $\delta \geq 1/\text{poly}(n)$. In this case, the input consists of $(C, 1^t)$, let $\delta := 1/t$, and the goal is to output a δ-light element of D_C.

We say that Non-Uniform-Heavy-Avoid can be solved in polynomial time if for every constant $c \geq 1$, Non-Uniform-Heavy-Avoid over inputs where the circuit C is of size at most n^c and $\delta \geq 1/n^c$ can be solved in deterministic polynomial time.

Note that it is not hard to solve Non-Uniform-Heavy-Avoid with randomness.

Proposition 8.2.3. *Let $c \geq 1$. There is a probabilistic polynomial-time algorithm A such that, given a circuit $C\colon \{0,1\}^m \to \{0,1\}^n$ of size at most n^c and a parameter $\delta \geq 1/n^c$, A runs in time polynomial in n^c and outputs with high probability a set $T_{C,\delta}$ of size at most $2\cdot(1/\delta)$ that contains all δ-heavy elements of D_C. Hence, if we output the lexicographically smallest string not in $T_{C,\delta}$, then we obtain a probabilistic polynomial-time algorithm solving* Non-Uniform-Heavy-Avoid*.*

Let $S_{C,\delta} = \{x \in \{0,1\}^c \mid D_C(x) \geq \delta\}$, where $D_C = C(\mathcal{U}_m)$. Note that the algorithm A outputs with high probability a set $T_{C,\delta}$ of bounded size such that $S_{C,\delta} \subseteq T_{C,\delta}$. However, different executions of A might produce different sets $T_{C,\delta}$. Consequently, this does not give rise to a *pseudodeterministic* algorithm for Non-Uniform-Heavy-Avoid.

Implicit Maps (Locally Samplable Distributions)

For locally samplable distributions (which will also be called "implicit maps" in this chapter), it will be important to fix the following notation. A *map*, or *generator*, is a function $G\colon \{0,1\}^m \to \{0,1\}^N$ (typically $N \gg m$) such that our input distribution is $G(\mathcal{U}_m)$. We say the map is *implicitly computed* by a circuit C if $C\colon \{0,1\}^m \times [N] \to \{0,1\}$ satisfies that for every $r \in \{0,1\}^m$, $C(r,i)$ outputs the i-th bit of $G(r)$. The input of Implicit-Heavy-Avoid will be a circuit C even though we are actually solving Heavy Avoid on the corresponding instance G. (In the uniform case, the circuit C is generated by a uniform procedure, in which case the input to the problem is simply 1^N.)

Although the input length, $\text{poly}(|C|)$, is usually much smaller than N, the output length is still N, hence we still measure the time complexity of algorithms solving Implicit-Heavy-Avoid by N. For example, we say Implicit-Heavy-Avoid can be solved in deterministic polynomial time if it can be solved by a deterministic machine that runs in time polynomial in N.

Definition 8.2.4 ($\mathcal{C}$-Implicit-δ-Heavy-Avoid for non-uniform samplers)**.** Let $\mathcal{C}$ be a circuit class, $\delta\colon \mathbb{N} \to [0,1]$, $m, N, s\colon \mathbb{N} \to \mathbb{N}$ be parameters. We define the *$\mathcal{C}$-implicit δ-heavy avoid*

problem for maps that stretch $m(N)$ bits to N bits and are implicitly computed by $\mathcal{C}$-circuits of size at most $s(N)$. (A typical parameter regime is that $N = 2^n$ for some integer n, $\delta(N) = 1/\mathrm{poly}(n)$, and $m(N), s(N) \leq \mathrm{poly}(n)$. The parameters will be clear in each statement.)

The input of this problem is a size-s $\mathcal{C}$-circuit $C\colon \{0,1\}^m \times [N] \to \{0,1\}$. Recall that this circuit C implicitly defines a map $G\colon \{0,1\}^m \to \{0,1\}^N$ such that, for every $r \in \{0,1\}^m$, and $i \in [N]$, $C(r,i)$ outputs $G(r)_i$ (the i-th bit of the N-bit string $G(r)$). Given C, the goal is to output an element $y \in \{0,1\}^N$ such that $\Pr[G(\mathcal{U}_m) = y] < \delta$.

Similarly, we can also define $\mathcal{C}$-`Implicit`-δ-`Heavy-Avoid` for uniformly-samplable maps. Here, we consider families of maps $\{G_N\}$ that are implicitly computed by DLOGTIME-uniform $\mathcal{C}$-circuits $\{C_N\}$ of size at most $s(N)$. In other words, there is a DLOGTIME-uniform sequence $\{C_N\}_{N\geq 1}$ of size-$s(N)$ $\mathcal{C}$-circuits such that, for every $N \geq 1$, $r \in \{0,1\}^{m(N)}$, and $i \in [N]$, $C_N(r,i)$ outputs $G_N(r)_i$, i.e., the i-th bit of the N-bit string $G_N(r)$. (Here, we say $\{C_N\}$ is DLOGTIME-uniform if the *direct connection language* of C_N can be decided in $O(\log s(N))$ time.)

Definition 8.2.5 ($\mathcal{C}$-`Implicit`-δ-`Heavy-Avoid` for uniform samplers)**.** Let $m(N), s(N), \delta(N)$ be parameters as above, and $\{C_N\}$ be a DLOGTIME-uniform sequence of $\mathcal{C}$-circuits that defines a family of maps $\{G_N\}$. That is, given $r \in \{0,1\}^m$ and $i \in [N]$, $C_N(r,i)$ outputs the i-th bit of $G_N(r)$. The $\mathcal{C}$-`Implicit`-δ-`Heavy-Avoid` problem corresponding to $\{G_N\}$ is the following problem: Given 1^N the goal is to output a string $x \in \{0,1\}^N$ such that $\Pr_r[G_N(r) = x] < \delta(N)$.

Note that the input to the Heavy Avoid problem is given by a circuit when we consider the non-uniform formulations (in both the implicit and explicit settings), while the input to the problem is simply the input length when we consider uniform formulations (since the sampler can be efficiently obtained from the input length).

8.2.3 Time-Bounded Kolmogorov Complexity

We review some notions from time-bounded Kolmogorov complexity (see, e.g., [LO22] for more details). Let U be a Turing machine. Given a positive integer t and a string $x \in \{0,1\}^*$, we let

$$\mathsf{K}^t_U(x) = \min_{p\in\{0,1\}^*} \Big\{ |p| \;\Big|\; U(p) \text{ outputs } x \text{ in at most } t \text{ steps} \Big\}.$$

We say that $\mathsf{K}^t_U(x)$ is the *t-time-bounded Kolmogorov complexity of x* (with respect to U). As usual, we fix U to be a time-optimal machine [LV19], i.e., a universal machine that is almost as fast and length efficient as any other universal machine, and drop the index U when referring to time-bounded Kolmogorov complexity measures.

For $x \in \{0,1\}^*$, the *probabilistic t-time-bounded Kolmogorov complexity* of x is defined as

$$\mathrm{pK}^t(x) = \min \left\{ k \in \mathbb{N} \;\middle|\; \Pr_{w\sim\{0,1\}^t} \Big[\exists\, p \in \{0,1\}^k,\, U(p,w) \text{ outputs } x \text{ within } t \text{ steps} \Big] \geq \frac{2}{3} \right\}.$$

In other words, if $k = \mathrm{pK}^t(x)$, then with probability at least $2/3$ over the choice of the random string w, given w, the string x admits a t-time-bounded encoding of length k.

We can also consider the *randomised* Kt *complexity* of a string $x \in \{0,1\}^*$, defined as

$$\mathrm{rKt}(x) = \min_{t \in \mathbb{N},\, p \in \{0,1\}^*} \left\{ |p| + \lceil \log t \rceil \;\middle|\; \Pr_{r \sim \{0,1\}^*} [U(p,r) \text{ outputs } x \text{ in } t \text{ steps}] \geq 2/3 \right\}.$$

All these notions of time-bounded Kolmogorov complexity can be generalised to capture the conditional complexity of x given y in the natural way, i.e., by providing y as an extra input string to the universal machine U.

8.2.4 Pseudorandomness and Derandomisation

Fix an input length n. A *generator* is simply a multiset $G \subseteq \{0,1\}^n$. We will consider families of generators $\{G_n\}_{n \in \mathbb{N}}$ where each $G_n \subseteq \{0,1\}^n$ is a generator outputting n-bit strings. In the literature, it is also common to consider these generators as functions: let $\ell(n) < n$ denote the seed length of the generator, then the function $G_n \colon \{0,1\}^{\ell(n)} \to \{0,1\}^n$ is equivalent to the multiset

$$\{G_n(s) : s \in \{0,1\}^{\ell(n)}\}.$$

In this chapter, we will use the subset- and functional-definitions of generators interchangeably.

Let $\mathcal{A} : \{0,1\}^n \to \{0,1\}$ be a function, $H \subseteq \{0,1\}^n$ be a generator, and $\varepsilon > 0$ be a parameter. We say that $\mathcal{A}$ is *ε-dense* if $\Pr_{x \sim \{0,1\}^n}[\mathcal{A}(x) = 1] \geq \varepsilon$. We say that $\mathcal{A}$ *ε-avoids* H if $\mathcal{A}$ is ε-dense, and for every string $x \in H$, we have $\mathcal{A}(x) = 0$. If $\mathcal{A}$ does not ε-avoid H, then we say that H *ε-hits* $\mathcal{A}$.

Let $\mathcal{A} : \{0,1\}^n \to \{0,1\}$ be a function, $G \colon \{0,1\}^\ell \to \{0,1\}^n$ be a generator, and $\varepsilon > 0$ be a parameter. We say that $\mathcal{A}$ *ε-distinguishes* G, if

$$\left| \Pr_{x \sim \{0,1\}^n} [\mathcal{A}(x) = 1] - \Pr_{s \sim \{0,1\}^\ell} [\mathcal{A}(G(s)) = 1] \right| > \varepsilon;$$

otherwise (if the above inequality does not hold), we say that G *ε-fools* A.

Like many papers in derandomisation [Gol11b, CT21a, LP22, LP23], we will consider *promise* versions of randomised complexity classes, such as prRP and prBPP. A *promise problem* [ESY84] $(\Pi_{\mathsf{YES}}, \Pi_{\mathsf{NO}})$ is a pair of disjoint sets $(\Pi_{\mathsf{YES}} \cap \Pi_{\mathsf{NO}} = \varnothing)$. A machine solves the corresponding promise problem if given an input $x \in \{0,1\}^*$, it outputs 1 when $x \in \Pi_{\mathsf{YES}}$ and outputs 0 when $x \in \Pi_{\mathsf{NO}}$; note that there is no requirement on the behaviour of the machine when $x \notin (\Pi_{\mathsf{YES}} \cup \Pi_{\mathsf{NO}})$.

We also recall the definitions of the canonical prRP-complete problem $\mathsf{Gap\text{-}SAT}$ and the canonical prBPP-complete problem CAPP.

Definition 8.2.6 ($\mathsf{Gap\text{-}SAT}$)**.** The problem $\mathsf{Gap\text{-}SAT}$ is the following promise problem $(\Pi_{\mathsf{YES}}, \Pi_{\mathsf{NO}})$: Π_{YES} consists of all circuits $C : \{0,1\}^n \to \{0,1\}$ that are 1/10-dense, and Π_{NO} consists of all circuits $C : \{0,1\}^n \to \{0,1\}$ such that $C(x) = 0$ for every $x \in \{0,1\}^n$.

Definition 8.2.7 (CAPP)**.** The problem CAPP is the following promise problem $(\Pi_{\mathsf{YES}}, \Pi_{\mathsf{NO}})$: On input (C, δ), where $C : \{0,1\}^n \to \{0,1\}$ is a circuit and $\delta \in (0,1)$ is a number, Π_{YES} consists of (C,δ) where $\delta \geq \Pr_{x \sim \{0,1\}^n}[C(x)] + 1/10$, and Π_{NO} consists of (C,δ) where $\delta \leq \Pr_{x \sim \{0,1\}^n}[C(x)] - 1/10$.

The constant $1/10$ in the above two definitions is arbitrary and can be amplified to $1/\mathrm{poly}(n)$ by parallel repetition.

8.3 Heavy Avoid and Lower Bounds Against Uniform Probabilistic Circuits

In this section, we study the connection between the `Heavy-Avoid` problem and the problem of proving lower bounds against *uniform probabilistic* circuits. Our main result is that in many settings, lower bounds against uniform probabilistic circuits are characterised by the existence of algorithms for `Implicit-Heavy-Avoid`.

Let $\mathcal{C}$ be a circuit class. A probabilistic $\mathcal{C}$-circuit $E(x; z)$ is a circuit from $\mathcal{C}$ that computes over an input x and an input z, where the latter corresponds to the random choice of E. We denote $\mathsf{BP}\text{-}\mathcal{C}$ the set of languages L that can be computed by a $\mathsf{DLOGTIME}$-uniform sequence of probabilistic $\mathcal{C}$-circuits of polynomial size. We stress that the uniform machine generating the $\mathcal{C}$ circuit is deterministic, while the circuit itself is allowed to make random choices (z).

Our results hold for any (uniform probabilistic) circuit class $\mathcal{C}$ that is *nice*, i.e., satisfies a few technical conditions. More precisely, we say a circuit class $\mathcal{C}$ is *nice* if the following holds:

- **($\mathcal{C}$ contains $\mathsf{AC}^0[\oplus]$.)** $\mathsf{AC}^0[\oplus] \subseteq \mathcal{C}$.
- **($\mathcal{C}$ is closed under composition.)** For every language $L \in \mathcal{C}$ and every $\mathsf{DLOGTIME}$-uniform family of oracle circuits $\{C_n^{(-)}\}_{n\in\mathbb{N}}$ making non-adaptive projection queries where the top (i.e., post-processing) circuit is in $\mathcal{C}$, the language computed by the circuit family $\{C_n^{L_n}\}_{n\in\mathbb{N}}$ is still in $\mathsf{DLOGTIME}$-uniform $\mathcal{C}$.
- **($\mathcal{C}$ admits universal circuits.)** There is a $\mathsf{DLOGTIME}$-uniform family of $\mathcal{C}$-circuits Eval such that given the description of a $\mathcal{C}$-circuit C (i.e., the truth table of the direct connection language of C) and an input x, $\mathsf{Eval}(\langle C\rangle, x)$ outputs $C(x)$.

  In the case that $\mathcal{C}$ is the union of depth-d circuits for every constant d, such as AC^0 or TC^0, we allow Eval to have higher depth than C: for every fixed depth d, the circuit evaluation problem can be solved by a family of $\mathsf{DLOGTIME}$-uniform $\mathcal{C}$-circuit of constant depth.

It is not hard to check that many standard circuit classes considered in the literature are nice, e.g., $\mathsf{AC}^0[\oplus]$, ACC^0, TC^0, NC^1, $\mathsf{P/poly}$. For instance, universal circuits for NC^1 are constructed in [Bus87], while universal circuits for TC^0 can be built using the universal threshold function (see, e.g., [BW05]) and standard techniques.

A note on notation: throughout this section, when we use parameters n and N together, we implicitly assume $N = 2^n$. We switch back and forth between the two parameters based on which one is more natural in a given context.

8.3.1 Equivalences for PSPACE via Instance Checkers

Our equivalences for PSPACE follow from the existence of *instance checkers* [BK95, TV07] for PSPACE-complete languages. To establish our equivalences with respect to restricted circuit classes, we use a recent construction of $\mathsf{AC}^0[\oplus]$-computable instance checkers by Chen [Che23].

Below, we say that an oracle circuit $E^{(-)}(x,z)$ from BP-$\mathcal{C}$ makes *projection* queries if every query it makes to the oracle can be computed by a projection over the inputs (x,z). After gathering the answers of the oracles, the final output is computed by a $\mathcal{C}$ circuit over (x,z) and these oracle answers. We stress that any oracle circuit that makes projection queries is *non-adaptive*. When we say such a circuit is DLOGTIME-uniform, we mean that both the projection (computing the queries to the oracles) and the top $\mathcal{C}$ circuit are DLOGTIME-uniform.

Theorem 8.3.1 (A PSPACE-Complete Language with Useful Properties)**.** *There is a language $L^\star \subseteq \{0,1\}^*$ with the following properties:*

1. **(Complexity Upper Bound)** $L^\star \in \mathsf{PSPACE}$.
2. **(Completeness)** $L^\star$ *is* PSPACE*-hard under* DLOGTIME*-uniform projection reductions.*
3. **(Instance Checkability)** *There is a* DLOGTIME*-uniform family of* BP-$\mathsf{AC}^0[\oplus]$ *oracle circuits $\{\mathsf{IC}_n\}_{n\geq 1}$ making projection queries such that, on every input string $x \in \{0,1\}^n$ and for every oracle $\mathcal{O} \subseteq \{0,1\}^*$, the following holds:*
   - $\mathsf{IC}_n^{\mathcal{O}}(x)$ *only makes queries of length n to $\mathcal{O}$.*
   - *If $\mathcal{O}$ agrees with $L^\star$ on inputs of length n, then $\Pr_r[\mathsf{IC}_n^{\mathcal{O}}(x;r) = L^\star(x)] = 1$.*
   - *For every oracle $\mathcal{O}$, $\Pr_r[\mathsf{IC}_n^{\mathcal{O}}(x;r) \in \{\perp, L^\star(x)\}] \geq 1 - \exp(-n)$.*

Theorem 8.3.1 follows from [Che23, Section 7]; we refer the reader to Section 8.5 for more details.

We say that the $\mathcal{C}$-`Implicit`-δ-`Heavy-Avoid` problem corresponding to a given family $\{G_N\}$ of implicitly computed maps can be solved in space $s(N)$ if there is an algorithm A of space complexity $s(N)$ such that, for every input length N, there is some $x \in \{0,1\}^N$ for which $A(1^N, i)$ outputs the i-th bit of x for all $i \in [N]$, and x is a solution to the $\mathcal{C}$-`Implicit`-δ-`Heavy-Avoid` problem for G_N.

Theorem 8.3.2 (Equivalence for PSPACE)**.** *Let $\mathcal{C}$ be a nice class of Boolean circuits. The following statements are equivalent:*

(*i*) PSPACE $\not\subseteq$ BP-$\mathcal{C}$.

(*ii*) *For every choice of $c, d, \ell \in \mathbb{N}$, with $m(N) = n^d$ and $\delta(N) = 1/n^\ell$, and for every sequence $\{G_N\}$ of maps $G_N \colon \{0,1\}^{m(N)} \to \{0,1\}^N$ implicitly computed by* DLOGTIME*-uniform $\mathcal{C}$-circuits of size at most n^c, the corresponding $\mathcal{C}$-`Implicit`-δ-`Heavy-Avoid` problem can be solved in space $O(\log N)$ on infinitely many input lengths N.*

Proof. We consider each implication below.

(*ii*) $\Rightarrow$ (*i*). Using the assumption, we show below that for every choice of $k \geq 1$, there is $L \in \mathsf{DSPACE}[n^2]$ such that L cannot be computed by $\mathsf{DTIME}[k \cdot \log n]$-uniform randomised $\mathcal{C}$-circuits of size n^k.[13] Since there exist PSPACE-complete problems, a standard argument shows that this implies PSPACE $\not\subseteq$ BP-$\mathcal{C}$.

[13] If $\mathcal{C}$ is a constant-depth circuit class defined as a union of classes for each fixed depth k, the argument can be adapted accordingly.

Fix a large enough $k \geq 1$, and consider the map $G_N \colon \{0,1\}^{m(N)} \to \{0,1\}^N$ defined as follows, where $m(N) := n^{3k}$. The map G_N views its input string x as a pair (M, r), where M is the description of a clocked deterministic machine running in time $10k \cdot \log n$, and r is the rest of x, treated as a random string. We assume that this encoding satisfies that for a random x, every machine M of description length ℓ occurs with probability $\Theta(2^{-\ell}/\ell^2)$ (this is possible since $\sum_{\ell \geq 1} \frac{1}{\ell^2}$ is bounded). The important part is that if the description length of M is a constant, then it occurs with constant probability. Let $D_M \colon \{0,1\}^{n^{2k}} \times \{0,1\}^n \to \{0,1\}$ be the $\mathcal{C}$-circuit of size at most n^{2k} encoded by the machine $M(1^n, \cdot)$ (i.e., we assume that M computes the direct connection language of D_M). For $i \in \{0,1\}^n$, we define the i-th output bit of $G_N(x)$ as $D_M(r, i)$. Note that a uniform computation over inputs of length at most $10k \cdot \log n$ and running in time $10k \cdot \log n$ can be uniformly converted into an AC^0 circuit of size at most n^{10k}. Since $\mathcal{C}$ contains AC^0, admits universal circuits, and is closed under composition, G_N can be implicitly computed by a $\mathsf{DLOGTIME}$-uniform probabilistic $\mathcal{C}$-circuit C_N defined over $m(N) + n$ input bits and of size at most $n^{C \cdot k}$, where C is a large enough universal constant that depends only on the circuit class $\mathcal{C}$.

Let $B(1^N)$ be an algorithm of space complexity $O(\log N)$ that solves $\mathcal{C}$-$\mathtt{Implicit}$-δ-$\mathtt{Heavy}$-$\mathtt{Avoid}$ on infinitely many values of N for the sequence G_N, with parameters $c, d \leq C \cdot k$ and function $\delta(N) \leq o(1)$. Let L_B be the language defined by B, i.e., a string $z \in \{0,1\}^n$ is in L_B if and only if the z-th bit of $B(1^N)$ (with $N = 2^n$) is 1. Note that L_B is in $\mathsf{DSPACE}[O(n)]$.

We now argue that L_B cannot be computed by $\mathsf{DTIME}[k \cdot \log n]$-uniform randomised $\mathcal{C}$-circuits of size n^k. To prove this, it is enough to show that for every language L computed by such circuits, each string in the sequence $\{y_N^L\}_N$ of truth-tables obtained from L is δ-heavy in $G_N(\mathcal{U}_{m(N)})$ for every large enough N. Under this claim, since B solves $\mathcal{C}$-$\mathtt{Implicit}$-δ-$\mathtt{Heavy}$-$\mathtt{Avoid}$ for the sequence $\{G_N\}$, it follows that $L_B \neq L$.

To see that the claim holds, recall that L is computed by $\mathsf{DTIME}[k \cdot \log n]$-uniform randomised $\mathcal{C}$-circuits of size n^k. Consequently, there is a deterministic machine M_L that runs in time $k \cdot \log n$ and decides the direct connection language of a corresponding randomised $\mathcal{C}$-circuit D_L of size at most n^{2k} and using at most n^k random bits that computes L on n-bit inputs. We now boost the success probability of the circuit D_L via repetition and (approximate) majority vote. More precisely, since the approximate majority function can be computed by $\mathsf{DLOGTIME}$-uniform AC^0 circuits [Ajt90, Vio09], $\mathcal{C}$ contains AC^0, and $\mathcal{C}$ is closed under composition, there is a deterministic machine $\widetilde{M}_L$ that runs in time $10k \cdot \log n$ and decides the direct connection language of a corresponding randomised $\mathcal{C}$-circuit $\widetilde{D}_L$ of size at most $n^{C \cdot k}$ and using at most n^{2k} random bits that computes L on each n-bit input string with probability at least $1 - 2^{-2n}$. Moreover, we can assume that the description length of $\widetilde{M}_L$ is a constant ℓ_{desc}, hence it occurs with constant probability. Let y_N^L be the truth-table of L on input length n, i.e., $|y_N^L| = N = 2^n$. By construction, using an union bound over all n-bit input strings, the probability that $G_N(\mathcal{U}_{m(N)}) = y_N^L$ is at least $\Omega(2^{-\ell_{\mathsf{desc}}}/\ell_{\mathsf{desc}}^2) \cdot (1 - 2^{-n}) \geq \Omega(1) > \delta$, for large enough N. This shows that y_N^L is δ-heavy, concluding the proof of this item.

$(i) \Rightarrow (ii)$. We argue in the contrapositive. In other words, suppose that there is a choice of constants c, d, and ℓ, with $m(N) = n^d$ and $\delta(N) = 1/n^\ell$, and a sequence $G_N \colon \{0,1\}^{m(N)} \to \{0,1\}^N$ implicitly computed by $\mathsf{DLOGTIME}$-uniform $\mathcal{C}$-circuits of size n^c such that every algorithm $A(1^N)$

running in space $O(\log N)$ time fails to solve $\mathcal{C}$-`Implicit-`δ-`Heavy-Avoid` on every large enough input length N. Next, we use this assumption to establish that $\mathsf{PSPACE} \subseteq \mathsf{BP}\text{-}\mathcal{C}$, which concludes the proof.

We consider a candidate algorithm $A(1^N)$ that computes as follows. Consider the language $L^\star$ from Theorem 8.3.1, and assume that $L^\star \in \mathsf{DSPACE}[n^a]$, where $a \in \mathbb{N}$. For a given $N' = 2^{n'}$, we let $\mathsf{tt}^\star_{N'} \in \{0,1\}^{N'}$ denote the truth table of $L^\star$ over inputs of length n'. On input 1^N, algorithm A outputs the string $y_N = \mathsf{tt}^\star_{N'} 0^{u_N} \in \{0,1\}^N$, where $N' = 2^{n'}$ for $n' = n^{1/a}$, and $u_N = N - N'$.

Note that A computes in space $O(\log N)$, due to our choice of parameters. Therefore, A fails to solve $\mathcal{C}$-`Implicit-`δ-`Heavy-Avoid` on every large enough input length N. This means that for every large enough N the probability of y_N under $G_N(\mathcal{U}_{m(N)})$ is at least $\delta = 1/n^\ell$.

We first describe a randomised algorithm that computes $L^\star$, deferring for now a discussion of its correctness, circuit complexity, and uniformity. Let $n = (n')^a$, as above. To compute $L^\star$ on a given input x of length $n' \in \mathbb{N}$, we sample $v = n^{3\ell}$ strings $z_1, \ldots, z_v \in \{0,1\}^{m(N)}$ uniformly and independently at random, and use the $2^{n'}$-bit prefixes $\mathcal{O}'_1, \ldots, \mathcal{O}'_v$ of the corresponding oracles $\mathcal{O}_1, \ldots, \mathcal{O}_v$ as candidate oracles for $L^\star$ on input length n', where each $\mathcal{O}_i$ is the oracle associated with the string $G_N(z_i) \in \{0,1\}^N$. In more detail, let $b_i = \mathsf{IC}^{\mathcal{O}'_i}_{n'}(x)$, where $\mathsf{IC}_{n'}$ is the algorithm from Theorem 8.3.1. We output 1 if at least one bit among $b_1, \ldots, b_v$ is 1, and 0 otherwise.

Next, we argue that A computes $L^\star$ with high probability. Consider an arbitrary input length n' and a given input string $x \in \{0,1\}^{n'}$. By our choice of v, with high probability the string y_N appears among the strings $G_N(z_1), \ldots, G_N(z_v)$. In particular, with high probability the truth table $\mathsf{tt}^\star_{N'}$ appears as an N'-bit prefix of one of these strings, meaning that one of the oracles $\mathcal{O}'_i$ computes $L^\star$ on inputs of length n'. Consequently, in this case, if $L^\star(x) = 1$ then at least one bit $b_i = 1$, and the procedure outputs 1. On the other hand, if $L^\star(x) = 0$, then by a union bound over the internal randomness of $\mathsf{IC}_{n'}$, with high probability every bit $b_i \in \{0, \bot\}$. In this case, the procedure outputs 0. This establishes the correctness of A.

It remains to establish an upper bound on the circuit complexity of A and to analyse the uniformity of the corresponding circuits. Note that each bit $b_i \in \{0, 1, \bot\}$ can be computed by a randomised $\mathcal{C}$-circuit of polynomial size, since G_N is implicitly computed by $\mathcal{C}$-circuits of polynomial size, $\mathsf{IC}_{n'}$ is computable by randomised $\mathcal{C}$-circuits of polynomial size, and $\mathcal{C}$ is closed under composition. Moreover, the disjunction of the bits b_i can also be computed in $\mathcal{C}$, since this class contains $\mathsf{AC}^0[\oplus]$. Therefore, A can be implemented by randomised $\mathcal{C}$-circuits of polynomial size. Finally, it is not hard to check that the corresponding sequence of randomised $\mathcal{C}$-circuits is $\mathsf{DLOGTIME}$ uniform, since IC is computed by $\mathsf{DLOGTIME}$-uniform randomised circuits, and G_N is implicitly computed by $\mathsf{DLOGTIME}$-uniform circuits.

The above discussion implies that $L^\star \in \mathsf{BP}\text{-}\mathcal{C}$. Since $L^\star$ is complete under $\mathsf{DLOGTIME}$-uniform projection reductions, we get that $\mathsf{PSPACE} \subseteq \mathsf{BP}\text{-}\mathcal{C}$, as desired. □

Our characterisations also extend to *almost-everywhere* lower bounds and *subexponential* lower bounds, as demonstrated in the following theorems.

Theorem 8.3.3. *Let $\mathcal{C}$ be a nice class of Boolean circuits. The following statements are equivalent:*

(i) $\mathsf{PSPACE} \not\subseteq \text{i.o.-}\mathsf{BP}\text{-}\mathcal{C}$.

(*ii*) *For every choice of* $c, d, \ell \in \mathbb{N}$*, with* $m(N) = n^d$ *and* $\delta(N) = 1/n^\ell$*, and for every sequence* $\{G_N\}$ *of maps* $G_N \colon \{0,1\}^{m(N)} \to \{0,1\}^N$ *implicitly computed by* DLOGTIME*-uniform* $\mathcal{C}$*-circuits of size at most* n^c*, the corresponding* $\mathcal{C}$-`Implicit`-δ-`Heavy-Avoid` *problem can be solved in* $O(\log(N))$ *space* for all large enough N.

Proof Sketch. The result follows from the same argument given for Theorem 8.3.2:

- $(ii) \Rightarrow (i)$: If our algorithm B is correct on input 1^N, then our language L is hard on input length n.
- $(i) \Rightarrow (ii)$: If the language $L^\star$ is hard on input length n', then our algorithm A is correct on input 1^N where $N = 2^{(n')^a}$. □

We use BP-$\mathcal{C}$-SIZE$[f(n)]$ to denote the class of languages computable by DLOGTIME-uniform BP-$\mathcal{C}$ circuits of size $f(n)$.

Theorem 8.3.4. *Let* $\mathcal{C}$ *be a nice class of Boolean circuits. The following statements are equivalent:*

(*i*) *There is a constant* $\varepsilon > 0$ *such that* PSPACE $\not\subseteq$ BP-$\mathcal{C}$-SIZE$[2^{n^\varepsilon}]$.

(*ii*) *There is a constant* $\varepsilon > 0$ *such that for* $\delta(N) := 2^{-n^\varepsilon}$ *and for every sequence* $\{G_N\}$ *of maps* $G_N \colon \{0,1\}^{2^{n^\varepsilon}} \to \{0,1\}^N$ *implicitly computed by* DLOGTIME*-uniform* $\mathcal{C}$*-circuits of size* 2^{n^ε}*, the corresponding* $\mathcal{C}$-`Implicit`-δ-`Heavy-Avoid` *problem can be solved in* $\mathrm{poly}(N)$ *time on infinitely many input lengths* N.

Proof Sketch. The argument is an adaptation of the proof of Theorem 8.3.2 by adjusting a few parameters, so we refer the reader to that proof for more details.

To see that $(ii) \Rightarrow (i)$ holds, let $\varepsilon' := \varepsilon/4$, and consider the map $G_N \colon \{0,1\}^{m(N)} \to \{0,1\}^N$ where $m(N) := 2^{n^{2\varepsilon'}}$ and the input bits are parsed into the description of a Turing machine M that encodes a size-$2^{n^{\varepsilon'}}$ $\mathcal{C}$-circuit D_M and the rest random inputs (fed to D_M). Like in Theorem 8.3.2, we assume that every constant-size Turing machine occurs with constant probability. This map G_N can be implicitly computed by a DLOGTIME-uniform probabilistic $\mathcal{C}$-circuit C_N of size 2^{n^ε}, in the sense that for every $x \in \{0,1\}^{m(N)}$ and $i \in [N]$, the i-th bit of $G_N(x)$ is equal to $C_N(x, i)$. We can see that for every language $L \in$ BP-$\mathcal{C}$-SIZE$[2^{n^{\varepsilon'}}]$, let y_N^L denote the truth table of L_n, then the probability that $G_N(\mathcal{U}_{m(N)}) = y_N^L$ is at least a constant. Hence, given an algorithm $B(1^N)$ that solves the $\mathcal{C}$-`Implicit`-δ-`Heavy-Avoid` problem for $\delta = o(1)$ (on infinitely many N), the language whose truth table is the output of $B(1^N)$ is not in BP-$\mathcal{C}$-SIZE$[2^{n^{\varepsilon'}}]$ (on infinitely many n). Since we further assumed that B runs in space $O(\log N)$, we obtain a hard language in SPACE$[O(\log N)]$ = SPACE$[O(n)]$ that is not in BP-$\mathcal{C}$-SIZE$[2^{n^{\varepsilon'}}]$.

To see that $(i) \Rightarrow (ii)$ holds, let $L^\star$ denote the PSPACE-complete language in Theorem 8.3.1 and let $\varepsilon' := \varepsilon/(5a)$, where $a \geq 1$ is a constant such that $L^\star \in$ SPACE$[n^a]$. Consider the following algorithm $A(1^N)$ for solving the $\mathcal{C}$-`Implicit`-δ-`Heavy-Avoid` problem with parameter ε'. On input 1^N, let $n' := n^{1/a}$, $N' := 2^{n'}$, $y_{n'} \in \{0,1\}^{N'}$ be the truth table of $L^\star$ on input length n', then A outputs $y_{n'}0^{N-N'} \in \{0,1\}^N$. If A fails to solve $\mathcal{C}$-`Implicit`-δ-`Heavy-Avoid`, then we can compute $L^\star$ in BP-$\mathcal{C}$-SIZE$[2^{n^\varepsilon}]$ as follows. Let $x \in \{0,1\}^{n'}$ be an instance of $L^\star$, we set $n := (n')^a$ and $N' := 2^{n'}$. We sample $v := (1/\delta(N))^3 \leq 2^{3n^\varepsilon}$ strings $z_1, \ldots, z_v \in \{0,1\}^{2^{n^\varepsilon}}$ uniformly and

independently at random, and for each string z_i we define an oracle $\mathcal{O}_i \colon \{0,1\}^{n'} \to \{0,1\}$ whose truth table is the first N' bits of $G_N(z_i)$. We then run IC against each $\mathcal{O}_i$ and obtain $b_i := \mathsf{IC}^{\mathcal{O}_i}_{n'}(x)$ for each $i \in [v]$, and finally we output 1 if some b_i is equal to 1. This algorithm computes $L^\star$ because with high probability, the truth table $y_{n'}$ appears in these oracles, and also the instance checker will never output $1 - L(x)$ by mistake. Our algorithm can be implemented in $\mathsf{BP}\text{-}\mathcal{C}\text{-}\mathsf{SIZE}[2^{n^{5\varepsilon'}}] \subseteq \mathsf{BP}\text{-}\mathcal{C}\text{-}\mathsf{SIZE}[2^{(n')^{\varepsilon}}]$. □

Remark 8.3.5. In the proof of Theorem 8.3.2, we only need to solve the $\mathcal{C}$-`Implicit-`δ-`Heavy-Avoid` problem for $\delta = o(1)$ to obtain the lower bound (i.e., Item (i)), while the latter implies algorithms for the $\mathcal{C}$-`Implicit-`δ-`Heavy-Avoid` problem even when $\delta = 1/\mathrm{poly}(n) = 1/\mathrm{polylog}(N)$. This illustrates the *robustness* of the parameter δ in $\mathcal{C}$-`Implicit-`δ-`Heavy-Avoid` with respect to $O(\log N)$-space algorithms: if the problem is solvable for $\delta = o(1)$, then it is also solvable for $\delta = 1/\mathrm{polylog}(N)$. Similarly, Theorem 8.3.4 shows that if we consider implicit maps computable in the 2^{n^ε} time regime, then this problem is solvable for $\delta = o(1)$ if and only if it is solvable for $\delta = 2^{-n^\varepsilon}$. In fact, it is evident from the proofs that the robustness of the parameter δ holds in every characterisation result in Section 8.3.

8.3.2 Equivalences for NP via Search-to-Decision Reductions

In this section, we show equivalences between uniform randomised lower bounds for NP and Heavy Avoid algorithms implementable by constant-depth circuits. NP is not known to be instance-checkable, hence we cannot use the technique from the previous section. However, it turns out that search-to-decision reductions can also be used to argue the desired equivalences. The standard search-to-decision reduction is highly sequential, so in order to show equivalences that work for any nice circuit class, we use a depth-efficient version based on the Valiant-Vazirani Isolation Lemma [VV86] which exploits our access to randomness.

We first need a generalisation of the standard result that $\mathsf{NP} \not\subseteq \mathsf{BPP}$ iff $\mathsf{PH} \not\subseteq \mathsf{BPP}$.

Lemma 8.3.6. *Let $\mathcal{C}$ be a nice circuit class.* $\mathsf{NP} \not\subseteq \mathsf{BP}\text{-}\mathcal{C}$ *if and only if* $\mathsf{PH} \not\subseteq \mathsf{BP}\text{-}\mathcal{C}$.

Proof Sketch. The proof is essentially the same inductive argument as for the standard equivalence between $\mathsf{NP} \not\subseteq \mathsf{BPP}$ and $\mathsf{PH} \not\subseteq \mathsf{BPP}$. We must show that if $\mathsf{NP} \subseteq \mathsf{BPP}$ then $\mathsf{PH} \subseteq \mathsf{BPP}$. In order to implement that argument, we need to be able to do error reduction to exponentially small error by $\mathsf{DLOGTIME}$-uniform randomised $\mathcal{C}$ circuits, which holds since $\mathcal{C}$ contains AC^0, and Approximate Majority can be computed in $\mathsf{DLOGTIME}$-uniform AC^0 [Ajt90, Vio09]. We also need the closure of $\mathcal{C}$ under composition to be able to do induction, but that also holds since $\mathcal{C}$ is nice. □

Now we proceed to our equivalences for NP.

Theorem 8.3.7 (Equivalences for NP)**.** *Let $\mathcal{C}$ be a nice circuit class. The following statements are equivalent:*

(i) $\mathsf{NP} \not\subseteq \mathsf{BP}\text{-}\mathcal{C}$.

(ii) There are positive integers k and r such that for every $c, d, \ell \in \mathbb{N}$, with $m(N) = n^d$ and $\delta(N) = 1/n^\ell$, and for every sequence $\{G_N\}$ of maps $G_N \colon \{0,1\}^{m(N)} \to \{0,1\}^N$ implicitly computed by $\mathsf{DLOGTIME}$*-uniform $\mathcal{C}$-circuits of size at most n^c, the corresponding*

$\mathcal{C}$-`Implicit`-δ-`Heavy-Avoid` *problem can be solved by* DLOGTIME*-uniform unbounded fan-in circuits of size* $2^{\log(N)^r}$ *and depth* k *on infinitely many input lengths* N*.*

Proof. We consider each implication below.

$(ii) \Rightarrow (i)$. We will use the assumption to show that $\mathsf{PH} \not\subseteq \mathsf{BP}\text{-}\mathcal{C}$, and the desired implication then follows from Lemma 8.3.6 and the assumption that $\mathcal{C}$ is nice.

As in the proof of the analogous equivalence for PSPACE, fix a large enough $a \geq 1$, and consider the map $G_N \colon \{0,1\}^{m(N)} \to \{0,1\}^N$ defined as follows, where $m(N) := n^{3a}$. The map G_N parses its input string x into (M, r), where M is the description of a clocked deterministic machine running in time $10a \cdot \log n$, and r consists of the remaining bits of x, treated as randomness. Let $D_M \colon \{0,1\}^{n^{2a}} \times \{0,1\}^n \to \{0,1\}$ be the randomised $\mathcal{C}$ circuit of size at most n^{2a} encoded by the machine $M(1^n, \cdot)$ (i.e., we assume that M computes the direct connection language of D_M). For $i \in \{0,1\}^n$, we define the i-th output bit of $G_N(x)$ as $D_M(r, i)$. Note that G_N can be implicitly computed by a DLOGTIME-uniform randomised $\mathcal{C}$ circuit C_N defined over $m(N) + n$ input bits and of size at most $n^{C \cdot a}$, where C is a large enough universal constant.

As per assumption, let $\{C_N\}$ be a DLOGTIME-uniform family of unbounded fan-in circuits of size $2^{\log(N)^r}$ and depth k such that for infinitely many N, C_N solves $\mathcal{C}$-`Implicit`-δ-`Heavy-Avoid` on G_N, with parameters $c = C \cdot a$, $d = 2a + 1$, and $\delta(N) = o(1)$. (Recall that each output bit of G_N is computed in time n^c, $m(N) \leq n^d$, and we want to find a $\delta(N)$-light element.) Let L be the language defined by $\{C_N\}$, i.e., a string $z \in \{0,1\}^n$ is in L if and only if the z-th bit of $C_N(1^N)$ (with $N = 2^n$) is 1. Note that there are integers s and k' (depending only on r and k) such that L is in $\Sigma_{k'}$-TIME$[n^s]$ by the known equivalence [BIS90] between PH and DLOGTIME-uniform circuits of exponential size in n (which is quasi-polynomial size in N).

We now argue that L cannot be computed by DTIME$[a \cdot \log n]$-uniform randomised $\mathcal{C}$ circuits of size n^a. To prove this, it is enough to show that for every language L' computed by such circuits, each string in the sequence $\{y_N^{L'}\}_N$ of truth-tables obtained from L' is δ-heavy in $G_N(\mathcal{U}_{m(N)})$ for every large enough N. Under this claim, as B solves $\mathcal{C}$-`Implicit`-δ-`Heavy-Avoid` for the sequence $\{G_N\}$, it follows that $L \neq L'$.

To see that the claim holds, suppose that L' is computed by DTIME$[a \cdot \log n]$-uniform randomised $\mathcal{C}$ circuits $D_{L'}$ of size n^a. Consequently, there is a deterministic machine $M_{L'}$ that runs in time $a \cdot \log n$ and decides the direct connection language of a corresponding randomised $\mathcal{C}$ circuit $D_{L'}$, and the circuit $D_{L'}$ computes L' on n-bit inputs, has size at most n^{2a} and uses at most n^a random bits. We can then reduce the error of the circuit D_L to be exponentially small by using the facts that Approximate Majority is in DLOGTIME-uniform AC^0 [Ajt90, Vio09] and that $\mathcal{C}$ is nice. Thus we obtain a family of randomised Boolean circuits $\widetilde{D}_L$ that has size at most $n^{C \cdot a}$, uses at most n^{2a} random bits, and computes L on each n-bit input string with probability at least $1 - 2^{-2n}$. Moreover, there is a deterministic machine $\widetilde{N}_{L'}$ that runs in time $10a \cdot \log n$ and decides the direct connection language of $\widetilde{D}_L$. Since the description length of $\widetilde{N}_{L'}$ is constant, it occurs with constant probability in the distribution sampled by G_N. Let $y_N^{L'}$ be the truth-table of L' on input length n, i.e., $|y_N^{L'}| = N = 2^n$. By construction, using an union bound over all n-bit input strings, the probability that $G_N(\mathcal{U}_{m(N)}) = y_N^{L'}$ is at least $\Omega(1) \cdot (1 - 2^{-n}) \geq \delta(N)$. This shows that $y_N^{L'}$ is δ-heavy, concluding the proof of the claim.

Note that the language L defined above depends on a, however by the standard fact that there is a language L_{comp} complete for $\Sigma_{k'}\text{-}\mathsf{TIME}[n^s]$ under $\mathsf{DLOGTIME}$-uniform projections of linear size, we get that that for each a, L_{comp} is not computed by $\mathsf{DTIME}[a \cdot \log n]$-uniform randomised $\mathcal{C}$ circuits of size n^a. This implies that $L_{\text{comp}} \notin \mathsf{BP}\mathcal{C}$, and hence that $\mathsf{PH} \nsubseteq \mathsf{BP}\mathcal{C}$. Therefore $\mathsf{NP} \nsubseteq \mathsf{BP}\mathcal{C}$ by Lemma 8.3.6, concluding the proof of this item.

$(i) \Rightarrow (ii)$. We argue in the contrapositive. Suppose that there is a choice of constants c,d, and ℓ, with $m(N) = n^d$ and $\delta(N) = 1/n^\ell$, and a sequence $G_N \colon \{0,1\}^{m(N)} \to \{0,1\}^N$ implicitly computed by $\mathsf{DLOGTIME}$-uniform $\mathcal{C}$ circuits of size n^c such that every $\mathsf{DLOGTIME}$-uniform sequence of unbounded fan-in circuits of size $2^{\log(N)^2}$ and depth 3 fails to solve $\mathcal{C}$-$\mathtt{Implicit}$-δ-$\mathtt{Heavy\text{-}Avoid}$ on every large enough input length N. We use this assumption to establish that $\mathsf{NP} \subseteq \mathsf{BP}\mathcal{C}$, which concludes the proof.

We consider a candidate algorithm $A(1^N)$ that simply outputs tt_N, where tt_N is the truth table of SAT on n-bit inputs. We consider a standard encoding of SAT in which SAT is depth-efficiently paddable, i.e., there is an algorithm Pad implemented by $\mathsf{DLOGTIME}$-uniform AC^0 circuits which, given as inputs 1^t for a positive integer t and a formula ϕ of length at most t, outputs an equisatisfiable formula ϕ' of length t. Note that A can be implemented by $\mathsf{DLOGTIME}$-uniform unbounded fan-in circuits of size $2^{\log(N)^2}$ and depth 3, using the known simulation of non-deterministic quasi-linear time by uniform unbounded fan-in circuits [BIS90]. By assumption, A fails to solve $\mathcal{C}$-$\mathtt{Implicit}$-δ-$\mathtt{Heavy\text{-}Avoid}$ on every large enough input length N. We show how to use this failure together with a depth-efficient randomised search-to-decision reduction based on the Valiant-Vazirani Isolation Lemma [VV86] and depth-efficient paddability of SAT to solve SAT in $\mathsf{BP}\mathcal{C}$, which implies $\mathsf{NP} \subseteq \mathsf{BP}\mathcal{C}$ by the NP-completeness of SAT with respect to $\mathsf{DLOGTIME}$-uniform projections.

By the failure of A, we have that for every large enough N, the probability of tt_N under $G_N(\mathcal{U}_{m(N)})$ is at least $\delta = 1/n^\ell$. First, we describe a polynomial-time randomised algorithm B to solve SAT. Let ϕ be a length-n input to SAT. For some $T = \text{quasipoly}(N)$ and $t = \log T$ to be determined later, we sample $v := t^{5\ell}$ strings $z_1, \ldots, z_v \in \{0,1\}^{m(T)}$ uniformly and independently at random, and for each string z_i, we define an oracle $\mathcal{O}_i$ whose truth table is the string $G_T(z_i) \in \{0,1\}^T$. For each $i \leq v$, we try to use $\mathcal{O}_i$ and a depth-efficient search-to-decision reduction to find a satisfying assignment to ϕ, as follows. Assume without loss of generality that ϕ has n variables. We do the following for each i in parallel. We check if $\mathcal{O}_i(\mathsf{pad}(1^t, \phi)) = 1$. If this is not the case for any i, we reject. If $\mathcal{O}_i$ does evaluate to 1 on the padded version of ϕ, we use this oracle to find a candidate satisfying assignment w to ϕ as follows. The idea is to use the Valiant-Vazirani technique of intersecting the solution space of ϕ with k randomly chosen hyperplanes for $k = 1 \ldots n$ to obtain formulas $\phi_1, \ldots, \phi_n$. The Valiant-Vazirani Isolation Lemma [VV86] states that if ϕ is satisfiable, then with probability at least $1/4n$ over random choices of these formulas, some ϕ_j has a unique solution. Note that each ϕ_j can be constructed from ϕ by randomised constant-depth circuits. We would like to use O_i to find and check the unique satisfying assignment so that we can verify that ϕ is satisfiable. An issue is that the ϕ_j are in general of size larger than n, but they are still of size $\text{poly}(n)$ and we choose t a large enough polynomial in n so that they can all be padded to length t. For each ϕ_j and each of the n original variables x_k in ϕ, we use an oracle call to O_i (using padding if necessary) to determine

if there is a satisfying assignment to ϕ with the variable x_k set to 0. If yes, we set the wire $b_{j,k}$ to 1, else to 0. We check if there is a j such that the assignment $x_k = b_{j,k}$ for each k satisfies ϕ. If this is the case for some i, we accept; otherwise we reject. Note that all of the above can be implemented in constant-depth, apart from the oracle calls to O_i, which we simulate by evaluations of the implicit sampler for O_i.

By the niceness of $\mathcal{C}$, specifically the assumptions that $\mathsf{AC}^0[\oplus]$ is contained in $\mathcal{C}$ and $\mathcal{C}$ is closed under composition, as well as the fact that our sampler is implicitly computed by uniform $\mathcal{C}$ circuits, there are $\mathsf{DLOGTIME}$-uniform randomised $\mathcal{C}$ circuits of polynomial size implementing the procedure above. We need to argue that these circuits correctly solve SAT with high probability. Note that a circuit only accepts a formula ϕ if it verifies that some assignment satisfies ϕ. Hence, it suffices to check that any satisfiable ϕ is accepted with high probability. By our choice of v, with high probability the string tt_T appears with multiplicity $\omega(n)$ among the strings $G_T(z_1), \ldots, G_T(z_v)$, meaning that $\omega(n)$ of the oracles $\mathcal{O}_i$ compute SAT on inputs of length t. By the correctness of the paddability procedure, for all of these correct oracles, satisfiability questions about the randomised formulas ϕ_j are all answered correctly. This, together with the lower bound on probability that one of the ϕ_j is uniquely satisfiable, implies that with probability at least $1 - o(1)$, oracle calls to some oracle $\mathcal{O}_i$ yield a satisfying assignment for ϕ, which is then correctly verified and results in acceptance of the circuit. □

8.3.3 Equivalences for EXP and $\mathsf{EXP}^{\mathsf{NP}}$ via a Win-Win Argument and Selectors

In this section, we generalise our equivalence results to the classes EXP and $\mathsf{EXP}^{\mathsf{NP}}$.

We provide two proofs. The first proof uses a win-win argument and relies on the existing equivalence result for PSPACE (Theorem 8.3.2). The second proof uses *selectors* for $\mathsf{EXP}^{\mathsf{NP}}$-complete languages [Hir15] or *instance checkers* for EXP-complete languages [BFL91]. Each proof has its advantages and disadvantages, as will be discussed in Remark 8.3.15.

We start with the first proof. We first state the equivalence result for $\mathsf{EXP}^{\mathsf{NP}}$.

Theorem 8.3.8 (Equivalence for $\mathsf{EXP}^{\mathsf{NP}}$)**.** *Let $\mathcal{C}$ be a nice class of Boolean circuits. The following statements are equivalent:*

(*i*) $\mathsf{EXP}^{\mathsf{NP}} \not\subseteq \mathsf{BP}\text{-}\mathcal{C}$.

(*ii*) *For every choice of $c, d, \ell \in \mathbb{N}$, with $m(N) = n^d$ and $\delta(N) = 1/n^\ell$, and for every sequence $\{G_N\}$ of maps $G_N \colon \{0,1\}^{m(N)} \to \{0,1\}^N$ implicitly computed by $\mathsf{DLOGTIME}$-uniform $\mathcal{C}$-circuits of size at most n^c, the corresponding $\mathcal{C}$-$\mathtt{Implicit}$-δ-$\mathtt{Heavy\text{-}Avoid}$ problem can be solved in deterministic time $\mathrm{poly}(N)$ with access to an NP oracle on infinitely many input lengths N.*

Proof. We consider each implication below.

(*ii*) $\Rightarrow$ (*i*). The proof is completely analogous to the implication from (*ii*) to (*i*) in Theorem 8.3.2. The only difference is that due to the access to an NP oracle provided to each deterministic polynomial-time algorithm for $\mathtt{Implicit}$-δ-$\mathtt{Heavy\text{-}Avoid}$, the resulting hard language is in $\mathsf{EXP}^{\mathsf{NP}}$ as opposed to PSPACE.

$(i) \Rightarrow (ii)$. If $\mathsf{EXP}^{\mathsf{NP}} \not\subseteq \mathsf{BP}\text{-}\mathcal{C}$ then either $\mathsf{EXP}^{\mathsf{NP}} \not\subseteq \mathsf{PSPACE}$ or $\mathsf{PSPACE} \not\subseteq \mathsf{BP}\text{-}\mathcal{C}$. We show that the desired conclusion holds in each one of these cases.

First, assume that $\mathsf{EXP}^{\mathsf{NP}} \not\subseteq \mathsf{PSPACE}$. Recall that if $\mathsf{EXP}^{\mathsf{NP}} \subseteq \mathsf{SIZE}[\mathrm{poly}]$ then $\mathsf{EXP}^{\mathsf{NP}} = \mathsf{PSPACE}$ [BH92]. Therefore, it follows that $\mathsf{EXP}^{\mathsf{NP}} \not\subseteq \mathsf{SIZE}[\mathrm{poly}]$. In particular, there is a language $L \in \mathsf{DTIME}[2^{O(n)}]^{\mathsf{NP}}$ such that $L \notin \mathsf{SIZE}[n^k]$ for every choice of k. Now fix a choice of $c, d, \ell \in \mathbb{N}$, with $m(N) = n^d$ and $\delta(N) = 1/n^\ell$, and consider a sequence $\{G_N\}$ of maps $G_N \colon \{0,1\}^{m(N)} \to \{0,1\}^N$ implicitly computed by $\mathsf{DLOGTIME}$-uniform $\mathcal{C}$-circuits of size at most n^c. Consider the following algorithm A: On input 1^N, where $N = 2^n$, A outputs the N-bit string w_N corresponding to the truth-table of L on n-bit strings. Since $L \in \mathsf{DTIME}[2^{O(n)}]^{\mathsf{NP}}$, A computes in time $\mathrm{poly}(N)$ with access to an NP oracle. Moreover, since L is not computed by (general) Boolean circuits of size n^k on infinitely many input lengths, where k is an arbitrary constant, it follows that w_N is not in the range of G_N for infinitely many values of N. Otherwise, there would be a choice a for the seed of G_N such that $w_N = G_N(a)$, which yields a bounded-size circuit for the function encoded by w_N, given that G_N is implicitly computed by $\mathcal{C}$-circuits of bounded size. In particular, it follows that on infinitely many values of N, w_N is not δ-heavy for G_N, as desired.

Now consider the remaining case, i.e., assume that $\mathsf{PSPACE} \not\subseteq \mathsf{BP}\text{-}\mathcal{C}$. Then, by Theorem 8.3.2, we can solve the required $\mathcal{C}$-`Implicit`-δ-`Heavy-Avoid` problem in space $O(\log N)$ on infinitely many input lengths N. Since $O(\log N)$ space algorithms can be simulated by $\mathrm{poly}(N)$ time algorithm (not to mention that we have access to an NP oracle), the desired conclusion also holds in this case. □

It is easy to see that a similar equivalence result for EXP also holds. Actually, the proof is essentially the same as Theorem 8.3.8, with the only difference being that we use the Karp–Lipton theorem for EXP [KL80] instead of the one for $\mathsf{EXP}^{\mathsf{NP}}$ [BH92]. Hence, we only state the result and omit the proof here.

Theorem 8.3.9 (Equivalence for EXP)**.** *Let $\mathcal{C}$ be a nice class of Boolean circuits. The following statements are equivalent:*

- (i) $\mathsf{EXP} \not\subseteq \mathsf{BP}\text{-}\mathcal{C}$.
- (ii) *For every choice of $c, d, \ell \in \mathbb{N}$, with $m(N) = n^d$ and $\delta(N) = 1/n^\ell$, and for every sequence $\{G_N\}$ of maps $G_N \colon \{0,1\}^{m(N)} \to \{0,1\}^N$ implicitly computed by $\mathsf{DLOGTIME}$-uniform $\mathcal{C}$-circuits of size at most n^c, the corresponding $\mathcal{C}$-`Implicit`-δ-`Heavy-Avoid` problem can be solved in deterministic time $\mathrm{poly}(N)$ on infinitely many input lengths N.*

Next, we present another proof of Theorem 8.3.8 for the special case where $\mathcal{C}$ is the class of general Boolean circuits. The proof is direct and does not go through win-win arguments. In one sentence, we use *selectors* for $\mathsf{EXP}^{\mathsf{NP}}$-complete problems [Hir15] and note that the proof strategy from Theorem 8.3.2 extends to selectors.

Definition 8.3.10. We say that a probabilistic polynomial-time oracle machine S is a *selector* for a language $L \subseteq \{0,1\}^*$ if the following holds. Let $\mathcal{O}_1, \mathcal{O}_2 \subseteq \{0,1\}^*$ be arbitrary oracles. Then, for any input $x \in \{0,1\}^*$, if $L \in \{\mathcal{O}_1, \mathcal{O}_2\}$ then

$$\Pr_S[S^{\mathcal{O}_1,\mathcal{O}_2}(x) = L(x)] \geq 2/3.$$

It is possible to boost the success probability of the selector using standard techniques. In addition, [Hir15] proved that if a language L admits a selector, then it also admits a selector that succeeds when given access to polynomially many oracles, provided that at least one of them correctly computes L. These are summarised in the following result.

Theorem 8.3.11 ([Hir15])**.** *Every* $\mathsf{EXP}^{\mathsf{NP}}$*-complete language admits a selector. Moreover, there is a paddable* $\mathsf{EXP}^{\mathsf{NP}}$*-complete language* $L' \in \mathsf{DTIME}[2^{O(n)}]^{\mathsf{NP}}$*, a polynomial* q*, and a probabilistic polynomial-time oracle algorithm* S *such that the following conditions hold:*

- *For every* $n \geq 1$*,* $x \in \{0,1\}^n$*, and* $t \geq 1$*, if* $\mathcal{O}_1, \ldots, \mathcal{O}_t \subseteq \{0,1\}^*$ *and* $L' \in \{\mathcal{O}_1, \ldots, \mathcal{O}_t\}$*, then*
$$\Pr_S[S^{\mathcal{O}_1,\ldots,\mathcal{O}_t}(x, 1^t) = L'(x)] \geq 1 - 2^{-n}.$$
- *Every oracle query of* $S(x)$ *has length exactly* $q(n)$*. Consequently, it is enough to assume that the oracles* $\mathcal{O}_1, \ldots, \mathcal{O}_t \colon \{0,1\}^{q(n)} \to \{0,1\}$ *and that* $L'_{q(n)} \in \{\mathcal{O}_1, \ldots, \mathcal{O}_t\}$*, where* $L'_{q(n)} = L \cap \{0,1\}^{q(n)}$*.*

The "moreover" part of the result follows from the existence of a selector for every $\mathsf{EXP}^{\mathsf{NP}}$-complete problem [Hir15] combined with the paddability of L' and the discussion above.

Proof of Theorem 8.3.8, for the case that $\mathcal{C}$ *is the class of general Boolean circuits.*

$(ii) \Rightarrow (i)$. Again, the proof is completely analogous to the same implication in Theorem 8.3.2, and we omit the details.

$(i) \Rightarrow (ii)$. We argue in the contrapositive. Suppose there is a choice of constants c, d, and ℓ, with $m(N) = n^d$ and $\delta(N) = 1/n^\ell$, and a sequence $G_N \colon \{0,1\}^{m(N)} \to \{0,1\}^N$ implicitly computed by general Boolean circuits of size at most n^c such that every deterministic algorithm $A(1^N)$ with access to an NP oracle that runs in $\mathrm{poly}(N)$ time fails to solve $\texttt{Implicit-}\delta\texttt{-Heavy-Avoid}$ on every large enough input length N. Next, we use this assumption to establish that $\mathsf{EXP}^{\mathsf{NP}} \subseteq \mathsf{BPP}$.

We consider a candidate algorithm $A'(1^N)$ with access to an NP oracle that computes as follows. Consider the $\mathsf{EXP}^{\mathsf{NP}}$-complete language L' from Theorem 8.3.11. On input 1^N, A' outputs the truth-table $tt_N \in \{0,1\}^N$ of L' over strings of length n, where $N = 2^n$. Note that A' uses an NP oracle and computes in time $\mathrm{poly}(N)$, since $L' \in \mathsf{DTIME}[2^{O(n)}]^{\mathsf{NP}}$. Since A' fails to solve $\texttt{Implicit-}\delta\texttt{-Heavy-Avoid}$, for every large enough N, the probability of tt_N under $G_N(\mathcal{U}_{m(N)})$ is at least $\delta = 1/n^\ell$.

The rest of the argument is similar to that of the proof of Theorem 8.3.2. To compute L' on some input $x \in \{0,1\}^{n'}$, we let $n := q(n')$, where q is the polynomial from Theorem 8.3.11. We sample $t := n^{3\ell}$ strings $z_1, \ldots, z_t \in \{0,1\}^{m(N)}$ uniformly and independently at random, and use the corresponding oracles $\mathcal{O}_1, \ldots, \mathcal{O}_t$ as candidate oracles for L'_n, where each $\mathcal{O}_i$ computes according to the string $G_N(z_i) \in \{0,1\}^N$. By our choice of t, with high probability the string tt_N is among the oracles obtained from $z_1, \ldots, z_t$. In this case, there is at least one correct oracle $\mathcal{O}_i$ among the oracles $\mathcal{O}_1, \ldots, \mathcal{O}_t$. Consequently, if we run $S(x, 1^t)$ with access to $\mathcal{O}_1, \ldots, \mathcal{O}_t$, we compute $L'(x)$ with high probability. Since $n = \mathrm{poly}(n')$, $t = \mathrm{poly}(n)$, the selector runs in time $\mathrm{poly}(n', t)$, and the simulation of each oracle query to $\mathcal{O}_i$ can be done using a computation of the corresponding bit of $G_N(z_i)$ in time $\mathrm{poly}(n)$, it follows that given x of length n' we can

compute $L'(x)$ with high probability in time $\mathrm{poly}(n')$. Finally, since L' is complete for $\mathsf{EXP}^{\mathsf{NP}}$, it follows that $\mathsf{EXP}^{\mathsf{NP}} \subseteq \mathsf{BPP}$, as desired. □

Inspecting the proof, it is not hard to see that it also extends to *almost-everywhere* and *subexponential* lower bounds. Since the proofs are straightforward modifications of the argument given above, we only state the results and omit the proof details.

Theorem 8.3.12. *The following statements are equivalent:*

(*i*) $\mathsf{EXP}^{\mathsf{NP}} \not\subseteq$ i.o.-BPP.

(*ii*) *For every choice of the parameters $c, d, \ell \in \mathbb{N}$, with $m(N) = n^d$ and $\delta(N) = 1/n^\ell$, and for every sequence $\{G_N\}$ of maps $G_N \colon \{0,1\}^{m(N)} \to \{0,1\}^N$ implicitly computed in time n^c, the* `Implicit-`δ`-Heavy-Avoid` *problem can be solved with access to an* NP *oracle in deterministic time* $\mathrm{poly}(N)$ for every large enough N.

Theorem 8.3.13. *The following statements are equivalent:*

(*i*) *There is a constant $\varepsilon > 0$ such that $\mathsf{E}^{\mathsf{NP}} \not\subseteq \mathsf{BPTIME}[2^{n^\varepsilon}]$.*

(*ii*) *There is a constant $\varepsilon > 0$ such that for $\delta(N) := 2^{-n^\varepsilon}$ and for every sequence $\{G_N\}$ of maps $G_N \colon \{0,1\}^{2^{n^\varepsilon}} \to \{0,1\}^N$ implicitly computed in time 2^{n^ε}, the* `Implicit-`δ`-Heavy-Avoid` *problem on $\{G_N\}$ can be solved with access to an* NP *oracle in deterministic time* $\mathrm{poly}(N)$ *on infinitely many values of N.*

One can also use the instance checkers for EXP-complete languages [BFL91] (which imply selectors for such languages [Hir15]) to prove similar characterisations for EXP. For example, the following theorem holds (we omit the details as it is the same as our second proof for $\mathsf{EXP}^{\mathsf{NP}}$):

Theorem 8.3.14. *The following statements are equivalent:*

(*i*) $\mathsf{EXP} \not\subseteq$ i.o.-BPP.

(*ii*) *For every choice of the parameters $c, d, \ell \in \mathbb{N}$, with $m(N) = n^d$ and $\delta(N) = 1/n^\ell$, and for every sequence $\{G_N\}$ of maps $G_N \colon \{0,1\}^{m(N)} \to \{0,1\}^N$ implicitly computed in time n^c, the* `Implicit-`δ`-Heavy-Avoid` *problem can be solved in deterministic* $\mathrm{poly}(N)$ *time for every large enough N.*

Remark 8.3.15 (Comparison between the two proof methods). We presented two proofs for Theorem 8.3.8. Both proofs have their advantages and disadvantages, as we summarise below:

- The first proof only uses (non-adaptive) instance checkers [Che23] that have small circuit complexity overheads, hence it works for restricted circuit classes such as $\mathsf{AC}^0[\oplus]$ and ACC^0. On the other hand, the second proof needs to use the selectors for $\mathsf{EXP}^{\mathsf{NP}}$ [Hir15] which is highly adaptive, hence does not extend to smaller circuit classes such as $\mathcal{C} = \mathsf{NC}^1$ or $\mathcal{C} = \mathsf{TC}^0$.[a]
- The second proof proceeds by a direct argument and hence generalises to almost-everywhere and subexponential time lower bounds (Theorem 8.3.12 and Theorem 8.3.13), while the first proof uses a win-win analysis and does not seem to generalise to these cases.

[a] One can also construct non-adaptive instance checkers for EXP-complete languages from highly-efficient PCPs such as [BGH+06, BS08, BCGT13, BV14]. In fact, the results in [BV14] imply an instance checker whose circuit complexity is only a 3-CNF over its randomness r. However, it is unclear if that instance checker also has low circuit complexity over the input x. (Looking into the proof, it seems that one needs

to at least compute a Reed–Solomon encoding of x). Therefore, we chose to use the off-the-shelf $\mathsf{AC}^0[\oplus]$ instance checker in [Che23] for PSPACE and resort to a win-win argument for EXP. On the other hand, it is unclear to the authors whether the selectors in [Hir15] can be made non-adaptive.

8.4 Heavy Avoid and Derandomisation

In this section, we study the relation between algorithms for Heavy Avoid and derandomisation, with connections to recent developments in *instance-wise* hardness-randomness trade-offs [CT21a, LP22, Kor22, LP23, CTW23]. This section mainly considers Heavy Avoid for *non-uniformly* and *implicitly* sampled distributions.

8.4.1 A Non-Black-Box Reduction

We show that in some scenarios, solving the `Implicit-Heavy-Avoid` problem on non-uniform samplers implies general derandomisation of prBPP. Intriguingly, our reduction from prBPP to Heavy Avoid is *non-black-box* and relies on the *code* of an algorithm for `Implicit-Heavy-Avoid`.

We first introduce appropriate notation. We say that a Boolean circuit family $\{C_n\}$ has *dimension* $d(n) \times T(n)$ if for each $n \in \mathbb{N}$, the gates of C_n are partitioned into $d(n)$ layers, each layer contains at most $T(n)$ gates, and each gate on layer i only receives inputs from layer $i-1$. The *depth* of the circuit is $d(n)$ and the *width* of the circuit is $T(n)$. We will always assume $d(n) \leq T(n)$. A circuit family of dimension $d(n) \times T(n)$ is *logspace-uniform* if there is a Turing machine that on input 1^n, uses at most $O(\log T(n))$ space, and prints the description of C_n.

Recall that for $\varepsilon(n) > 0$, a Boolean function $f\colon \{0,1\}^M \to \{0,1\}$ is *ε-dense* if

$$\Pr_{x \sim \{0,1\}^M}[f(x) = 1] \geq \varepsilon(M).$$

We say f *ε-avoids* a hitting set $H \subseteq \{0,1\}^M$ if f is ε-dense and for every $y \in H$, $f(y) = 0$.

Now we are ready to state our main technical tool, the instance-wise hardness-randomness trade-off in [CT21a].

Theorem 8.4.1 ([CT21a] with the improved parameters from [CLO+23])**.** *There is an absolute constant $c \geq 1$ such that the following holds. Let $f\colon \{0,1\}^n \to \{0,1\}^{T(n)}$ be a multi-output function computable by a logspace-uniform circuit of dimension $d(n) \times T(n)$. Let $M(n)$ be a parameter such that $c \log T \leq M \leq T^{1/c}$. Then there are algorithms $\mathsf{CT21.HSG}_f$ and $\mathsf{CT21.Recon}_f$ depending on f, such that:*

- *The algorithm $\mathsf{CT21.HSG}_f(x)$ runs in deterministic T^c time and outputs a set of M-bit strings.*
- *Given $x \in \{0,1\}^n$ and $i \in [T]$ as inputs, and oracle access to a candidate distinguisher $D\colon \{0,1\}^M \to \{0,1\}$, $\mathsf{CT21.Recon}_f^D(x,i)$ runs in randomised $(dnM)^c$ time. If D $(1/M)$-avoids $\mathsf{CT21.HSG}_f(x)$, then with probability $\geq 1 - 2^{-M}$, $\mathsf{CT21.Recon}_f^D(x,i)$ outputs the i-th bit of $f(x)$.*

Moreover, there is a deterministic algorithm that, given the Turing machine M_f that prints the circuit for f in logspace, outputs the descriptions of $\mathsf{CT21.HSG}_f$ and $\mathsf{CT21.Recon}_f$ in time $\mathrm{poly}(|\langle M_f \rangle|)$.

Remark 8.4.2. The hardness-randomness trade-offs in [CT21a, CLO+23] were stated for hard functions of the form $f\colon \{0,1\}^n \to \{0,1\}^n$, where the reconstruction algorithm prints the entire string $f(x)$ in $\ll T$ time. Inspecting their proofs, it is easy to see that the same holds when we have a function $f\colon \{0,1\}^n \to \{0,1\}^T$ and the reconstruction algorithm prints the i-th bit of $f(x)$ in $\ll T$ time, given $i \in [T]$ as an input.

Theorem 8.4.3 (Non-black-box reduction under logspace-uniform sub-polynomial depth algorithms)**.** *Let $\delta(n) = o(1)$ be any function. Suppose there is a constant $\varepsilon > 0$ and an algorithm $\mathcal{A}(\langle C \rangle)$ that solves the* `Implicit-`δ`-Heavy-Avoid` *problem on instances $G\colon \{0,1\}^{N^\varepsilon} \to \{0,1\}^N$ that are implicitly computable by a circuit C of size N^ε, where a description of C is given as input. Moreover, assume that $\mathcal{A}$ can be implemented as a logspace-uniform circuit of size $\mathrm{poly}(N)$ and depth $N^{o(1)}$. Then $\mathsf{prBPP} = \mathsf{prP}$.*

Proof. Since $\mathsf{prBPP} \subseteq \mathsf{prRP}^{\mathsf{prRP}}$ [BF99], it suffices to prove that $\mathsf{prRP} = \mathsf{prP}$. That is, we want a deterministic algorithm that, given as input a size-$2M$ circuit $D\colon \{0,1\}^M \to \{0,1\}$, distinguishes between the case that D rejects every input and the case that D is $1/2$-dense.

Let c be the constant in Theorem 8.4.1, and set $N := M^{3c/\varepsilon}$. We recall our assumption: given an implicit map with parameters as above, for some $T \le \mathrm{poly}(N)$, $d \le N^{o(1)}$ and $\delta \le o(1)$, there is a logspace-uniform circuit $\mathcal{A}$ of dimension $d \times T$ that solves the corresponding `Implicit-`δ`-Heavy-Avoid` problem deterministically.

Consider the implicit map $G_D\colon \{0,1\}^{N^\varepsilon} \to \{0,1\}^N$, where the underlying circuit C_D for computing each output bit of G_D is as follows. Given $x \in \{0,1\}^{N^\varepsilon}$ and $i \in [N]$, we parse x as (M_f, r), where M_f is a program, and r consists of the remaining bits of x (treated as randomness). We can encode x in such a way that every program of length ℓ appears with probability mass $\Theta(2^{-\ell}/\ell^2)$, hence every program with constant description length appears with probability mass $\Omega(1) > \delta$. Suppose that M_f is a logspace-uniform Turing machine that defines a circuit of size $T \cdot \mathrm{poly}(M)$ and depth $d + \mathrm{polylog}(M)$ that computes a function $f\colon \{0,1\}^{\tilde{O}(M)} \to \{0,1\}^N$ (this can be syntactically ensured by imposing a space constraint on M_f). We let

$$C_D(x,i) := \mathsf{CT21.Recon}_f^D(\langle D \rangle, i; r). \tag{8.1}$$

Here, $\mathsf{CT21.Recon}(\langle D \rangle, i; r)$ denotes the output of $\mathsf{CT21.Recon}(\langle D \rangle, i)$ *on randomness* r. Note that when we compute $C_D(x,i)$, we treat D both as the distinguisher (for the reconstruction algorithm $\mathsf{CT21.Recon}$) and as the input of f. We then run $\mathsf{CT21.Recon}$ on randomness r and (attempt to) reconstruct the i-th bit of $f(\langle D \rangle)$. If the length of $\langle M_f \rangle$ is a constant, then we have $|r| \ge N^\varepsilon - O(1) \ge M^{2.9c} \ge (d^2 \cdot \tilde{O}(M) \cdot M)^c$, hence there is always enough randomness to feed into Recon. Also, by Theorem 8.4.1, the description $\langle C_D \rangle$ can be computed from the description $\langle D \rangle$ in $\mathrm{poly}(M)$ time and $\mathrm{polylog}(M)$ depth.[14] Note that G_D can be implicitly computed in time N^ε due to our choice of parameters.

Define a function $f'\colon \{0,1\}^{\tilde{O}(M)} \to \{0,1\}^N$ as follows: Given a size-$2M$ circuit $D\colon \{0,1\}^M \to \{0,1\}$ as input, f' computes the circuit C_D as in (8.1) and outputs $\mathcal{A}(\langle C_D \rangle)$. Recall that $\mathcal{A}$ is

[14] The depth upper bound is dominated by computing the description of $\mathsf{CT21.Recon}_f$ from M_f, which takes time $\mathrm{polylog}(M)$. Although $\mathsf{CT21.Recon}_f$ is an *adaptive* oracle algorithm, to compute the *code* of $\mathsf{CT21.Recon}_f^D$ from $\langle D \rangle$ we only need to concatenate the codes of $\mathsf{CT21.Recon}_f$ and D together, hence this step is depth-efficient.

computed by a logspace-uniform circuit of dimension $d \times T$, hence f' is computed by a logspace-uniform circuit of size $T \cdot \mathrm{poly}(M)$ and depth $d + \mathrm{polylog}(M)$.

Assuming D is 1/2-dense, we argue that $\mathsf{CT21.HSG}_{f'}(\langle D \rangle)$ hits D. Otherwise, D (1/2)-avoids $\mathsf{CT21.HSG}_{f'}(\langle D \rangle)$. By Theorem 8.4.1, for every $i \in [N]$, with probability at least $1 - 2^{-M}$ over the randomness r, $\mathsf{CT21.Recon}^{D}_{f'}(\langle D \rangle, i; r)$ is equal to the i-th output bit of $f'(\langle D \rangle)$. By a union bound, w.p. at least $1 - N \cdot 2^{-M}$ over the randomness r, we have that for every $i \in [N]$, $\mathsf{CT21.Recon}^{D}_{f'}(\langle D \rangle, i; r) = f'(\langle D \rangle)_i$. Since $f'(\langle D \rangle) = \mathcal{A}(\langle C_D \rangle)$ by definition, we have:

$$\Pr_x[G_D(x) = \mathcal{A}(\langle C_D \rangle) \mid \langle M_f \rangle = \langle M_{f'} \rangle \text{ where } (M_f, r) = x] \geq 1 - N \cdot 2^{-M} \geq 1/2.$$

Since the description length of $M_{f'}$ is a constant, it follows that $\langle M_f \rangle = \langle M_{f'} \rangle$ with constant probability. Hence,

$$\Pr_x[G_D(x) = \mathcal{A}(\langle C_D \rangle)] \geq \Omega(1),$$

contradicting our assumption that $\mathcal{A}$ solves the $\texttt{Implicit-}\delta\texttt{-Heavy-Avoid}$ problem.

We have shown that if D is 1/2-dense, then $\mathsf{CT21.HSG}_{f'}(\langle D \rangle)$ hits D. We can compute $\mathsf{CT21.HSG}_{f'}(\langle D \rangle)$ in $\mathrm{poly}(T, N) \leq \mathrm{poly}(M)$ time, hence we can solve the $\mathsf{Gap\text{-}UNSAT}$ problem in deterministic polynomial time. This implies that $\mathsf{prRP} = \mathsf{prP}$, as desired. □

Remark 8.4.4. The above reduction is non-black-box for two reasons. First, the statement $\mathsf{prBPP} \subseteq \mathsf{prRP}^{\mathsf{prRP}}$ can be seen as a non-black-box reduction from prBPP to prRP [BF99]. Second, and perhaps more interestingly, the reduction from prRP to $\texttt{Implicit-Heavy-Avoid}$ is also non-black-box, as it requires the algorithm $\mathcal{A}$ for $\texttt{Implicit-Heavy-Avoid}$ to be a *logspace-uniform circuit of low depth* and relies on an application of Theorem 8.4.1 over this low-depth circuit. Indeed, the proof of Theorem 8.4.1 performs arithmetisation on this low-depth circuit.

Remark 8.4.5. It is also interesting to compare Theorem 8.4.3 with [Kor22, Theorem 8]. The latter result is a *black-box* reduction from prBPP to a problem called R-Lossy Code. In contrast, our Theorem 8.4.3 needs additional constraints on the algorithm solving $\texttt{Implicit-Heavy-Avoid}$ and makes non-black-box use of that algorithm.

Our $\texttt{Implicit-Heavy-Avoid}$ is a special case of R-Lossy Code in the following sense. Recall that R-Lossy Code is the problem where, given circuits $\mathsf{Comp} : \{0,1\}^n \times \{0,1\}^m \to \{0,1\}^{n-1}$ and $\mathsf{Decomp} : \{0,1\}^{n-1} \to \{0,1\}^n$ and a parameter $\delta > 0$, one needs to find some $x \in \{0,1\}^n$ such that $\Pr_{r \leftarrow \{0,1\}^m}[\mathsf{Decomp}(\mathsf{Comp}(x, r)) = x] < \delta$. Given an implicit map $C : \{0,1\}^r \times [N] \to \{0,1\}$ as the input of $\texttt{Implicit-}\delta\texttt{-Heavy-Avoid}$ (recall that $r < N$ in the typical parameter setting), we can reduce it to the R-Lossy Code instance $(\mathsf{Comp}, \mathsf{Decomp})$ where $\mathsf{Comp}(x, r)$ simply outputs its randomness r, and $\mathsf{Decomp}(r) = C(r, 1)C(r, 2) \ldots C(r, N)$. In this sense, $\texttt{Implicit-Heavy-Avoid}$ is no more than a special case of R-Lossy Code where the compressor circuit is *trivial*.

8.4.2 Getting Rid of the Depth Assumption

In this section, we show how to get rid of the low-depth assumption in Theorem 8.4.3 in a weaker but non-trivial setting. An ideal statement would be: a deterministic polynomial-time algorithm for $\texttt{Implicit-Heavy-Avoid}$ implies a deterministic polynomial-time algorithm for $\mathsf{Gap\text{-}SAT}$ or CAPP. Compared to the ideal statement, the actual result that we prove only holds for subexponential-time infinitely-often algorithms, as we shall explain later.

We note that it should not be too surprising that a subexponential-time infinitely-often

algorithm for `Implicit-Heavy-Avoid` implies a subexponential-time infinitely-often algorithm for Gap-SAT. In fact, combining Theorem 8.3.9 and [IW01], it is easy to show that this holds for *heuristic* algorithms.[15]

Theorem 8.4.6. *The following items are equivalent.*

- (*Infinitely-often subexponential-time heuristic derandomisation.*)

  For every language $L \in \mathsf{BPP}$, every ensemble of polynomial-time samplable distributions $\mathcal{D} = \{D_n\} \in \mathsf{PSAMP}$, every polynomial $p(\cdot)$, and every constant $\delta > 0$, there exists a deterministic Turing machine M running in 2^{n^δ} time, such that for infinitely many input lengths n,

  $$\Pr_{x \sim D_n}[L(x) \neq M(x)] \leq 1/p(n).$$

- (*Infinitely-often polynomial-time algorithms for uniform* `Implicit-Heavy-Avoid`.)

  For every $\delta(N) = 1/\mathrm{polylog}(N)$, and for every sequence $\{G_N\}$ of maps $G_N \colon \{0,1\}^{\mathrm{polylog}(N)} \to \{0,1\}^N$ where each bit of G_N is implicitly computed in $\mathrm{polylog}(N)$ time, there is a deterministic $\mathrm{poly}(N)$-time algorithm that solves the `Implicit-`δ`-Heavy-Avoid` *problem for $\{G_N\}$ on infinitely many input lengths N.*

Proof Sketch. In fact, both items are equivalent to $\mathsf{EXP} \neq \mathsf{BPP}$. The equivalence between the first item and $\mathsf{EXP} \neq \mathsf{BPP}$ is shown in [IW01], and the equivalence between the second item and $\mathsf{EXP} \neq \mathsf{BPP}$ follows from Theorem 8.3.9. □

We now attempt to prove a version of Theorem 8.4.6 with respect to *worst-case* algorithms, instead of *heuristics*. Our worst-case version of Theorem 8.4.6 also has many caveats such as being infinitely-often and requiring subexponential time, but the biggest caveat might be that we could only obtain infinitely-often algorithms in the following, somewhat artificial, setting: For a sequence of inputs $\{x_n\}_{n \in \mathbb{N}}$, the algorithms read many inputs $x_1, x_2, \ldots, x_{\mathrm{poly}(n)}$ but are only required to solve x_n. We call this "infinitely-often*" algorithms. Formally, we have:

Definition 8.4.7. Let $\mathcal{P}$ be a computational problem and $\{x_n\}_{n \in \mathbb{N}}$ be a sequence of inputs. We say an algorithm $\mathcal{A}$ *infinitely-often** *solves* $\mathcal{P}$ *on* $\{x_n\}$ if there is a polynomial $p(\cdot)$ such that for infinitely many integers n, $\mathcal{A}(1^n, x_1, x_2, \ldots, x_{p(n)})$ outputs a valid $\mathcal{P}$-solution for x_n.

In other words, an infinitely-often* algorithm has access to the (non-uniform) sequence of inputs around x_n, as opposed to the usual setting where the algorithm only has access to the given input string. We remark that our algorithm in Theorem 8.4.8 works in a weaker model where the machine only reads x_n and $x_{p(n)}$ and outputs an answer for x_n. However, we believe that Definition 8.4.7 is a more accurate model to capture win-win analyses in complexity theory, hence choose to define it in this way.

For ease of notation, in what follows, we will denote the sequence $x_1, x_2, \ldots, x_\ell$ simply by $x_{1 \sim \ell}$. We are now able to state our main result in this section.

[15] Note that Theorem 8.4.6 refers to `Implicit-Heavy-Avoid` for *uniformly samplable* distributions, which is different from most results in this section. We believe that more connections between `Implicit-Heavy-Avoid` for *uniformly samplable* distributions and *average-case* derandomisation can be obtained (e.g., using the recent "unstructured hardness to average-case randomness" [CRT22]), but we do not pursue this direction here.

Theorem 8.4.8 (A non-black-box reduction for `Implicit-Heavy-Avoid`)**.**

Assume there is an infinitely-often polynomial-time algorithm for `Implicit-Heavy-Avoid` *with subexponential stretch. That is, for every constants $a \geq 1$, $\varepsilon > 0$, and function $\delta \leq o(1)$, there exists a deterministic algorithm* Avoid *running in* $\mathrm{poly}(N)$ *time such that for infinitely many $n \in \mathbb{N}$ and $N := 2^{n^\varepsilon}$, and for every generator $G\colon \{0,1\}^{n^a} \to \{0,1\}^N$ implicitly described by a size-n^a circuit $C\colon \{0,1\}^{n^a} \times [N] \to \{0,1\}$,* $\mathsf{Avoid}(\langle C \rangle)$ *solves* `Implicit-`δ`-Heavy-Avoid` *on G.*

Then there is an infinitely-often subexponential-time algorithm for* Gap-SAT*. That is, for every $\varepsilon > 0$ and $c \geq 1$, there exists a deterministic algorithm* Derand *running in 2^{n^ε} time such that the following holds: For every sequence of circuits $\{D_n\}_{n\in\mathbb{N}}$, where each $D_n : \{0,1\}^{n^c} \to \{0,1\}$ is a circuit of size $2n^c$,* $\mathsf{Derand}(1^n, \langle D_{1\sim\mathrm{poly}(n)} \rangle)$ *infinitely-often* solves* Gap-SAT *on $\{D_n\}$.*

Our proof combines the two instance-wise hardness-randomness trade-offs introduced by Chen and Tell [CT21a] and Liu and Pass [LP23] recently. Since the hardness-randomness trade-off in [CT21a] is already summarised in Theorem 8.4.1, in what follows, we summarise the hardness-randomness trade-off in [LP23].

Definition 8.4.9. Let $f\colon \{0,1\}^n \to \{0,1\}^{T(n)}$ be a function, A be a randomised algorithm, and $x \in \{0,1\}^n$ be an input of f. We say that $f(x)$ is *ℓ-leakage resilient hard* against A if for every "leakage string" $\mathsf{leak} \in \{0,1\}^\ell$, there is some $i \in [T]$ such that $\Pr[A(x, \mathsf{leak}, i) = f(x)_i] \leq 2/3$, where the probability is over the internal randomness of A.

We need the following result by Liu and Pass [LP23] showing that leakage resilient hardness can be used for derandomisation.

Theorem 8.4.10 ([LP23])**.** *There are algorithms* LP23.PRG *and* LP23.Recon *and an absolute constant $c \geq 1$ such that the following holds. Let $f\colon \{0,1\}^n \to \{0,1\}^{T(n)}$ be a function, $D\colon \{0,1\}^M \to \{0,1\}$ be a distinguisher, and $x \in \{0,1\}^n$ be an input of f. Let $\ell := (M \log T)^c$ and $r := O(\log^2 T / \log M)$. Then:*

- $\mathsf{LP23.PRG}\colon \{0,1\}^T \times \{0,1\}^r \to \{0,1\}^M$ *runs in deterministic* $\mathrm{poly}(T, M)$ *time.*
- $\mathsf{LP23.Recon}^{(-)}\colon \{0,1\}^\ell \times [T] \to \{0,1\}$ *runs in randomised* $\mathrm{poly}(\ell, \log T)$.
- *If $f(x)$ is ℓ-leakage resilient hard against* $\mathsf{LP23.Recon}^D$*, then* $\mathsf{LP23.PRG}(f(x), -)$ *is a (targeted) PRG that $(1/10)$-fools D.*

Proof Sketch. This follows by observing that the leakage resilient hardness-randomness trade-off in [LP23] holds *instance-wise*. In particular, if we let g be the "k-reconstructive PRG" described in [LP23, Theorem 3.11] (which follows from [STV01]), then $\mathsf{LP23.PRG}(f(x), z) = g^{f(x)}(1^{m(n)}, 1^{M(n)}, z)$; the algorithm LP23.Recon is simply the corresponding "reconstruction algorithm" R as defined in [LP23, Definition 3.10]. □

Suppose that $f\colon \{0,1\}^n \to \{0,1\}^{m(n)}$ is computable by a deterministic Turing machine M running in time $T(n)$. We can define $f^{\mathsf{hist}}\colon \{0,1\}^n \to \{0,1\}^{T'(n)}$ for some function $T'(n) \leq \mathrm{poly}(T(n), m(n))$ such that $f^{\mathsf{hist}}(x)$ outputs the *computational history* of $f(x)$, i.e., the sequence of configurations of M when computing over the input string x. It will be useful to consider the leakage-resilience hardness of f^{hist}, since if f^{hist} is not leakage resilient hard, then f can be computed by a *low-depth* circuit:

Claim 8.4.11. *Let $f\colon \{0,1\}^n \to \{0,1\}^{T(n)}$ be a function computable in time $T(n)$, $\ell = \ell(n)$ be a parameter, and A be a randomised algorithm running in time $T_A(n)$. Then, for $d(n) := O(T_A(n) + \ell(n) + \log T(n))$, there is a logspace-uniform circuit $C\colon \{0,1\}^n \to \{0,1\}^{T(n)}$ of dimension $d(n) \times 2^{d(n)}$ such that the following holds. For every input $x \in \{0,1\}^n$, if $f^{\mathsf{hist}}(x)$ is not $\ell(n)$-leakage resilient hard against A, then $f(x) = C(x)$.*

Proof. The circuit $C(x)$ enumerates all strings $\mathsf{leak} \in \{0,1\}^\ell$ and computes $A(x, \mathsf{leak}, i)$ for each i. Although A is a randomised algorithm, we can compute $A(x, \mathsf{leak}, i)$ by brute force using a logspace-uniform circuit of width $2^{O(T_A(n))}$ and depth $O(T_A(n))$. Then, (for each leak) C verifies whether the computational history $H(x, \mathsf{leak})$ defined by $H(x, \mathsf{leak})_i = A(x, \mathsf{leak}, i)$ is indeed the correct history for $f(x)$; this can be done by a logspace-uniform circuit of width $\mathrm{poly}(T(n))$ and depth $O(\log T(n))$, by checking that every local step in $H(x, \mathsf{leak})$ is correct. Whenever there is some leak that corresponds to the correct history for $f(x)$, the circuit C outputs the value of $f(x)$ according to this history. The depth of C is $d(n) \le O(T_A(n) + \ell(n) + \log T(n))$ and the size of C is exponential in $d(n)$. □

Now we are ready to prove Theorem 8.4.8.

Proof of Theorem 8.4.8. Let $c \ge 1$ and $\varepsilon > 0$ be constants. Let $\{D_n\}_{n\in\mathbb{N}}$ be a sequence of circuits where each $D_n\colon \{0,1\}^{n^c} \to \{0,1\}$ is of size $2n^c$. Recall that we want a deterministic 2^{n^ε}-time algorithm that infinitely-often* solves Gap-SAT on $\{D_n\}$. Let c_1 be the constant in Theorem 8.4.1 (the trade-off in [CT21a]) or Theorem 8.4.10 (the trade-off in [LP23]), whichever is larger. Let $\kappa := \lceil \max\{4c/\varepsilon, 8c_1/\varepsilon\} \rceil$; the meaning of this constant is that we will perform a win-win analysis over input lengths m and m^κ. Finally, let $a := 4\kappa c c_1$ and $L(t) := 2^{t^{\varepsilon/4}}$. The input instances of our Heavy Avoid algorithm will be generators $G_t\colon \{0,1\}^{t^a} \to \{0,1\}^{L(t)}$ whose output bits are implicitly computable in t^a size.

c, ε	We want a Gap-SAT algorithm on n^c-size circuits in 2^{n^ε} time
c_1	Overheads of the trade-offs in Theorem 8.4.1 and Theorem 8.4.10
$\kappa = O(c/\varepsilon)$	Our win-win analysis is over input lengths m and m^κ
$a = O(c^2/\varepsilon)$	We need a Heavy Avoid algorithm for generators $G\colon \{0,1\}^{t^a} \to \{0,1\}^{L(t)}$, implicitly computable in t^a size
$L(t) = 2^{t^{\varepsilon/4}}$	

Table 8.1: Constants used in this proof.

For any integer M, its *index* $idx(M)$ is defined as the largest integer such that $M = m^{\kappa^{idx(M)}}$ for some integer m. (Note that most integers have index 0.) We say an input length M is *big* if $idx(M)$ is odd, and is *small* if $idx(M)$ is even. For convenience, we will always use upper-case M to denote big input lengths and lower-case m to denote small input lengths. Looking ahead, each small input length m will be paired with a big input length m^κ and vice versa; if our Heavy Avoid algorithm succeeds on input length m, then our Gap-SAT algorithm succeeds on either length m or length m^κ.

We now define the sequence of implicit descriptions $C_t\colon \{0,1\}^{t^a} \times [L(t)] \to \{0,1\}$ for each $t \in \mathbb{N}$, which we feed into our Heavy Avoid algorithm. Each C_t also defines the generator $G_t : \{0,1\}^{t^a} \to \{0,1\}^{L(t)}$. Let M be the t-th smallest big input length. Note that $M \le O(t^\kappa)$. Let x denote the input of G_t. We parse x into $(\langle M_f \rangle, r)$, where $\langle M_f \rangle$ is the description of a

Turing machine M_f and the remaining $M^{(2c+1)c_1}$ bits r are treated as randomness. As in the proof of Theorem 8.4.3, we encode x in such a way that every constant-length program M_f occurs with constant probability. Now, let $d := M^{2\varepsilon/3}$ and f be the $d \times 2^d$ circuit outputted by M_f within space constraint d. Then

$$C_t(x, i) := \mathsf{CT21.Recon}_f^{D_M}(\langle D_M \rangle, i; r).$$

Recall that $\mathsf{CT21.Recon}$ uses at most $(d \cdot |\langle D_M \rangle|^2)^{c_1} \leq (dM^{2c})^{c_1}$ random bits, hence we have enough random bits to feed into $\mathsf{CT21.Recon}$. We can see that each bit of G_t can be computed in $M^{3cc_1} \leq t^{4\kappa cc_1} \leq t^a$ size.

Let Avoid be our algorithm for $\texttt{Implicit-}\delta\texttt{-Heavy-Avoid}$ that runs in deterministic $\mathrm{poly}(L(t))$ time. Let I be the set of input lengths t for which $\mathsf{Avoid}(\langle C_t \rangle)$ correctly outputs a δ-light element of G_t. Using the same reasoning as Theorem 8.4.3, one can see that:

Claim 8.4.12. *Let $t \in I$, M be the t-th big input length, and let $d := M^{\varepsilon/2}$. Suppose there is a constant-length Turing machine that outputs a circuit f' of dimension $d \times 2^d$ in $O(d)$ space such that $f'(\langle C_t \rangle) = \mathsf{Avoid}(\langle C_t \rangle)$. Then $\mathsf{CT21.HSG}_{f'}(\langle C_t \rangle)$ hits D_M.*

Let $\mathsf{Avoid}^{\mathsf{hist}}$ denote the algorithm that outputs the computational history of Avoid. Note that $\mathsf{Avoid}^{\mathsf{hist}}(\langle C_t \rangle)$ runs in $T_{\mathsf{hist}}(t) \leq \mathrm{poly}(L(t))$ time. Now let $m \in \mathbb{N}$ be the t-th small input length, $M := m^\kappa$ be the t-th big input length, and $\ell(t) := (m \cdot \log T_{\mathsf{hist}}(t))^{c_1}$. Consider the following criterion for our win-win analysis:

- ▶ $\mathsf{Crit}(t)$: $\mathsf{Avoid}^{\mathsf{hist}}(\langle C_t \rangle)$ is $\ell(t)$-leakage resilient hard against $\mathsf{LP23.Recon}^{D_m}$.

Our algorithm Derand works as follows. On input $(1^n, \langle D_{1 \sim n^\kappa} \rangle)$:

- Suppose n is small, $m := n$, and $M := n^\kappa$. Assume that m is the t-th small input length and assume that $\mathsf{Crit}(t)$ holds. Then by Theorem 8.4.10,

  $$\mathsf{LP23.PRG}(\mathsf{Avoid}^{\mathsf{hist}}(\langle C_t \rangle), -) \colon \{0,1\}^r \to \{0,1\}^m$$

  is a PRG that $(1/10)$-fools D_m, where $r := O(\log^2 T_{\mathsf{hist}}(t)/\log m) \leq O(t^{\varepsilon/2}/\log m) < m^{\varepsilon/2}$. By enumerating this PRG, we can solve the $\mathsf{Gap\text{-}SAT}$ (in fact, CAPP) problem on input D_m in deterministic 2^{m^ε} time.

  Note: The definition of C_t involves $D_M = D_{n^\kappa}$; this is why we need our infinitely-often* algorithm to have access to inputs on a larger length.

- Suppose n is big, $M := n$, and $m := n^{1/\kappa}$. Suppose that M is the t-th big input length and assume that $\mathsf{Crit}(t)$ does not hold. Then it follows from Claim 8.4.11 that there is a logspace-uniform circuit $\widetilde{\mathsf{Avoid}}$ of dimension $d' \times T'$ such that $\widetilde{\mathsf{Avoid}}(\langle C_t \rangle) = \mathsf{Avoid}(\langle C_t \rangle)$, where $d' = O(m^c + \ell(m) + \log T_{\mathsf{hist}}(t)) \leq O(m^c + (mt^{\varepsilon/4})^{c_1}) < M^{\varepsilon/2}$ and $T' = 2^{d'}$. By Claim 8.4.12, $\mathsf{CT21.HSG}_{\widetilde{\mathsf{Avoid}}}(\langle C_t \rangle)$ hits D_M. Since the size of this HSG is $2^{O(d')} < 2^{M^{0.9\varepsilon}}$, we can solve the $\mathsf{Gap\text{-}SAT}$ problem on D_M in 2^{M^ε} time.

For every $t \in I$, let m be the t-th small input length and $M = m^\kappa$. If $\mathsf{Crit}(t)$ holds, then our algorithm solves $\mathsf{Gap\text{-}SAT}$ on input D_m; otherwise our algorithm solves $\mathsf{Gap\text{-}SAT}$ on input D_M. Since $|I|$ is infinite, it follows that our algorithm infinitely-often* solves $\mathsf{Gap\text{-}SAT}$ on $\{D_n\}$. □

8.4.3 On Black-Box Reductions to Heavy-Avoid

We complement the previous non-black-box reductions by showing that if there is a black-box reduction from Gap-SAT to `Heavy-Avoid` of a certain type, then we would have prBPP = prP *unconditionally*. In more detail, we consider the natural notion of *Levin reductions* [Lev73], i.e., "witness-mapping" reductions between search problems, and show that a Levin reduction from search-Gap-SAT to `Heavy-Avoid` implies prBPP = prP.

Intriguingly, these results together *separate* the notion of (weak) non-black-box reductions and black-box (i.e., Levin) reductions between two natural problems w.r.t. current techniques! That is, improving the weak non-black-box reductions in Theorem 8.4.3 and Theorem 8.4.8 to Levin reductions (which is a stronger notion of black-box reduction) would imply breakthroughs in complexity theory.

As the notion of Levin reductions is standard in complexity theory (for a recent example, see [MP24]), we only recall its definition in the special case of reducing search-Gap-SAT to `Heavy-Avoid`:

Definition 8.4.13. We say there is a *Levin reduction* from search-Gap-SAT to `Heavy-Avoid` if there are functions f, g computable in deterministic polynomial time such that the following holds:

- For every circuit $C : \{0,1\}^n \to \{0,1\}$ that is 1/2-dense (i.e., that is a valid search-Gap-SAT instance), $f(C) = (D, 1^t)$ is a `Heavy-Avoid` instance where we want to find a $(1/t)$-light string in D.
- For every string y that is $(1/t)$-light for D, $g(C, y)$ is a valid solution for search-Gap-SAT for C. That is, if C is 1/2-dense, then $C(g(C, y)) = 1$.

Theorem 8.4.14. *If there is a polynomial-time Levin reduction (f, g) from* search-Gap-SAT *to* `Heavy-Avoid`*, then* prP = prBPP.

Proof. First, we describe the main idea. The crucial observation is that there is a trivial algorithm that "*list-solves*" any `Heavy-Avoid` instance, that is, outputs a list of solutions such that some element in the list is not heavy. In particular, let $(D, 1^t)$ be an input instance of `Heavy-Avoid`, and consider the trivial algorithm that outputs an arbitrary list of $t + 1$ distinct strings. By an averaging argument, at least one string in this list will be a $(1/t)$-light element of D. This means that using the list and a witness-mapping reduction, we can produce at least one satisfying assignment if the input circuit is dense.

We proceed to the formal proof. Again, by [BF99], it suffices to prove that prP = prRP. Let $C : \{0,1\}^n \to \{0,1\}$ be an input to Gap-SAT. We first reduce C to a `Heavy-Avoid` instance $(D, 1^t) := f(C)$. Then we compute an arbitrary list of $t+1$ distinct strings $x_1, x_2, \ldots, x_{t+1}$. We are guaranteed that some x_i is a δ-light element of D, hence if C is 1/2-dense, then $g(C, x_i)$ would be a satisfying assignment of C. Consequently, if there is an index $i \in [\ell]$ such that $C(g(C, x_i)) = 1$, then we output 1, otherwise we output 0. It is easy to see that if C is unsatisfiable then we always output 0, while if C is 1/2-dense then we always output 1. Hence, we have prP = prRP, and this concludes the proof. □

8.4.4 Heavy Avoid versus Almost-All-Inputs Hardness

Finally, we show connections between `Implicit-Heavy-Avoid` and the *almost-all-inputs* hardness assumptions, introduced recently in [CT21a].

The results in this section are motivated by two conjectures. Given the non-black-box reductions presented in Theorem 8.4.3 and Theorem 8.4.8, it seems natural to conjecture that `Implicit-Heavy-Avoid` is complete for prBPP under "the most natural notion of non-black-box reductions":

Conjecture 8.4.15 (Informal)**.** *If* `Implicit-Heavy-Avoid` *for non-uniform samplers admits a deterministic polynomial-time algorithm, then* prBPP = prP*.*

On the other hand, there is another intriguing conjecture implicit in the work of Chen–Tell [CT21a]. Recall that Chen and Tell [CT21a] showed how to derive prBPP = prP given a multi-output function $f\colon \{0,1\}^n \to \{0,1\}^n$ that is *almost-all-inputs* hard against randomised algorithms, *provided that* f *can be computed in low depth.* They also showed that prBPP = prP *necessitates* the existence of multi-output functions with almost-all-inputs hardness, but the hard function they construct might require high depth. Still, this demonstrates that almost-all-inputs hardness might be *the right hardness assumption* for derandomisation. It is tempting to conjecture that low-depth constraints are, in fact, not necessary:

Conjecture 8.4.16 (Informal)**.** *If there is a multi-output function* $f\colon \{0,1\}^n \to \{0,1\}^n$ *computable in deterministic polynomial time that is almost-all-inputs hard against fixed-polynomial time randomised algorithms, then* prBPP = prP*.*

In this section, we show that in an "implicit" setting, Conjecture 8.4.15 and Conjecture 8.4.16 are equivalent! In fact, we show that `Implicit-Heavy-Avoid` is the computational problem characterising the task of "creating" almost-all-inputs hardness (against randomised algorithms). We also show that creating such hardness is equivalent to generating strings with high conditional sublinear-time probabilistic Kolmogorov complexity.

Our "implicit" setting. We consider functions with a possibly long output, i.e., $f\colon \{0,1\}^n \to \{0,1\}^{\ell(n)}$ where $\ell(n)$ is larger than the running time of our adversaries. Consequently, it makes sense to only require our adversaries to output each bit of $f(x)$ given x and the index of that bit. It is worth noting that the hardness-randomness trade-offs in both [CT21a] and [LP23] hold for such implicit adversaries.

Formally, let $\mathcal{A}$ be a randomised algorithm, we say that $\mathcal{A}$ *locally* computes $f(x)$ if for every $i \in [\ell(n)]$, it holds that $\Pr[\mathcal{A}(x,i) = f(x)_i] \geq 2/3$, where the probability is over the internal randomness of $\mathcal{A}$. An equivalent way of saying this is that $f(x)$ is not 0-leakage resilient hard against $\mathcal{A}$ in the sense of Definition 8.4.9.

We also consider the task of generating strings with high (conditional) *sublinear-time* probabilistic Kolmogorov complexity. (Our equivalence results hold for both $\mathrm{pK}^{\mathrm{poly}}$ and $\mathrm{rK}^{\mathrm{poly}}$.) Roughly speaking, fix a universal Turing machine U, a time bound t, and strings x, y, where $|x| \ll t \ll |y|$. The conditional complexity of y given x is the length of the shortest program p such that $U(p, w, x, i)$ outputs the i-th bit of y in t steps, with w being the randomness. However, there is a technical detail that is worth stressing: In our definition, we require the resources

(x and r) to have length at most t, hence the universal Turing machine U has time to read them in their entirety. A possible alternative definition would be that U has *oracle access* to strings x and r (whose lengths might be $\gg t$), but it is unclear if we can extend our equivalence result (Theorem 8.4.18) to these alternative definitions.

Definition 8.4.17 (Sublinear-time probabilistic Kolmogorov complexity)**.** Let U be a universal Turing machine, $x, y \in \{0,1\}^*$, and $t \in \mathbb{N}$. (Think of $|x| \ll t$ and $|y| \gg t$.) We artificially define $y_{|y|+1} = \star$.

- **Sublinear-time pK^t complexity:**

$$\mathrm{pK}^t_U(y \mid x) := \min\left\{ k \in \mathbb{N} \;\middle|\; \Pr_{w \sim \{0,1\}^t}\left[\exists p \in \{0,1\}^k, \forall i \in [|y|+1], U(1^t, p, w, x, i) = y_i\right] \geq \frac{2}{3} \right\}.$$

  In other words, if $k = \mathrm{pK}^t_U(y \mid x)$, then with probability at least $2/3$ over the choice of the *length-t* random string w, given w and x, the string y admits a t-time-bounded *local* encoding of length k.

- **Sublinear-time rK^t complexity:**

$$\mathrm{rK}^t_U(y \mid x) = \min_{p \in \{0,1\}^*}\left\{ |p| \;\middle|\; \forall i \in [|y|+1], \Pr_{w \sim \{0,1\}^t}[U(1^t, p, w, x, i) = y_i] \geq \frac{2}{3} \right\}.$$

Now we are ready to present our equivalence result.

Theorem 8.4.18. *The following are equivalent.*

(1) **(Almost-all-inputs hardness.)** *For every polynomial $p(\cdot)$, there exists a polynomial $q(\cdot)$ and a function $f\colon \{0,1\}^n \to \{0,1\}^{q(n)}$ computable by a deterministic polynomial-time algorithm, such that every algorithm running in randomised $p(n)$ time only locally computes $f(x)$ on finitely many inputs $x \in \{0,1\}^*$.*

(2) **(Deterministic algorithms for `Heavy-Avoid`.)** *For every polynomial $p(\cdot)$, there exists a polynomial $q(\cdot)$ such that the following holds. Let $\mathcal{C}$ denote the class of circuits with n input bits and $q(n)$ output bits where each output bit is implicitly computed by a size-$p(n)$ circuit, then $\mathcal{C}$-`Implicit`-$(1/p(n))$-`Heavy-Avoid` can be solved in deterministic polynomial time.*

(3) **(Finding strings with large conditional $\mathrm{pK}^{\mathrm{poly}}$-complexity.)** *For every polynomial $p(\cdot)$, there exists a polynomial $q(\cdot)$ and a deterministic polynomial-time algorithm that given an input $x \in \{0,1\}^n$, finds a string $y \in \{0,1\}^{q(n)}$ such that $\mathrm{pK}^{p(n)}(y \mid x) \geq \log p(n)$.*

(4) **(Finding strings with large conditional $\mathrm{rK}^{\mathrm{poly}}$-complexity.)** *For every polynomial $p(\cdot)$, there exists a polynomial $q(\cdot)$ and a deterministic polynomial-time algorithm that given an input $x \in \{0,1\}^n$, finds a string $y \in \{0,1\}^{q(n)}$ such that $\mathrm{rK}^{p(n)}(y \mid x) \geq \log p(n)$.*

Arguably, the most interesting implication above is (1) $\Rightarrow$ (2), which can be interpreted as *instance-wise hardness vs. randomness* for solving `Implicit-Heavy-Avoid` without depth constraints.

Lemma 8.4.19. *In Theorem 8.4.18, (1) ⇒ (2) holds. That is, almost-all-inputs hardness (of any kind, without depth restrictions) can be used to solve* `Implicit-Heavy-Avoid`.

Proof. Let $T(n)$ and $q'(n)$ be two polynomials such that there is a multi-output function $f\colon \{0,1\}^n \to \{0,1\}^{q'(n)}$ computable in deterministic $T(n)$ time that is almost-all-inputs hard against randomised algorithms running in time $p(n)^6$. Let $q(n) := \mathrm{poly}(T(n), q'(n))$, and $f_{\mathsf{PCP}}\colon \{0,1\}^n \to \{0,1\}^{q(n)}$ denote the PCP of f: On input $x \in \{0,1\}^n$, $f_{\mathsf{PCP}}(x)$ outputs the concatenation of strings z, pcp_1, $\mathsf{pcp}_2, \ldots, \mathsf{pcp}_{q'(n)}$, where $z = f(x)$ and each pcp_i is a length-poly$(T(n))$ PCP proof for the assertion that the i-th output bit of $f(x)$ is equal to z_i. (Any efficiently-computable PCP with polynomial length and constant query complexity works here, e.g., [ALM+98, BGH+06, Din07].) We claim that f_{PCP} is an algorithm that solves $\mathcal{C}$-`Implicit`-$(1/p(n))$-`Heavy-Avoid`.

Suppose, towards a contradiction, that f_{PCP} does not solve $\mathcal{C}$-`Implicit`-$(1/p(n))$-`Heavy-Avoid` on an input $\langle C\rangle \in \{0,1\}^\ell$. The input $\langle C\rangle$ encodes a circuit C of size $p(n)$ (hence $\ell = \tilde{O}(p(n))$) that implicitly represents a generator $G\colon \{0,1\}^n \to \{0,1\}^{q(n)}$. Given $\langle C\rangle$, $r \in \{0,1\}^n$, and $i \in [q(n)]$, the i-th output bit of $G(r)$ can be computed in $\tilde{O}(p(n))$ time. Since f_{PCP} fails on $\langle C\rangle$, we have

$$\Pr_{r\sim\{0,1\}^n}[f_{\mathsf{PCP}}(\langle C\rangle) = G(r)] \geq 1/p(n). \tag{8.2}$$

Now we present a randomised algorithm $\mathcal{A}$ running in $p(n)^5$ time that locally computes f on input $\langle C\rangle$. Given an integer i, we want to compute the i-th bit of $f(\langle C\rangle)$. We repeat the following $O(p(n)^2)$ times:

1. Sample a random string $r \sim \{0,1\}^n$.
2. Parse $G(r)$ as the concatenation of $z, \mathsf{pcp}_1, \mathsf{pcp}_2, \ldots, \mathsf{pcp}_{q'(n)}$.
3. Invoke the PCP verifier $O(p(n))$ times to verify that pcp_i is indeed a correct PCP proof that $f(\langle C\rangle)_i = z_i$.
4. If all invocations of the PCP verifier are successful, then we output z_i and halt.

If we have not outputted anything after these $O(p(n)^2)$ iterations, then we output a random bit.

Let $\mathcal{A}$ denote the above randomised algorithm. We now analyse $\mathcal{A}$.

- **(Running time.)** Since each bit of pcp_i can be retrieved in $\tilde{O}(p(n))$ time, Step 3 above takes $\tilde{O}(p(n)^2)$ time, hence the whole algorithm runs in at most $\tilde{O}(p(n)^4) \leq p(n)^5$ time.
- **("Soundness.")** At each iteration where we parse $G(r)$ as $z, \mathsf{pcp}_1, \ldots, \mathsf{pcp}_{q(n)}$, if $f(\langle C\rangle)_i \neq z_i$, then no matter what pcp_i is, the PCP verifier will catch an error with probability at least $1 - \exp(-p(n))$. Hence, the probability that $\mathcal{A}(\langle C\rangle, i)$ halts in Step 4 above and outputs $1 - f(\langle C\rangle)_i$ is at most $\exp(-p(n))$.
- **("Completeness.")** On the other hand, by Eq. (8.2), with probability at least $1 - (1 - 1/p(n))^{p(n)^2} \geq 1 - \exp(-p(n))$, there is some iteration of the above algorithm in which $G(r) = f_{\mathsf{PCP}}(\langle C\rangle)$. During this iteration, it will be the case that $z_i = f(\langle C\rangle)_i$ and pcp_i is a valid PCP proof for this, therefore we will output $f(\langle C\rangle)_i$ and halt. It follows that the probability that none of the $p(n)^2$ iterations succeed and we output a random bit at the end is also upper bounded by $\exp(-p(n))$.

The "Soundness" and "Completeness" above imply that $\mathcal{A}$ locally computes $f(\langle C\rangle)$. To conclude, if f is indeed almost-all-inputs hard against randomised algorithms running in time $p(n)^6$, then f_{PCP} solves the $\mathcal{C}$-`Implicit`-$(1/p(n))$-`Heavy-Avoid` problem on all but finitely many inputs. □

Now we present the complete proof for Theorem 8.4.18.

Proof of Theorem 8.4.18. We consider each implication below.

(1) ⇒ (2): This follows from Lemma 8.4.19.

(2) ⇒ (3): Let $p(n)$ be a polynomial and $p'(n) := p(n)^2$. By (2), for some polynomial $q(n)$, there is a polynomial-time algorithm Avoid solving the $\mathcal{C}$-`Implicit`-$(1/p'(n))$-`Heavy-Avoid` problem, where the generators have output length $q(n)$ and each bit can be computed in size $p'(n)$. Let U be a universal Turing machine and suppose that given input $x \in \{0,1\}^n$, we want to find a string $y \in \{0,1\}^{q(n)}$ such that $\mathrm{pK}_U^{p(n)}(y \mid x) \geq \log p(n)$.

Consider the generator $G_x : \{0,1\}^{p(n)+\log p(n)} \to \{0,1\}^{q(n)}$ that is implicitly computed by the following circuit C_x: given (z,i) as input, where $z \in \{0,1\}^{p(n)+\log p(n)}$ and $i \in [q(n)]$, $C_x(z,i)$ outputs the i-th bit of $G(z)$. We parse z into $(\langle M\rangle, r)$, where M is a Turing machine of description length $\log p(n)$, and $r \in \{0,1\}^{p(n)}$ is treated as randomness. Then, $C_x(z,i)$ outputs $U(1^{p(n)}, \langle M\rangle, r, x, i)$. Clearly, C_x can be implemented by a circuit of size $\tilde{O}(p(n)) < p'(n)$.

Let $y \in \{0,1\}^{q(n)}$ be any string that is $\frac{1}{p'(n)}$-light for G_x, we claim that $\mathrm{pK}_U^{p(n)}(y \mid x) \geq \log p(n)$. This is easily shown by contradiction. Suppose that $\mathrm{pK}_U^{p(n)}(y \mid x) < \log p(n)$, then w.p. at least $2/3$ over $r \sim \{0,1\}^{p(n)}$, there is a program M of description length $\log p(n)$ such that for every $i \in [n+1]$, $U(1^{p(n)}, \langle M\rangle, r, x, i) = y_i$; in other words, $G_x(\langle M\rangle, r) = y$. This contradicts our assumption that y is $\frac{1}{p'(n)}$-light for G_x.

Hence, solving the $\mathcal{C}$-`Implicit`-$(1/p'(n))$-`Heavy-Avoid` problem on G_x will give us a string $y \in \{0,1\}^{q(n)}$ such that $\mathrm{pK}_U^{p(n)}(y \mid x) \geq \log p(n)$.

(3) ⇒ (4): This follows from the fact ([LO22, Fact 2]) that for every universal Turing machine U, every $x, y \in \{0,1\}^*$ and every time bound t, we have $\mathrm{pK}_U^t(y \mid x) \leq \mathrm{rK}_U^t(y \mid x)$. It is easy to verify that this is still true with respect to our notions of sublinear-time-bounded Kolmogorov complexity.

(4) ⇒ (1): Let $q(n)$ be any polynomial, $f\colon \{0,1\}^n \to \{0,1\}^{q(n)}$ be any function, and $\mathcal{A}$ be a randomised algorithm running in $p(n)$ time. We claim that for every input $x \in \{0,1\}^n$, if $\mathrm{rK}^{p(n)^2}(f(x) \mid x) > |\mathcal{A}| + \omega(1)$, then $\mathcal{A}$ fails to locally compute $f(x)$. Indeed, if $\mathcal{A}$ locally computes $f(x)$, then

$$\forall i \in [|x|+1], \Pr_{r\sim\{0,1\}^{p(n)}}[\mathcal{A}(x,i;r) = U(1^{p(n)}, \langle \mathcal{A}\rangle, r, x, i) = f(x)_i] \geq 2/3.$$

Clearly, this means that $\mathrm{rK}^{p(n)^2}(f(x) \mid x) \leq |\mathcal{A}| + O(1)$.

Suppose that f is a deterministic polynomial-time algorithm that given an input $x \in \{0,1\}^n$, outputs a string $y \in \{0,1\}^{q(n)}$ such that $\mathrm{rK}^{p(n)^2}(y \mid x) \geq 2\log p(n)$. It follows directly that every randomised algorithm running in $p(n)$ time only locally computes $f(x)$ for finitely many inputs

$x \in \{0,1\}^*$ (as long as the algorithm admits a constant-size description). Hence f is almost-all-inputs hard against $p(n)$-time randomised algorithms. □

Given the above equivalence, it is easy to see that Conjecture 8.4.15 and Conjecture 8.4.16 are equivalent, since Conjecture 8.4.15 asserts the equivalence between (2) and $\mathsf{prBPP} = \mathsf{prP}$, while Conjecture 8.4.16 asserts the equivalence between (1) and $\mathsf{prBPP} = \mathsf{prP}$.

8.5 Properties of the PSPACE-Complete Language

In this section, we discuss the proof of the following result.

Theorem 8.3.1 (A PSPACE-Complete Language with Useful Properties)**.** *There is a language $L^\star \subseteq \{0,1\}^*$ with the following properties:*

1. **(Complexity Upper Bound)** $L^\star \in \mathsf{PSPACE}$.
2. **(Completeness)** $L^\star$ *is* PSPACE*-hard under* DLOGTIME*-uniform projection reductions.*
3. **(Instance Checkability)** *There is a* DLOGTIME*-uniform family of* $\mathsf{BP}\text{-}\mathsf{AC}^0[\oplus]$ *oracle circuits $\{\mathsf{IC}_n\}_{n\geq 1}$ making projection queries such that, on every input string $x \in \{0,1\}^n$ and for every oracle $\mathcal{O} \subseteq \{0,1\}^*$, the following holds:*
   - $\mathsf{IC}_n^{\mathcal{O}}(x)$ *only makes queries of length n to $\mathcal{O}$.*
   - *If $\mathcal{O}$ agrees with $L^\star$ on inputs of length n, then $\Pr_r[\mathsf{IC}_n^{\mathcal{O}}(x;r) = L^\star(x)] = 1$.*
   - *For every oracle $\mathcal{O}$, $\Pr_r[\mathsf{IC}_n^{\mathcal{O}}(x;r) \in \{\perp, L^\star(x)\}] \geq 1 - \exp(-n)$.*

To establish this theorem, we verify that the language called $L^{\mathsf{WH\text{-}TV}}$ described in [Che23, Section 7] satisfies the properties we need.[16] In particular, [Che23] showed that this language is PSPACE-complete and has instance checkers in $\mathsf{AC}^0[\oplus]$. However, [Che23] only considered P-uniformity. For both the instance checker and the PSPACE-hardness reduction, a few minor modifications of the construction are needed to achieve DLOGTIME-uniformity.

The rest of this section will verify the required uniformity conditions by inspecting the proof in [Che23, Section 7]. Note that we will assume familiarity with [Che23, Section 7].[17]

8.5.1 Preliminaries

We assume familiarity with notations in [Che23, Section 7] such as pw_ℓ, ℓ_n, sz_n, and $\mathbb{F}_n$. We will use length-sz_n strings and elements in $\mathbb{F}_n$ interchangeably (instead of explicitly going through the bijection κ_n). The following tasks can be computed in DLOGTIME-uniform $\mathsf{AC}^0[\oplus]$ [HV06]:

- (Iterated addition.) Given $\alpha_1, \ldots, \alpha_t \in \{0,1\}^{\mathsf{sz}_n}$, compute $\sum_{i\in[t]} a_i \in \{0,1\}^{\mathsf{sz}_n}$.
- (Iterated multiplication.) Given $\alpha_1, \ldots, \alpha_t \in \{0,1\}^{\mathsf{sz}_n}$ where $t \leq \log n$, compute $\prod_{i\in[t]} a_i \in \{0,1\}^{\mathsf{sz}_n}$.

[16] We work with $L^{\mathsf{WH\text{-}TV}}$ instead of the final language L^{PSPACE} because the only difference between $L^{\mathsf{WH\text{-}TV}}$ and L^{PSPACE} is *paddability*, which is not required for our arguments.

[17] The full version of [Che23] can be found at ECCC Report TR22-183.

We will need a DLOGTIME-uniform $\mathsf{AC}^0[\oplus]$ circuit for *polynomial interpolation* over $\mathbb{F}_n$, which is the following task. Let $\alpha_1, \alpha_2, \ldots, \alpha_t$ be the lexicographically smallest t elements in $\mathbb{F}_n$. Given $\beta_1, \beta_2, \ldots, \beta_t \in \mathbb{F}_n$ and $z \in \mathbb{F}_n$ as inputs, the goal is to output $p(z)$, where $p : \mathbb{F}_n \to \mathbb{F}_n$ is the unique degree-$(t-1)$ polynomial over $\mathbb{F}_n$ such that $p(\alpha_i) = \beta_i$ for every $i \in [t]$.

It is shown in [Che23, Corollary 7.2] that the polynomial interpolation problem admits a uniform $\mathsf{AC}^0[\oplus]$ circuit when $t \leq \log n$. Jumping ahead, the circuit is not DLOGTIME-uniform since (8.3) requires one to compute the inverse of $\alpha_i - \alpha_j$, and it is unclear how to compute inverses over $\mathbb{F}_n$ in DLOGTIME-uniform $\mathsf{AC}^0[\oplus]$. One way to work around this technical issue is to let the interpolation algorithm output two numbers $u, v \in \mathbb{F}_n$ such that $p(z) = u \cdot v^{-1}$.

Claim 8.5.1. *For any constant $t \geq 1$[18], there is a DLOGTIME-uniform $\mathsf{AC}^0[\oplus]$ circuit that given $z, \beta_1, \ldots, \beta_t \in \mathbb{F}_n$ as inputs, outputs two elements $u, v \in \mathbb{F}_n$ such that $p(z) = u \cdot v^{-1}$, where $p : \mathbb{F}_n \to \mathbb{F}_n$ is the unique degree-$(t-1)$ polynomial over $\mathbb{F}_n$ such that $p(\alpha_i) = \beta_i$ for every $i \in [t]$.*

Proof Sketch. The expression for $p(z)$ is

$$p(z) = \sum_{i \in [t]} \beta_i \cdot \prod_{j \in [t] \setminus \{i\}} \frac{z - \alpha_j}{\alpha_i - \alpha_j}. \tag{8.3}$$

Hence, we have $p(z) = u \cdot v^{-1}$ where

$$\begin{aligned} v &= \prod_{1 \leq i < j \leq t} (\alpha_i - \alpha_j), \text{ and} \\ u = p(z)v &= \sum_{i \in [t]} \beta_i \cdot \prod_{j \in [t] \setminus \{i\}} (z - \alpha_j) \cdot \prod_{\substack{1 \leq i' < j' \leq t \\ i' \neq i \text{ and } j' \neq i}} (\alpha_i - \alpha_j). \end{aligned}$$

(Note that we omitted some $(-1)^i$ terms since our field has characteristic 2.)

The desired DLOGTIME-uniform $\mathsf{AC}^0[\oplus]$ circuit follows from [HV06]. □

8.5.2 The Instance Checker

We assume familiarity with the notations in [Che23, Section 7], such as $S_Q, f_{n,i}, J_{n,j}, Q_{n,j}$. We start by showing that [Che23, Algorithm 7.1] (i.e., the instance checker for the polynomials $\{f_{n,i}\}$ defined in [Che23, Lemma 7.3]) admits a DLOGTIME-uniform family of BP-$\mathsf{AC}^0[\oplus]$ oracle circuits. The instance checker receives input parameters $n, i \in \mathbb{N}$ such that $1 \leq i \leq n$, input $\vec{x} \in \mathbb{F}_n^n$, and oracle access to $n-i+1$ functions $\tilde{f}_i, \tilde{f}_{i+1}, \ldots, \tilde{f}_n \colon \mathbb{F}_n^n \to \mathbb{F}_n$. It draws $z_i, z_{i+1}, \ldots, z_{n-1} \leftarrow \mathbb{F}_n$ uniformly at random and performs the following steps:

1. First, it computes $\vec{\alpha}_i, \vec{\alpha}_{i+1}, \ldots, \vec{\alpha}_n \in \mathbb{F}_n^n$ as in [Che23, Eq. (8)]. For each $i \leq j \leq n$ and $\ell \in [n]$, let j_{max} be the maximum $j' < j$ such that $j' \geq i$, $J_{n,j'} = \ell$ and $Q_{n,j'} \neq \mathsf{MUL}$ (or $\perp$ if such j' does not exist). If j_{max} does not exist, then $(\vec{\alpha}_j)_\ell = x_\ell$; otherwise $(\vec{\alpha}_j)_\ell = z_{j_{\mathsf{max}}}$. We will show in Claim 8.5.2 that given j and ℓ, we can compute j_{max} in $O(\log n)$ time; hence, we can compute each $\vec{\alpha}_j$ via a DLOGTIME-uniform projection.

2. Then, it queries the oracles to obtain $r_j = \tilde{f}_j(\vec{\alpha}_j)$ for every $i \leq j \leq n$.

[18] Actually, solving the polynomial interpolation problem for $t = 3$ suffices for our instance checker.

3. Let $t := c_{\mathsf{deg}} + 1 = 3$ (by [Che23, Lemma 7.7], the polynomials have individual degree at most 2), $w_1, \ldots, w_t$ be the first t non-zero elements of $\mathbb{F}_n$. For each $i \le j < n$ and $\ell \in [t]$, it queries the oracles to obtain $\beta_\ell^j := \tilde{f}_{j+1}((\vec{\alpha}_j)^{J_{n,j} \leftarrow w_\ell})$.

4. For every $i \le j < n$ *in parallel*:

   - If $Q_{n,j} = \mathsf{MUL}$, then it verifies that $r_j = r_{j+1} \cdot \mathsf{Term}_{n,j}(\vec{\alpha}_j)$, where $\mathsf{Term}_{n,j}$ is defined in [Che23, Eq. (4)] and hence computable by DLOGTIME-uniform $\mathsf{AC}^0[\oplus]$ circuits.
   - Otherwise, let p be the unique degree-$(t-1)$ polynomial such that $p(w_\ell) = \beta_\ell^j$ for every $\ell \in [t]$, we can use Claim 8.5.1 to obtain the values $p(0)$, $p(1)$, and $p(z_j)$. If $r_j \neq S_{Q_{n,j}}((\vec{\alpha}_j)_{J_{n,j}}, p(0), p(1))$ or $r_{j+1} \neq p(z_j)$, then output $\perp$ and halt.

5. Finally, if $r_n = 1$ then accept and output r_i $(= \tilde{f}(\vec{x}))$; otherwise output $\perp$.

We remark that the values $p(0), p(1), p(z_j)$ in Item 4 are represented as $u \cdot v^{-1}$ for some $u, v \in \mathbb{F}_n$, but it is still possible to check the equalities. For example:

- If $Q = \exists$, then $S_Q(x, y_0, y_1) = y_0 \cdot y_1$, hence $S_Q(x, y_0 z_0^{-1}, y_1 z_1^{-1}) = r$ if and only if $y_0 y_1 = r z_0 z_1$.
- If $Q = \forall$, then $S_Q(x, y_0, y_1) = 1 - (1 - y_0)(1 - y_1)$, hence $S_Q(x, y_0 z_0^{-1}, y_1 z_1^{-1}) = r$ if and only if $(z_0 - y_0)(z_1 - y_1) = z_0 z_1 (1 - r)$.
- If $Q = \mathsf{LIN}$, then $S_Q(x, y_0, y_1) = xy_1 + (1-x)y_0$, hence $S_Q(x, y_0 z_0^{-1}, y_1 z_1^{-1}) = r$ if and only if $xy_1 z_0 + (1-x) y_0 z_1 = r z_0 z_1$.

The first three steps above issue queries to the oracles $\tilde{f}_i, \tilde{f}_{i+1}, \ldots, \tilde{f}_n$, and it is easy to see that these queries can be generated by a DLOGTIME-uniform projection over $\vec{x}$ and $\vec{z} = (z_i, \ldots, z_{n-1})$. The last two steps above perform DLOGTIME-uniform $\mathsf{AC}^0[\oplus]$ computation over $\vec{z}$ and the answers returned from the oracles. This establishes the complexity of the instance checker.

Finally, we need to compute j_{max} efficiently:

Claim 8.5.2. *There is an algorithm running in $O(\log n)$ time that given $i < j \le n$ and $\ell \in [n]$, finds the maximum $j' < j$ such that $j' \ge i$, $J_{n,j'} = \ell$, and $Q_{n,j'} \neq \mathsf{MUL}$.*

Proof. We recall the definitions of $J_{n,j}$ and $Q_{n,j}$ from [Che23, Proof of Lemma 7.6]. For some integers $m < \lambda < \sqrt{n}$ computable in $O(\log n)$ time, we have:

$$(J_{n,j}, Q_{n,j}) = \begin{cases} (1, \mathsf{LIN}) & \text{if } j > \lambda^2, \\ (j \bmod \lambda, \mathsf{LIN}) & \text{otherwise, if } \lambda \nmid j, \\ (1, \mathsf{MUL}) & \text{otherwise, if } j \ge (m+1)\lambda, \\ (j/\lambda, Q_{j/\lambda} \in \{\exists, \forall\}) & \text{otherwise.} \end{cases}$$

If $\ell \ge \lambda$, then we can safely return $\perp$. If $\ell = 1$ and $j > \lambda^2 + 1$, then we can simply return $j_{\mathsf{max}} = j - 1$. Otherwise, there are only two possible candidates for j_{max}:

- The case that $(J_{n,j}, Q_{n,j}) = (j \bmod \lambda, \mathsf{LIN})$: the largest $j' < \min\{j, \lambda^2\}$ s.t. $j \bmod \lambda = \ell$;

- The case that $(J_{n,j}, Q_{n,j}) = (j/\lambda, Q_{j/\lambda})$: $j' = \ell \cdot \lambda$.

We discard any candidate not in the range $[i, j)$ and return the (larger) remaining candidate j'. If neither candidate is in $[i, j)$, then we return $\perp$. This finishes the algorithm for finding j_{max} in $O(\log n)$ time. □

The instance checker for $L^{\mathsf{WH\text{-}TV}}$ reduces to the instance checker for the polynomials $\{f_{n,i}\}$ in a straightforward way. In fact, let $\mathit{input} \in \{0,1\}^m$ be the input of $L^{\mathsf{WH\text{-}TV}}$ and k be the integer defined in [Che23, Algorithm 7.2] (computable in $O(\log m)$ time), then:

- given access to (a purported oracle for) the m-th slice of $L^{\mathsf{WH\text{-}TV}}$, one can access the polynomials $f_{n_k,i_k}, f_{n_k,i_k+1}, \ldots, f_{n_k,n_k}$ via $\mathsf{DLOGTIME}$-uniform projections;
- given input and the answers of each $f_{n_k,j}(\vec{x})$ $(j \geq i_k)$, where $\vec{x} \in \mathbb{F}_n^n$ corresponds to the length-$(n \cdot \mathsf{sz}_n)$ prefix of input, one can compute $L^{\mathsf{WH\text{-}TV}}(\mathit{input})$ using [Che23, Eq. (10) and (11)] via a $\mathsf{DLOGTIME}$-uniform $\mathsf{AC}^0[\oplus]$ circuit.

Since the instance checker for $\{f_{n,i}\}$ is a $\mathsf{DLOGTIME}$-uniform $\mathsf{BP\text{-}AC}^0[\oplus]$ circuit making projection queries, so is the instance checker for $L^{\mathsf{WH\text{-}TV}}$. Finally, the error probability of the instance checker for $\{f_{n,i}\}$ is at most $\mathrm{poly}(n)/2^n$ by [Che23, Claim 7.12], hence the error probability of the instance checker for $L^{\mathsf{WH\text{-}TV}}$ is also at most $\mathrm{poly}(n)/2^n$ by a union bound.

8.5.3 PSPACE-Completeness

We also need to show that $L^{\mathsf{WH\text{-}TV}}$ is PSPACE-complete under $\mathsf{DLOGTIME}$-uniform projections. Note that $L^{\mathsf{WH\text{-}TV}}$ is an arithmetisation of the problem $\mathsf{TQBF}^{\mathsf{u}}$ defined in [Che23, Section 7.3], thus we first show that $\mathsf{TQBF}^{\mathsf{u}}$ is PSPACE-complete under $\mathsf{DLOGTIME}$-uniform projections, and then reduce $\mathsf{TQBF}^{\mathsf{u}}$ to $L^{\mathsf{WH\text{-}TV}}$ using $\mathsf{DLOGTIME}$-uniform projections. However, the PSPACE-completeness reduction presented in [Che23, Section 7.3] (from the classical TQBF to $\mathsf{TQBF}^{\mathsf{u}}$) is not a projection, so we need to implement the reduction more efficiently here.

Definition of $\mathsf{TQBF}^{\mathsf{u}}$. There are $8 \cdot \binom{n}{3}$ possible width-3 clauses on n variables and we let $\phi_n^{\mathsf{cl\text{-}idx}}$ be a bijection between $[8 \cdot \binom{n}{3}]$ and the set of valid width-3 clauses. A 3-CNF ϕ can thus be described by a string $y \in \{0,1\}^{8 \cdot \binom{n}{3}}$. The $\mathsf{TQBF}^{\mathsf{u}}$ problem takes such a bit-string as an input, constructs the corresponding 3-CNF $\Phi(x_1, x_2, \ldots, x_n)$, and outputs

$$Q_1 x_1 Q_2 x_2 \ldots Q_n x_n \ \Phi(x_1, x_2, \ldots, x_n), \tag{8.4}$$

where Q_i equals $\exists$ for odd i and $\forall$ for even i.

PSPACE-completeness of $\mathsf{TQBF}^{\mathsf{u}}$. First, the proof of the PSPACE-completeness of TQBF (that is, computing (8.4) when Φ is a *general circuit*) actually shows the following stronger result: For every language $L \in \mathsf{PSPACE}$, there is a polynomial-time Turing machine M and a polynomial $\ell(n)$ such that, for every $z \in \{0,1\}^n$, $z \in L$ if and only if

$$Q_1 x_1 Q_2 x_2 \ldots Q_{\ell(n)} x_{\ell(n)} \ M(z_{1\sim n}, x_{1\sim \ell(n)}),$$

where, again, Q_i equals $\exists$ for odd i and $\forall$ for even i. (See [ALR99] for an exposition. In particular, [ALR99, Section 6.2] pointed out that the above PSPACE-completeness reduction can be computed by a DLOGTIME-uniform projection.)

Since P is equal to DLOGTIME-uniform SIZE[poly] [BI94], there is a family of poly(n)-size circuits $\{C_n \colon \{0,1\}^{n+\ell(n)\to\{0,1\}}\}$ that simulates M and satisfies the following uniformity conditions. Let $s(n) \le \mathrm{poly}(n)$ denote the number of gates in C_n (including input gates), then $1, 2, \ldots, s$ is a valid topological order of C_n (the first $n+\ell$ gates are inputs and the s-th gate is the output gate). By adding dummy gates, we may assume that no gate has both children being z_i variables (this will imply that our final is 1-local). Finally, the direct connection language of C_n can be computed in $O(\log n)$ time: there is an algorithm running in deterministic $O(\log n)$ time that given n, indices of gates $g_1, g_2, g_3 \in [|C_n|]$ (where $g_1 > \max\{g_2, g_3\}$), and assignments $b_1, b_2, b_3 \in \{0,1\}$, returns true if and only if the outputs of g_2 and g_3 are fed as inputs of g_1 and $\{g_i = b_i\}_{i\in[3]}$ is consistent with the gate type of g_1 (e.g., if g_1 is an AND gate, then it cannot be the case that $b_1 = 1$ but $b_2 = 0$).

Now we show that this implies a DLOGTIME-uniform projection reduction from L to $\mathsf{TQBF}^{\mathsf{u}}$. We will reduce an instance $z \in \{0,1\}^n$ of L to a $\mathsf{TQBF}^{\mathsf{u}}$ instance Φ_z with $s(n)$ variables. Let D be a clause expressing

$$(g_i \neq b_i) \vee (g_j \neq b_j) \vee (g_k \neq b_k),$$

where g_i, g_j, g_k are variables corresponding to gates in C_n and $b_i, b_j, b_k \in \{0,1\}$. We may assume that $g_i > \max\{g_j, g_k\}$. Then, D appears in Φ if and only if g_j, g_k are fed as inputs of g_i but $\{g_i = b_i\}_{i\in[3]}$ is *inconsistent* with the gate type of g_i. This information can be retrieved by $O(1)$ queries to the direct connection language.

Finally, the $\mathsf{TQBF}^{\mathsf{u}}$ instance we produced is

$$Q_1 x_1 Q_2 x_2 \ldots Q_{\ell(n)} x_{\ell(n)} \exists g_{(\ell(n)+n+1)\sim s(n)}\ \Phi(z_{1\sim n}, x_{1\sim\ell(n)}, g_{(\ell(n)+n+1)\sim s(n)}),$$

and we may insert $\forall$ quantifiers among $g_{(\ell(n)+n+1)\sim s(n)}$ to make the quantifiers alternate. It is easy to see that the reduction is computable in DLOGTIME; it is a projection because every clause (i.e., gate in C_n) only touches one z_i variable.

PSPACE-completeness of $L^{\mathsf{WH\text{-}TV}}$. The reduction from $\mathsf{TQBF}^{\mathsf{u}}$ to $L^{\mathsf{WH\text{-}TV}}$ is straightforward. First, by [Che23, Lemma 7.10], the truth-table of $\mathsf{TQBF}^{\mathsf{u}}$ on input length m coincides with the truth-table of $f_{n,1} = g_{1,1}^{(n)}$ over the Boolean cube, for some $n = \mathrm{poly}(m)$. Hence, when $L = \mathsf{TQBF}^{\mathsf{u}}$, the algorithm A_L^{red} (as defined in Item 4 of [Che23, Lemma 7.3]) is a DLOGTIME-uniform projection. Second, the reduction in [Che23, Lemma 7.19] produces the instance $(\vec{z}, \vec{y}, \vec{u})$ where $\vec{y}$ and $\vec{u}$ are constant vectors whose each bit can be computed trivially, and $\vec{z} = A_L^{\mathsf{red}}(x)$ and x is the input of L. Hence, when $L = \mathsf{TQBF}^{\mathsf{u}}$, this reduction is also a DLOGTIME-uniform projection over x.

Combining all of the above, we can see that the overall reduction from any language $L \in$ PSPACE to $L^{\mathsf{WH\text{-}TV}}$ is a DLOGTIME-uniform projection.

Chapter 9

Hardness of Range Avoidance from Demi-Bits

9.1 Introduction

We make progress on the hardness of the *range avoidance problem* and the existence of *proof complexity generators*. We begin with a brief overview of these two lines of research.

9.1.1 Range Avoidance

It is easy to see that the range avoidance problem admits a trivial randomised algorithm: given a circuit $G : \{0,1\}^n \to \{0,1\}^m$ as input, a uniformly random m-bit string would be a valid output with high probability. On the other hand, deterministic algorithms for Avoid would imply breakthroughs in explicit constructions and circuit lower bounds [Kor21, RSW22, GLW22, GGNS23]. Since such breakthroughs are widely believed to be *true* (albeit *difficult to prove*), the aforementioned results only suggest that deterministic algorithms for Avoid would be difficult to obtain *unconditionally*, rather than that such algorithms are *unlikely to exist*. This raises a natural question: Is there a deterministic algorithm for Avoid?

Perhaps surprisingly, recent results suggested that the answer is likely *no* under plausible cryptographic assumptions. Ilango, Li, and Williams [ILW23] showed that Avoid is hard for deterministic algorithms assuming the existence of subexponentially secure indistinguishability obfuscation ($i\mathcal{O}$) and that $\mathsf{NP} \neq \mathsf{coNP}$. Chen and Li [CL24] extended this result and showed that Avoid is hard even for *nondeterministic* algorithms, under certain assumptions regarding the nondeterministic hardness of LWE (Learning with Errors) or LPN (Learning Parity with Noise). In addition to providing compelling evidence for the hardness of Avoid, these results establish a strong separation between deterministic and randomised algorithms (recall that there exists a trivial randomised algorithm for Avoid).

The hardness results in [ILW23, CL24] open up several exciting research directions:

1. **Can the hardness of range avoidance be based on weaker (or alternative) assumptions?**

   The assumptions used in prior work, i.e., $i\mathcal{O}$ [ILW23] and public-key encryption [CL24], belong to Cryptomania in the terminology of Impagliazzo's worlds [Imp95]. Can we base

the hardness of range avoidance on assumptions of a "Minicrypt" flavor, such as one-way functions or pseudorandom generators? Additionally, both [ILW23] and [CL24] rely on *subexponential* indistinguishability assumptions[1]. Are such subexponential assumptions necessary?

2. **Can we obtain hardness of Avoid for instances computed by restricted circuits?**

   Previously, under assumptions related to LWE, Chen and Li [CL24] showed that Avoid remains hard even when each output bit of G is computed by a so-called "$\mathsf{DOR} \circ \mathsf{MAJ} \circ \mathsf{AND}_{O(\log n)}$ circuit". No such results were known for other restricted circuit classes. The related *remote point* problem has been shown to be hard under LPN-style assumptions for $\mathsf{XOR} \circ \mathsf{AND}_{O(\log n)}$ (i.e., $O(\log n)$-degree polynomials over $\mathbb{F}_2$) [CL24].

This chapter makes progress on both fronts. We show that Avoid is hard for nondeterministic algorithms under the existence of *demi-bits generators* with sufficient stretch[2] [Rud97]. A formal definition of demi-bits generators is deferred to Section 9.2.1, and candidate constructions supporting their existence are discussed in Section 9.5. For the purpose of this introduction, it suffices to keep in mind that demi-bits generators are a version of cryptographic pseudorandom generators secure against nondeterministic adversaries.

We highlight two key features of our results here:

1. **Minicrypt-style assumptions against nondeterministic adversaries.**

   Roughly speaking, demi-bits generators are (cryptographic) pseudorandom generators secure against nondeterministic adversaries[3]. They are arguably a natural "Minicrypt" analogue of pseudorandom generators in the context of cryptography against nondeterministic adversaries. Moreover, our results only rely on the *super-polynomial* hardness of these demi-bits generators, thereby completely getting rid of the subexponential (or "JLS"-style) assumptions used in prior work.

2. **Hardness for restricted circuit classes.**

   Under the assumption that certain concrete demi-bits generators are secure (e.g., those based on LPN or Goldreich's PRG), we show that the range avoidance problem remains hard for nondeterministic algorithms even when the underlying circuits belong to $\mathsf{XOR} \circ \mathsf{AND}_{O(1)}$, i.e., constant-degree polynomials over $\mathbb{F}_2$.

9.1.2 Proof Complexity Generators

Let $G : \{0,1\}^n \to \{0,1\}^m$ be a Boolean circuit where $m > n$, and $\mathcal{P}$ be a propositional proof system. We say that G is a (secure) *proof complexity generator* [ABRW04, Kra01a] against $\mathcal{P}$ if,

[1] More precisely, the assumptions in [ILW23, CL24] assert subexponential indistinguishability against polynomial-time adversaries. This level of security is referred to as "JLS-security" in [ILW23], where "JLS" comes from the strengths of the "well-founded" assumptions used to construct $i\mathcal{O}$ in [JLS21].

[2] The stretchability of generic demi-bits generators is only partially understood. Recent work of Tzameret and Zhang [TZ24] shows that demi-bits generator with 1-bit stretch $G : \{0,1\}^n \to \{0,1\}^{n+1}$ implies those with a *sublinear* bits of stretch $G' : \{0,1\}^n \to \{0,1\}^{n+n^c}$ for any constant $0 < c < 1$. This is the first proof that generic demi-bits generators are stretchable at all, but it still falls short of the *linear* or *polynomial* stretch assumed in our hypothesis.

[3] In fact, Rudich [Rud97] introduced two ways to define pseudorandomness against nondeterministic adversaries: super-bits and demi-bits. Demi-bits are weaker than super-bits.

for every string $y \in \{0,1\}^m$, the (properly encoded) statement "$y \notin \mathrm{Range}(G)$" does not admit short proofs in $\mathcal{P}$.[4] A comprehensive survey about proof complexity generators can be found in [Kra25].

The study of proof complexity generators is motivated by at least the following themes:

1. **Pseudorandomness in proof complexity [ABRW04].**

   A standard *pseudorandom generator* $G : \{0,1\}^n \to \{0,1\}^m$ [Yao82] fools a (polynomial-time) algorithm $\mathcal{D}$ if $\mathcal{D}$ cannot distinguish the outputs of G from truly random m-bit strings; that is, $\mathcal{D}(\mathcal{U}_m) \approx \mathcal{D}(G(\mathcal{U}_n))$, where $\mathcal{U}_\ell$ denotes the uniform distribution over ℓ-bit strings. Analogously, one can say that G fools a propositional proof system $\mathcal{P}$ if $\mathcal{P}$ cannot distinguish between the outputs of G and truly random m-bit strings, and a natural way of formalising this is to say that $\mathcal{P}$ cannot efficiently prove any string outside the range of G.

   Following the idea of pseudorandomness in proof complexity, subsequent works [Kra04, Pic11,Kra11,Raz15,Kha22] studied the hardness of the *Nisan–Wigderson* generator [NW94] as a proof complexity generator in various settings. In particular, an influential conjecture of Razborov [Raz15, Conjecture 2] asserts that the Nisan–Wigderson generator based on any "sufficiently hard" function in $\mathsf{NP} \cap \mathsf{coNP}$ is a proof complexity generator against Extended Frege; that is, computational hardness can be transformed into proof complexity pseudorandomness.

2. **Candidate hard tautologies for strong proof systems.**

   There are two difficulties in proving lower bounds for strong proof systems such as Frege and Extended Frege: the lack of techniques and the lack of candidate hard tautologies. The latter problem was highlighted by Bonet, Buss, and Pitassi [BBP95], who demonstrated that many combinatorial tautologies can be proved efficiently in Frege, hence disqualifying them as hard candidates. This issue has been further discussed in [Kra01b, Kra04, ST21, Kha22].

   Tautologies from proof complexity generators are among the few natural candidates that appear hard for strong proof systems. It seems plausible that for some mapping $G : \{0,1\}^n \to \{0,1\}^{n+1}$ and some (or even *every*) $y \in \{0,1\}^{n+1} \setminus \mathrm{Range}(G)$, the natural CNF encoding of the tautology "$y \notin \mathrm{Range}(G)$" requires super-polynomially long Extended Frege proofs.

3. **Unprovability of circuit lower bounds.**

   Given our very limited progress in circuit complexity, it is tempting to conjecture that circuit lower bounds are hard to prove in formal proof systems. For a Boolean function $f : \{0,1\}^n \to \{0,1\}$ and a size parameter s, one can write down a propositional formula $\mathsf{lb}(f,s)$ (of size $2^{O(n)}$) asserting that no circuit of size at most s computes f. The proof complexity of such formulas has been studied extensively [Raz98, Raz04, Kra04, Raz15,

[4] If y is in fact in the range of G, then "$y \notin \mathrm{Range}(G)$" is a false statement and hence has no proof in any sound proof system. Therefore, this requirement is equivalent to that, for every $y \notin \mathrm{Range}(G)$, the tautology "$y \notin \mathrm{Range}(G)$" is hard to prove in $\mathcal{P}$.

Pic15, PS19, ST21, PS22], due to its implications for the metamathematics of complexity theory.

Consider the *truth table generator* $\mathsf{TT} : \{0,1\}^{\mathrm{poly}(s)} \to \{0,1\}^{2^n}$, which maps a size-$s$ circuit C to its 2^n-bit truth table. By definition, TT is a proof complexity generator against a proof system $\mathcal{P}$ if and only if $\mathcal{P}$ cannot efficiently prove any circuit lower bound $\mathsf{lb}(f, s)$. Krajíček [Kra04] introduced the notion of *pseudo-surjectivity* and showed that TT is the hardest pseudo-surjective generator: The existence of any generator pseudo-surjective against $\mathcal{P}$ implies that TT is pseudo-surjective against $\mathcal{P}$ (and thus that $\mathcal{P}$ cannot prove circuit lower bounds). Razborov [Raz15] further showed unprovability of circuit lower bounds in the proof system $\mathrm{Res}(\varepsilon \log\log N)$ by exhibiting a proof complexity generator that is iterable for this system.[5]

Krajíček [Kra04, Kra23, Kra24] conjectured that there exists a proof complexity generator that is secure against every propositional proof system. One could also consider a slightly weaker conjecture that for every propositional proof system $\mathcal{P}$, there is a proof complexity generator $C_{\mathcal{P}}$ (possibly depending on $\mathcal{P}$) that is hard against $\mathcal{P}$. At first glance, these conjectures may appear unrelated to standard hardness assumptions in complexity theory or cryptography, as proof complexity generators require "$y \notin \mathrm{Range}(G)$" to be hard to prove for *every* y (i.e., the *best-case* y), while complexity-theoretic or cryptographic hardness assumptions tend to be either *worst-case* or *average-case*. We elaborate on the notion of "best-case" proof complexity in Section 9.1.4.

In this chapter, we give strong evidence for the weaker conjecture by showing that it follows from the existence of demi-bits generators (with sufficiently large stretch) [Rud97]. The latter is a natural and fundamental assumption in the study of *cryptography against nondeterministic adversaries*. Furthermore, we show that our generators are even pseudo-surjective under certain regimes.[6]

9.1.3 Our Results

Hardness of range avoidance. Our main result is that the existence of demi-bits generators implies that Avoid $\notin \mathsf{SearchNP}$, i.e., Avoid is hard for nondeterministic search algorithms.

Theorem 9.1.1 (Main)**.** *If there exists a demi-bits generator* $G : \{0,1\}^n \to \{0,1\}^{10n}$*, then* Avoid $\notin \mathsf{SearchNP}$.

In fact, we show that composing the demi-bits generator with a hash function in some pairwise independent hash family would yield a hard instance for Avoid. In its full generality, our arguments hold for arbitrary *strong seeded extractors*, and the theorem below follows from the leftover hash lemma [ILL89], which guarantees that pairwise independent hash families are such extractors; see Theorem 9.3.1 for details.

We present the version using pairwise independent hash families here due to its elegance:

[5] Iterability is a weaker notion than pseudo-surjectivity. Krajíček [Kra04] also showed that TT is the "hardest" iterable generator. Therefore, the existence of a proof complexity generator iterable for some proof system implies that TT is also iterable (and thus hard) for this proof system.

[6] The parameters of our pseudo-surjectivity results *fall just short* of those required to apply Krajíček's result [Kra04], hence they do not imply the hardness of the truth table generator. This limitation is inherent; we discuss this issue in more detail after presenting Theorem 9.1.7.

Theorem 9.1.2. *Let $G : \{0,1\}^n \to \{0,1\}^N$ be a demi-bits generator, $\mathcal{H} = \{h : \{0,1\}^N \to \{0,1\}^m\}$ be a family of pairwise independent hash functions, and $\mathcal{A}$ be a nondeterministic polynomial-time algorithm. If $N > 10m > n$, then there exists $h \in \mathcal{H}$ such that $\mathcal{A}$ fails to solve the range avoidance problem on the input $h \circ G$.*

As discussed before, this result improves upon [ILW23,CL24] in several key aspects. First, we only require super-polynomial hardness of the demi-bits generators, thereby completely eliminating the subexponential- or JLS-hardness assumptions. Second, our assumptions are solely based on the existence of demi-bits generators, a primitive arguably situated within "nondeterministic Minicrypt." Finally, by instantiating the extractors with pairwise independent hash functions computable by linear transformations over $\mathbb{F}_2$, and using demi-bits generators computable by constant-degree $\mathbb{F}_2$-polynomials, we establish the hardness of AVOID even for circuits where each output bit is computable in constant $\mathbb{F}_2$-degree (i.e., $\mathsf{XOR} \circ \mathsf{AND}_{O(1)}$ circuits):

Corollary 9.1.3 (Informal)**.** *Assuming the existence of demi-bits generators computable in $\mathsf{XOR} \circ \mathsf{AND}_{O(1)}$ (Assumption 9.2.3), the range avoidance problem for $\mathsf{XOR} \circ \mathsf{AND}_{O(1)}$ circuits is not in $\mathsf{SearchNP}$.*

Proof complexity generators. Building on this result, we show that for any fixed propositional proof system $\mathcal{P}$ closed under certain reductions, demi-bits generators for $\mathcal{P}$ imply proof complexity generators for $\mathcal{P}$. In particular, the existence of demi-bits generators secure against $\mathsf{NP}/_{\mathrm{poly}}$ implies the weaker version of Krajíček's conjecture, providing strong evidence that the latter conjecture is true.

Moreover, this result suggests a new approach for constructing proof complexity generators for concrete proof systems closed under certain reductions, such as $\mathsf{Res}[\oplus]$: it suffices to construct demi-bits generators secure against *the same proof system*.

Theorem 9.1.4. *Let $\mathcal{P}$ be a proof system closed under parity reductions. If there exists a demi-bits generator $G : \{0,1\}^n \to \{0,1\}^{10n}$ secure against $\mathcal{P}$, then there is a (non-uniform) proof complexity generator secure against $\mathcal{P}$.*

Unprovability of dwPHP(PV) in PV from demi-bits. Our results also have implications in *bounded arithmetic*. A central goal in bounded arithmetic is to delineate the logical power required to formalise reasoning about computational complexity. Two well-studied theories in this context are Cook's theory PV_1 [Coo75], which corresponds to reasoning in deterministic polynomial time, and Jeřábek's theory APC_1 [Jeř04, Jer07][7], which extends PV_1 by adding the *dual weak pigeonhole principle* for polynomial-time functions (dwPHP(PV)), and captures aspects of randomised polynomial-time reasoning.

Despite decades of interest, it has remained open whether APC_1 and PV_1 are actually distinct theories—that is, whether dwPHP(PV) is unprovable in PV_1. Recently, Ilango, Li, and Williams [ILW23] provided the first evidence separating the two: they showed that dwPHP(PV) is unprovable in PV_1 under the assumptions that indistinguishability obfuscation ($i\mathcal{O}$) with JLS security exists and that coNP is not infinitely often in AM. We remark that the same separation

[7] Note that the terminology "APC_1" was first used in [BKT14].

was also shown by Krajíček [Kra21], albeit under an assumption that is regarded as "unlikely" (P admits fixed-polynomial size circuits). In this work, we establish the same separation assuming the existence of demi-bits generators against $\mathsf{AM}/_{O(1)}$.

Theorem 9.1.5. *Assume there exists a demi-bits generator $G : \{0,1\}^n \to \{0,1\}^{\omega(n)}$ secure against $\mathsf{AM}/_{O(1)}$. Then, the Dual Weak Pigeonhole Principle for polynomial-time functions (dwPHP(PV)) is not provable in PV. (In particular, APC_1 is a strict extension of PV_1.)*

The only property of PV_1 used in our argument is the KPT witnessing theorem [KPT91], which states that if PV_1 proves the dual weak pigeonhole principle for polynomial-time functions, then there exists a deterministic polynomial-time algorithm for solving Avoid with $O(1)$ circuit-inversion oracle queries ([ILW23, Definition 19]). Our separation result in Theorem 9.1.5 follows by showing that no such algorithm can exist. Moreover, for any parameter $k = k(n)$, assuming the existence of demi-bits generators secure against $\mathsf{AM}/_{O(\log k)}$, we further rule out deterministic polynomial-time algorithms for Avoid that make k circuit-inversion oracle queries.

Theorem 9.1.6. *Let $m = m(n) > n$ and $k = k(n)$ be parameters. If there exists a demi-bits generator $G : \{0,1\}^n \to \{0,1\}^{100km}$ secure against $\mathsf{AM}/_{O(\log k)}$, then there is no polynomial-time deterministic algorithm for Avoid on circuits with n inputs and m outputs using k circuit-inversion oracle queries.*

Pseudo-surjective proof complexity generators. Assuming demi-bits generators against $\mathsf{NP}/_{\mathrm{poly}}$, we show that our proof complexity generators are even pseudo-surjective against every proof system. (The precise definition of *k-round* pseudo-surjectivity is presented in Definition 9.2.14.)

Theorem 9.1.7. *Let $m = m(n) > n$ and $k = k(n)$ be parameters. If there exists a demi-bits generator $G : \{0,1\}^n \to \{0,1\}^{100km}$ secure against $\mathsf{NP}/_{\mathrm{poly}}$, then for every non-uniform propositional proof system $\mathcal{P}$, there is a non-uniform proof complexity generator $G_{\mathcal{P},k} : \{0,1\}^n \to \{0,1\}^m$ that is k-round pseudo-surjective against $\mathcal{P}$.*

Krajíček [Kra04] proved that, under appropriate parameter settings, the existence of pseudo-surjective proof complexity generators is equivalent to the pseudo-surjectivity of the truth table generator. As a corollary, the existence of pseudo-surjective generators against a proof system $\mathcal{P}$ implies that *every* circuit lower bound is hard to prove in $\mathcal{P}$.

However, the parameters in our Theorem 9.1.7 *fall short* of applying Krajíček's result and therefore do *not* imply the pseudo-surjectivity of the truth table generator. Specifically, to apply Krajíček's results, we need a proof complexity generator that is computable by circuits of size s and is k-round pseudo-surjective for some $k \gg s$; see the proof of [Kra04, Theorem 4.2] for detailed discussions. In contrast, Theorem 9.1.7 guarantees a generator computable by circuits of size $\mathrm{poly}(n,k)$ that is k-round pseudo-surjective, which is in the regime where $k < s$ and thus outside the reach of Krajíček's equivalence.

This limitation is inherent to the generality of our result: we construct generators secure against *all* proof systems, whereas under the assumption $\mathsf{E} \not\subseteq \mathsf{SIZE}[2^{o(n)}]$, there exists a proof system that *can* prove circuit lower bounds (e.g., by simply hardwiring an axiom that certain E-complete language has exponential circuit complexity). Thus, under this standard hardness

assumption, it is provably impossible to extend our results to the regime where $k \gg s$ and thereby obtain pseudo-surjectivity of the truth table generator. It remains an intriguing open question whether our approach can be refined to construct proof complexity generators of size s that are k-round pseudo-surjective against specific systems such as *Extended Frege*, for some $k \gg s$. Such a result would imply that Extended Frege cannot prove *any* circuit lower bounds.

Finally, we comment on the strength of the adversaries required in our assumptions on demi-bits generators. In the main theorem (Theorem 9.1.1), a SearchNP algorithm for Avoid is transformed into a nondeterministic adversary that breaks the demi-bits generator. Since SearchNP is a uniform class, it suffices to assume that the generator is secure against uniform nondeterministic adversaries. In contrast, Theorem 9.1.6 requires the generator to be secure against $\mathsf{AM}/_{O(\log k(n))}$ adversaries. This is because the adversary invokes the Goldwasser–Sipser protocol [GS86], which is in AM, and needs to hardwire the index of the circuit-inversion query that succeeds with good probability. In Theorem 9.1.7, we require security against $\mathsf{NP}/_{\mathrm{poly}}$ adversaries, as our adversary needs to hardwire a "good" sequence of teacher responses in the student-teacher game, thus it is highly non-uniform.

9.1.4 Perspective: Average-Case to Best-Case Reductions in Proof Complexity

Theoretical computer science has traditionally focused on the *worst-case* complexity of problems: An algorithm $\mathcal{A}$ solves a problem if $\mathcal{A}(x)$ succeeds on every input x. Motivated by practical heuristics (where worst-case analysis tends to be overly pessimistic) and cryptography (where worst-case hardness is not sufficient for security), *average-case* complexity has emerged as an important research direction [Imp95, BT06a]. In this setting, fixing a distribution $\mathcal{D}$ over inputs, an algorithm $\mathcal{A}$ solves a problem if $\mathcal{A}(x)$ succeeds with good probability over $x \leftarrow \mathcal{D}$. Recently, average-case complexity has received attention in proof complexity as well: For example, [Pan21, dRPR23, CdRN+23] proved proof complexity lower bounds for Clique and Coloring for *random* graphs.

An important topic in average-case complexity is *worst-case to average-case* reductions: reductions showing that if a problem L is hard in the worst-case, then a related problem L' is hard on average. Worst-case to average-case reductions are known for the Permanent [CPS99], Discrete Logarithm [BM84], Quadratic Residuosity [GM84], certain lattice problems [Ajt96], and more recently for problems in meta-complexity [Hir18]. On the other hand, "black-box" worst-case to average-case reductions are unlikely to exist for NP-complete problems [FF93, BT06b].

In contrast, the notion of *best-case* complexity has received far less attention. Perhaps one reason is that this notion is often trivial in standard computational complexity: for every language L, either the all-zero function or the all-one function can decide L on the "best" input.[8] However, in proof complexity, best-case hardness is a meaningful and natural notion, as illustrated by proof complexity generators, which are stretching functions G such that the statement "$y \notin \mathrm{Range}(G)$" is hard to prove even for the *best* choice of y.

In this context, our results can be interpreted as an *average-case to best-case* reduction in

[8] One notable exception appears in recent derandomisation results [CT21a] based on *almost-all-input* hardness assumptions. In particular, it was shown that prP = prBPP follows from the existence of depth-efficient multi-output functions with high best-case complexity against randomised algorithms.

proof complexity. Indeed, Theorem 9.1.4 transforms demi-bits generators, where statements of the form "$y \notin \mathrm{Range}(G)$" are hard to prove for an *average-case* y, to proof complexity generators, where such statements are hard for a *best-case* y.

We find the existence of such average-case to best-case reductions quite surprising. Our arguments crucially exploit the power of nondeterministic computation, and the phenomenon of average-case to best-case reductions seems unique to the setting of proof complexity and hardness against nondeterministic algorithms. We believe that further exploring the scope and limitations of average-case to best-case reductions is a promising direction for future research.

We remark that there are also worst-case to best-case reductions in proof complexity. Krajíček [Kra09] constructed a proof complexity generator whose hardness can be based on the hardness of the pigeonhole principle, thereby reducing the best-case hardness of an entire family of tautologies to that of a single tautology. Inspired by this example, Garlík, Gryaznov, Ren, and Tzameret [GGRT25] recently showed a worst-case to best-case reduction for the *rank principles*. Let "$\mathrm{rank}(A) > r$" denote the collection of polynomial equations expressing that the rank of an $n \times n$ matrix A is greater than r. If a proof system (closed under certain algebraic reductions) cannot prove "$\mathrm{rank}(I_n) > r$" where I_n is the $n \times n$ identity matrix, then it also cannot prove "$\mathrm{rank}(A) > r$" for *every* $n \times n$ matrix A.

9.2 Preliminaries

9.2.1 Demi-Bits Generators

Definition 9.2.1 (Demi-Bits Generators)**.** Let n, m be length parameters such that $n < m$. A function $G : \{0,1\}^n \to \{0,1\}^m$ is an (s, ε)-*secure demi-bits generator* if there is no $\mathsf{NP}/_{\mathrm{poly}}$ adversary Adv of size s such that

$$\Pr_{y \leftarrow \{0,1\}^m}[\mathsf{Adv}(y) = 1] \geq \varepsilon \quad \text{and} \quad \Pr_{x \leftarrow \{0,1\}^n}[\mathsf{Adv}(G(x)) = 1] = 0.$$

The results in this chapter require demi-bits generators with large stretch and computable with small circuit complexity. In particular, we need the following assumptions:

Assumption 9.2.2 (Demi-bits Generators with Polynomial Stretch)**.** For every constant $c \geq 1$, there exists a family of demi-bits generators $\{g_n : \{0,1\}^n \to \{0,1\}^{n^c}\}$ secure against $\mathsf{NP}/_{\mathrm{poly}}$.

Assumption 9.2.3 (Demi-bits Generators with $n^{1+\varepsilon}$ Stretch in Constant Degree)**.** There exist constants $\varepsilon > 0$, $d \geq 2$, and a (non-uniformly computable) family of demi-bits generators $\{g_n : \{0,1\}^n \to \{0,1\}^{n^{1+\varepsilon}}\}$ secure against $\mathsf{NP}/_{\mathrm{poly}}$, such that each output bit of g_n is computable by a degree-d polynomial over $\mathbb{F}_2$ (i.e., an $\mathsf{XOR} \circ \mathsf{AND}_d$ circuit).

Our main hardness results for Avoid will be based on Assumption 9.2.2; we also need Assumption 9.2.3 to obtain hardness results for constant-degree Avoid. In Section 9.5, we justify these assumptions and provide some candidate constructions.

9.2.2 Arthur–Merlin Protocols

An *Arthur–Merlin* protocol [Bab85] for a language L is a constant-round public-coin interactive protocol between a computationally unbounded Prover (Merlin) and a randomised polynomial-time Verifier (Arthur) that satisfies the following properties for every input x:

- **Completeness:** If $x \in L$, then there is a Prover that makes the Verifier accept w.p. $\geq 2/3$.
- **Soundness:** If $x \notin L$, then no Prover can make the Verifier accept w.p. $> 1/3$.

Let AM denote the set of languages with an Arthur–Merlin protocol. The round-collapse theorem of Babai [Bab85] implies that every language in AM actually has an Arthur–Merlin protocol with two rounds: Verifier sends the first message, Prover sends a proof, and Verifier decides whether $x \in L$. Hence, AM can be seen as a randomised version of NP; indeed, one can prove that AM = NP under circuit lower bound assumptions [KvM02, MV05, SU05, SU06].

Goldwasser–Sipser set lower bound protocol. We need the following well-known AM protocol for proving lower bounds on the size of efficiently recognisable sets.

Lemma 9.2.4 ([GS86], also see [AB09, section 8.4])**.** *There is an Arthur–Merlin protocol such that the following holds. Suppose that both* Prover *and* Verifier *receive a nondeterministic circuit* $C : \{0,1\}^n \to \{0,1\}$ *and a number* $s \leq 2^n$*. Let* $S = \{x \in \{0,1\}^n : C(x) = 1\}$*. Then*

- ***Completeness:*** *If* $|S| \geq s$*, then there exist messages* Prover *can send such that* Verifier *accepts with probability at least* $2/3$*.*
- ***Soundness:*** *If* $|S| \leq s/2$*, then regardless of what* Prover *sends,* Verifier *accepts with probability at most* $1/3$*.*

Moreover, the protocol is a two-round public-coin protocol: Verifier *first sends a random seed* r *and receives a message* m*; then it deterministically decides whether to accept based on* r *and* m *in polynomial time.*

Arthur–Merlin protocols as adversaries.

Definition 9.2.5 (Breaking demi-bits generators by AM adversaries)**.** Let $m > n$. An AM adversary breaks a demi-bits generator $G : \{0,1\}^n \to \{0,1\}^m$ if both Prover and Verifier receives a common input $y \in \{0,1\}^m$, and:

- for $\geq 1/3$ fraction of $y \in \{0,1\}^m$, there exists a Prover that makes the Verifier accepts with probability $\geq 2/3$;
- for all $y \in \text{Range}(G)$, for every Prover, the Verifier accepts with probability $\leq 1/3$.

We also consider AM adversaries with advice:

Definition 9.2.6 ($\text{AM}/_{k(n)}$ adversaries)**.** An $\text{AM}/_{k(n)}$ adversary is an Arthur–Merlin protocol where the Verifier is a probabilistic polynomial-time machine that additionally receives a $k(n)$-bit advice string a_n (which may depend on the input length n but not on the specific input y). The interaction on input y proceeds as follows:

1. The Verifier uses a_n and randomness r to generate a message to the Prover;
2. The Prover replies with a message;
3. The Verifier accepts or rejects based on y, a_n, r, and the Prover's response.

The acceptance probabilities are still defined over Verifier's internal randomness, with the advice string fixed to a_n.

Proposition 9.2.7. *Let G be a demi-bits generator.*

- *If there exists an* AM *adversary breaking G, then there exists an* $\mathsf{NP}/_{\mathrm{poly}}$ *adversary breaking G* ([Adl78]).
- *For every constant $k \geq 2$, if there exists a k-round* AM *adversary breaking G, then there exists a (standard) two-round* AM *adversary breaking G* ([Bab85]).

9.2.3 FNP v.s. SearchNP

In this chapter, we need to distinguish between the two notions FNP and SearchNP.

Definition 9.2.8 (SearchNP [CL24])**.** Let P be a search problem and R be the binary relation defining P. We say P can be solved by a nondeterministic polynomial-time algorithm if there is a nondeterministic Turing machine M such that for every input x,

- If x has a solution, then $M(x)$ has an accepting computation path, and every accepting path will output a valid solution y, i.e., $R(x, y)$ is true.
- If x has no solution, then $M(x)$ has no accepting computation path.

The class of search problems solvable by nondeterministic polynomial-time algorithms is defined as SearchNP.

Definition 9.2.9 (FNP [CL24])**.** The class of search problems defined by a polynomial-time relation, i.e., $R \in \mathsf{P}$ is defined as FNP.

While it is clear that $\mathsf{FNP} \subseteq \mathsf{SearchNP}$, the following example suggests that this inclusion is strict.

Proposition 9.2.10 ([CL24])**.** *If* $\mathsf{P} \neq \mathsf{NP}$*, then there is a total search problem in* $\mathsf{SearchNP} \backslash \mathsf{FNP}$.

For more knowledge about nondeterministic algorithms, readers are referred to [CL24, Section 2.4].

9.2.4 Proof Complexity

Recall that TAUT, the set of DNF tautologies, is the canonical coNP-complete problem. A *propositional proof system* is simply a nondeterministic algorithm for TAUT. More formally:

Definition 9.2.11 ([CR79])**.** An algorithm $\mathcal{P}(\varphi, \pi)$ is called a *propositional proof system* if it satisfies the following conditions:

- **(Completeness)** For every $\varphi \in \mathsf{TAUT}$, there exists a string $\pi \in \{0,1\}^*$ such that $\mathcal{P}(\varphi, \pi)$ accepts.

- **(Soundness)** For every $\varphi, \pi \in \{0,1\}^*$, if $\mathcal{P}(\varphi, \pi)$ accepts, then $\varphi \in \mathsf{TAUT}$.
- **(Efficiency)** $\mathcal{P}(\varphi, \pi)$ runs in deterministic $\mathrm{poly}(|\varphi| + |\pi|)$ time.

We say that $\mathcal{P}$ is a *non-uniform* propositional proof system if $\mathcal{P}$ is a polynomial-size circuit instead of a uniform algorithm (that is, $\mathcal{P}$ is equipped with non-uniform advice).

Definition 9.2.12 (Proof Complexity Generators [ABRW04, Kra04])**.** Let $s(n) < n$ be a function for seed length. A proof complexity generator is a map $C_n : \{0,1\}^s \to \{0,1\}^n$ computed by a family of polynomial-size circuits $\{C_n\}_n$. A generator is secure against a propositional proof system P if for every large enough n and every $y \in \{0,1\}^n$, P does not have a polynomial-size proof of the (properly encoded) statement

$$\forall x \in \{0,1\}^s, C_n(x) \neq y.$$

It is easy to see that the existence of proof complexity generators is closely related to the hardness of range avoidance. In fact, we have:

Theorem 9.2.13 (Informal version of [RSW22, Theorem 6.6])**.** *The range avoidance problem with suitable stretch is in* FNP *if and only if there exists a propositional proof system that breaks every proof complexity generator.*

Pseudo-surjectivity. In addition to the basic notion of hardness for proof complexity generators, several stronger notions have been proposed in the literature, including freeness [Kra01b], iterability and pseudo-surjectivity [Kra04], and $\bigvee$-hardness [Kra24]. Pseudo-surjectivity is the strongest hardness notion among them. In this chapter, we show that our proof complexity generators are pseudo-surjective in certain parameter regimes.

To motivate the definition of pseudo-surjectivity [Kra04, Definition 3.1], it is helpful to consider *Student-Teacher games* for solving Avoid. Let $G : \{0,1\}^n \to \{0,1\}^m$ be a circuit where $m > n$. A polynomial-time *Student* attempts to find a string $y \in \{0,1\}^m \setminus \mathrm{Range}(G)$ with the help of a *Teacher* who has unbounded computational power. The game proceeds in rounds. In each round i, the Student proposes a candidate string $y_i \in \{0,1\}^m$, and if $y_i \in \mathrm{Range}(G)$, the Teacher returns a preimage $q_i \in \{0,1\}^n$ such that $G(q_i) = y_i$. If the Student ever proposes a string outside the range of G, they win the game.

A Student who attempts to solve Avoid in k rounds can be represented as k circuits $B_1, B_2, \ldots, B_k$, where each B_i uses the Teacher's responses from previous rounds to generate the next query. Specifically, B_1 outputs a fixed string $y_1 \in \{0,1\}^m$, and each subsequent circuit B_i ($i > 1$) takes the previous responses $q_1, q_2, \ldots, q_{i-1} \in \{0,1\}^n$ as inputs and outputs $y_i \in \{0,1\}^m$. The game proceeds as follows:

- The Student proposes $y_1 := B_1 \in \{0,1\}^m$. If $y_1 \notin \mathrm{Range}(G)$, then the Student wins the game; otherwise, the Teacher returns some $q_1 \in \{0,1\}^n$ such that $G(q_1) = y_1$.
- The Student then proposes $y_2 := B_2(q_1) \in \{0,1\}^m$. If $y_2 \notin \mathrm{Range}(G)$ then the Student wins the game; otherwise, the Teacher returns $q_2 \in \{0,1\}^n$ such that $G(q_2) = y_2$.
- ...

- This continues until round k, where the Student proposes $y_k := B_k(q_1, \ldots, q_{k-1}) \in \{0,1\}^m$. If $y_k \notin \text{Range}(G)$, then the Student wins the game; otherwise, the Student loses the game.

To formally express whether the Student succeeds in the game, we define a formula stating that at least one of the Student's queries is outside the range of G. Let $B : \{0,1\}^{n'} \to \{0,1\}^m$ be a circuit, $z \in \{0,1\}^{n'}$ and $x \in \{0,1\}^n$ be disjoint variables, we define $\tau(G)_{B(z)}(x)$ to be the (properly encoded) statement that $B(z) \neq G(x)$. Then, using $\vec{q}_1, \ldots, \vec{q}_{k-1} \in \{0,1\}^n$ to represent the Teacher's responses, the Student wins if and only if

$$\bigvee_{i=1}^{k} \tau(G)_{B_i(q_1,\ldots,q_{i-1})}(q_i). \tag{9.1}$$

Roughly speaking, a generator G is *pseudo-surjective* for a proof system $\mathcal{P}$ if $\mathcal{P}$ cannot prove any Student wins the game, no matter how the Student is constructed. In other words, a generator G is pseudo-surjective for $\mathcal{P}$ if, for every sequence of Student circuits $(B_1, \ldots, B_k)$, the formula (9.1) is hard to prove in $\mathcal{P}$.

Note that pseudo-surjectivity is indeed a stronger notion than standard hardness for proof complexity generators. Indeed, for every $y \in \{0,1\}^m \setminus \text{Range}(G)$, the trivial one-round Student with $B_1 = y$ clearly wins the game—yet pseudo-surjectivity implies that this fact is hard to prove in $\mathcal{P}$.

We proceed to the formal definition. We also introduce the notion of *k-round* pseudo-surjectivity, where the unprovability of (9.1) only holds for k-round Students for some fixed $k = k(n)$.

Definition 9.2.14 (k-round pseudo-surjectivity [Kra04])**.** Let $\mathcal{P}$ be any proof system, $G : \{0,1\}^n \to \{0,1\}^m$ be a circuit where $m > n$, and s be a size parameter.

- We say that G is *s-pseudo-surjective* for $\mathcal{P}$ if for every k and every sequence of Student circuits $(B_1, B_2, \ldots, B_k)$, (9.1) requires $\mathcal{P}$-proof of size at least s.
- Fixing a parameter $k = k(n)$, we say that G is *k-round s-pseudo-surjective* for $\mathcal{P}$ if for every sequence of Student circuits $(B_1, B_2, \ldots, B_k)$, (9.1) requires $\mathcal{P}$-proof of size $\geq s$.

When $s = n^{\omega(1)}$, we omit the parameter s and simply say that G is (k-round) pseudo-surjective for $\mathcal{P}$.

9.2.5 Bounded Arithmetic

Roughly speaking, PV_1 is a theory of bounded arithmetic capturing "polynomial-time" reasoning. The language of PV_1, $\mathcal{L}(\mathsf{PV})$, contains a function symbol for every polynomial-time algorithm $f : \mathbb{N}^k \to \mathbb{N}$, defined using Cobham's characterization of polynomial-time functions [Cob64]. Although Cook's PV [Coo75] was originally defined as an equational theory (i.e., the only relation in PV is equality and there are no quantifiers), one can define a first-order theory PV_1 by adding suitable induction schemes [Coo75, KPT91]. In the literature, the notation PV is often used to refer to the set of polynomial-time computable functions as well. The precise definition of PV_1 is somewhat involved, and we refer the reader to the textbooks [Kra95, CN10, Kra19] and references [Coo75, Jer06, CLO24, Li25].

To capture reasoning in *randomised* polynomial time, Jeřábek [Jeř04, Jeř05, Jer07] defined a theory APC_1 by extending PV_1 with the *dual weak Pigeonhole Principle* for PV_1 functions (i.e., polynomial-time functions). Let $\mathsf{Eval}(\langle C\rangle, x) := C(x)$ be the circuit evaluation function. For a function $\ell(n) > n$, define $\mathrm{dwPHP}_\ell(\mathsf{Eval})$ to be the following sentence

$$\mathrm{dwPHP}_\ell(\mathsf{Eval}) :=$$
$$\forall n \in \mathsf{Log}\ \forall \text{circuit } C : \{0,1\}^n \to \{0,1\}^{\ell(n)}\ \exists y \in \{0,1\}^{\ell(n)}\ \forall x \in \{0,1\}^n\ [\mathsf{Eval}(C,x) \neq y].$$

Here, "$n \in \mathsf{Log}$" is the standard notation in bounded arithmetic, which means that n is the bit-length of some object; this notation allows us to reason about objects of size $\mathrm{poly}(n)$ instead of merely size $\mathrm{polylog}(n)$. The above sentence can be interpreted as the totality of Avoid: every input $C : \{0,1\}^n \to \{0,1\}^{\ell(n)}$ has at least one solution y.

For this chapter, it suffices to think of $\ell(n)$ as a large polynomial in n; in fact, under suitable hardness assumptions, we will be able to show that PV_1 cannot prove dwPHP_ℓ for every polynomial $\ell(n)$. This suffices to separate APC_1 from PV_1.

KPT witnessing and Student-Teacher games. The only property of bounded arithmetic theories that we need in this chapter is the *KPT witnessing theorem*:

Theorem 9.2.15 (KPT Witnessing Theorem for PV_1 [KPT91])**.** *For every quantifier-free formula $\varphi(\vec{x}, y, z)$ in the language $\mathcal{L}(\mathsf{PV})$, if $\mathsf{PV}_1 \vdash \forall \vec{x}\ \exists y\ \forall z\ \varphi(\vec{x}, y, z)$, then there is a $k \in \mathbb{N}$ and $\mathcal{L}(\mathsf{PV})$-terms $t_1, t_2, \ldots, t_k$ such that*

$$\mathsf{PV}_1 \vdash \forall \vec{x}\ \forall z_1\ \forall z_2 \ldots \forall z_k \bigvee_{i=1}^{k} \varphi(\vec{x}, t_i(\vec{x}, z_1, \ldots, z_{i-1}), z_i). \tag{9.2}$$

In particular, Theorem 9.2.15 implies that if $\mathsf{PV}_1 \vdash \mathrm{dwPHP}_\ell(\mathsf{Eval})$, then there exists a constant k and a polynomial-time Student that wins the Student-Teacher game for the Range Avoidance problem. (Note that here the Student is computed by a *uniform* algorithm that gets $(1^n, C)$ as inputs, as opposed to a family of non-uniform circuits in Section 9.2.4). To see this, let $\varphi((1^n, C), y, x) = 1$ iff $\mathsf{Eval}(C, x) \neq y$ and apply Theorem 9.2.15. We obtain a constant $k \in \mathbb{N}$ and $\mathcal{L}(\mathsf{PV})$-terms $t_1, t_2, \ldots, t_k$ such that:

$$\mathsf{PV}_1 \vdash \forall n \in \mathsf{Log}\ \forall C\ \forall z_1\ \forall z_2 \ldots \forall z_k \bigvee_{i=1}^{k} \mathsf{Eval}(C, z_i) \neq t_i(C, z_1, \ldots, z_{i-1}). \tag{9.3}$$

This implies that the following Student wins the Student-Teacher game in k rounds:

1. The Student and the Teacher are given a circuit $C : \{0,1\}^n \to \{0,1\}^{\ell(n)}$ as input.
2. In the first round, the Student proposes $y_1 := t_1(C)$ as a candidate non-output. If y_1 is correct (i.e., $\forall z_1\ \varphi(C, y_1, z_1)$ is true), then the Student wins the game. Otherwise, the Teacher provides a counterexample z_1 such that $\mathsf{Eval}(C, z_1) = y_1$, i.e., a preimage $z_1 \in C^{-1}(y_1)$.

3. Then, the Student proposes a new candidate $y_2 := t_2(C, z_1)$ based on the counterexample given in the first round. If y_2 is a correct non-output, then the Student wins the game. Otherwise, the Teacher again provides a counterexample z_2 such that $\mathsf{Eval}(C, z_2) = y_2$, i.e., a preimage $z_2 \in C^{-1}(y_2)$.

4. The game proceeds until the Student provides a correct witness y.

9.2.6 Extractors

Definition 9.2.16 (k-Source)**.** A random variable X is a *k-source* if for every $x \in \mathrm{Supp}(X)$, $\Pr[X = x] \leq 2^{-k}$.

Definition 9.2.17 (Strong Seeded Extractors)**.** A polynomial-time computable function $\mathsf{Ext} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ is a *(k, ε)-strong seeded extractor* if for every k-source $\mathcal{X}$ over $\{0,1\}^n$, the statistical distance of $(\mathcal{U}_d, \mathsf{Ext}(X, \mathcal{U}_d))$ and $(\mathcal{U}_d, \mathcal{U}_m)$ is at most ε.

Below is the only property of strong seeded extractors that we will use:

Fact 9.2.18. *Suppose $\mathsf{Ext} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ is a (k, ε)-strong seeded extractor. Then for every (possibly unbounded) adversary $\mathcal{A} : \{0,1\}^{d+m} \to \{0,1\}$, the number of strings $x \in \{0,1\}^n$ such that*

$$\Pr_{r \leftarrow \{0,1\}^d}[\mathcal{A}(r, \mathsf{Ext}(x, r)) = 1] < \Pr_{r \leftarrow \{0,1\}^d, z \leftarrow \{0,1\}^m}[\mathcal{A}(r, z) = 1] - \varepsilon \tag{9.4}$$

is at most 2^k.

Proof Sketch. Fix an adversary $\mathcal{A}$, let $\mathcal{X}$ be the set of strings $x \in \{0,1\}^n$ such that (9.4) holds. (We abuse notation and also use $\mathcal{X}$ to denote the uniform distribution over itself.) Note that $\mathcal{A}$ distinguishes $\mathsf{Ext}(\mathcal{X}, r)$ from the uniform distribution with advantage ε. If $|\mathcal{X}| \geq 2^k$, then the min-entropy of $\mathcal{X}$ is at least k, contradicting the extractor properties of Ext. Hence $|\mathcal{X}| < 2^k$. □

We require extractors with exponentially small ε, which can be constructed from any family of pairwise independent hash functions.

Theorem 9.2.19 (Leftover Hash Lemma [ILL89])**.** *Let $h : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ be a family of pairwise independent hash functions, where the first component (length n) is its input and the second component (length d) is its key. Then for every k, ε such that $m = k - 2\log(1/\varepsilon)$, h is a (k, ε)-strong seeded extractor.*

In particular, if $n \geq m$ and $d \geq 2n$, there exists a family of pairwise independent hash functions h that is $\mathbb{F}_2$-linear.[9] If we set $n \geq 3m + 3$, $d \geq 2n$, $k := n - 1$, $\varepsilon := 2^{-m-1}$, then h is an $(n-1, \varepsilon)$-strong seeded extractor.

9.3 Hardness of Range Avoidance

Theorem 9.3.1. *Assume that for some $m > n$, there exists a demi-bits generator $G : \{0,1\}^n \to \{0,1\}^N$ and $\mathsf{Ext} : \{0,1\}^N \times \{0,1\}^d \to \{0,1\}^m$ is an $(N - 1, 2^{-m-1})$-strong seeded extractor. ($N, d \leq \mathrm{poly}(m)$.) Then* Avoid *for polynomial-size circuits of stretch $n \to m$ is not in* SearchNP.

[9] That is, for each fixed $r \in \{0,1\}^d$, the function $h(-, r)$ is an $\mathbb{F}_2$-linear function over its inputs.

Proof. Let $r \in \{0,1\}^d$, define the circuit $C_r : \{0,1\}^n \to \{0,1\}^m$ with r hardwired:

$$C_r(s) = \mathsf{Ext}(G(s), r).$$

Assume towards contradiction that there is a nondeterministic polynomial-time algorithm $\mathcal{A}$ solving Avoid. We construct the following nondeterministic adversary $\mathcal{B}(y)$ that breaks the demi-bits generator G. Given an input $y \in \{0,1\}^N$, the adversary $\mathcal{B}$ accepts y if and only if there exists $r \in \{0,1\}^d$ such that some nondeterministic branch of $\mathcal{A}(C_r)$ outputs $\mathsf{Ext}(y, r)$.

It is easy to see that $\mathcal{B}$ rejects every string $y \in \mathrm{Range}(G)$. To see this, suppose that $y = G(s)$ for some $s \in \{0,1\}^n$. Then

$$C_r(s) = \mathsf{Ext}(G(s), r) = \mathsf{Ext}(y, r),$$

hence $\mathcal{A}(C_r)$ will never output $\mathsf{Ext}(y, r)$.

It remains to show that $\mathcal{B}$ accepts at least $1/2$ fraction of strings $y \in \{0,1\}^N$. For $r \in \{0,1\}^d, z \in \{0,1\}^m$, let $\mathcal{A}'(r, z)$ be the adversary that outputs 1 if there is a nondeterministic branch of $\mathcal{A}(C_r)$ that outputs z, and outputs 0 otherwise. Since Ext is an $(N-1, 2^{-m-1})$-strong seeded extractor, the following is true for at least $1/2$ fraction of $y \in \{0,1\}^N$:

$$\Pr_{r \leftarrow \{0,1\}^d}[\mathcal{A}'(r, \mathsf{Ext}(y, r)) = 1] \geq \Pr_{\substack{r \leftarrow \{0,1\}^d \\ z \leftarrow \{0,1\}^m}}[\mathcal{A}'(r, z) = 1] - 2^{-m-1} \geq 2^{-m-1}.$$

For such y, there exists $r^\star \in \{0,1\}^d$ and a nondeterministic branch of $\mathcal{A}(C_{r^\star})$ that outputs $\mathsf{Ext}(y, r^\star)$.

It follows that $\mathcal{B}$ rejects every string in $\mathrm{Range}(G)$ but accepts at least $1/2$ fraction of strings $y \in \{0,1\}^N$. This contradicts the security of G. □

Theorem 9.1.2. *Let $G : \{0,1\}^n \to \{0,1\}^N$ be a demi-bits generator, $\mathcal{H} = \{h : \{0,1\}^N \to \{0,1\}^m\}$ be a family of pairwise independent hash functions, and $\mathcal{A}$ be a nondeterministic polynomial-time algorithm. If $N > 10m > n$, then there exists $h \in \mathcal{H}$ such that $\mathcal{A}$ fails to solve the range avoidance problem on the input $h \circ G$.*

Proof. We construct an extractor $\mathsf{Ext}(y, h) := h(y)$ $(y \in \{0,1\}^N, h \in \mathcal{H})$. According to Theorem 9.2.19, $\mathsf{Ext}\colon \{0,1\}^N \times \mathcal{H} \to \{0,1\}^m$ is an $(N-1, 2^{-m-1})$-strong seeded extractor. Therefore, by Theorem 9.3.1, there exists $h \in \mathcal{H}$ such that $\mathcal{A}$ fails to solve Avoid on $\mathsf{Ext}(G(\cdot), h)$, i.e., $h \circ G$. □

Corollary 9.3.2. *Under Assumption 9.2.3, there are constants $\varepsilon > 0$ and $d \geq 2$ such that* Avoid *for $\mathsf{XOR} \circ \mathsf{AND}_d$ circuits (i.e., degree-d polynomials over $\mathbb{F}_2$) of stretch $n \mapsto n^{1+\varepsilon}$ is not in* SearchNP.

Proof. Assumption 9.2.3 implies a demi-bits generator $G : \{0,1\}^n \to \{0,1\}^{n^{1+\delta}}$ and each output bit of G can be computed by a degree-d polynomial over $\mathbb{F}_2$, where $\delta > 0$ and $d \geq 2$ are constants. Let $\varepsilon := \delta/2$, $\mathsf{Ext} : \{0,1\}^{n^{1+\delta}} \times \{0,1\}^{2n^{1+\delta}} \to \{0,1\}^{n^{1+\varepsilon}}$ be a $(n^{1+\delta} - 1, 2^{-n^{1+\varepsilon}-1})$-strong linear seeded extractor guaranteed by Theorem 9.2.19. Then, for every nondeterministic adversary $\mathcal{A}$,

there exists $r \in \{0,1\}^{2n^{1+\delta}}$ such that $\mathcal{A}$ fails to solve AVOID on the instance

$$C_r(s) := \mathsf{Ext}(G(s), r).$$

Since Ext is multi-linear and G is a degree-d polynomial over $\mathbb{F}_2$, C_r is an $\mathsf{XOR} \circ \mathsf{AND}_d$ circuit. □

9.3.1 From Demi-Bits Generators to Proof Complexity Generators

Let $\mathcal{P}$ be a proof system and $G : \{0,1\}^n \to \{0,1\}^\ell$ be a function computable in polynomial size where $\ell > n$. (We allow G to take non-uniform advice.) Let $b \in \{0,1\}^\ell$, denote as $\tau_b(G)$ the propositional formula encoding that b is not in the range of G. We say G is a:

- *demi-bits generator* against $\mathcal{P}$, if for at least a $1/3$ fraction of $b \in \{0,1\}^\ell$, $\mathcal{P}$ does not have polynomial-size proof of $\tau_b(G)$; and G is a
- *proof complexity generator* against $\mathcal{P}$, if for *every* $b \in \{0,1\}^\ell$, $\mathcal{P}$ does not have polynomial-size proof of $\tau_b(G)$.

The precise definition of $\tau_b(G)$ as a 3-CNF is as follows. The variables of $\tau_b(G)$ consist of $x \in \{0,1\}^n$ and $\mathsf{hist} \in \{0,1\}^s$, where s is the number of internal gates in G (including the output gates but not including the input gates). The intended meaning is that $G(x) = b$ and hist represents the values of internal gates of G during the computation of $G(x)$. Each gate in G corresponds to a bit v_g; if g is an input (internal) gate then v_g refers to some x_i (hist_i). For each internal gate $g \in G$ labeled by an operation $\circ_g$ (such as $\wedge$, $\vee$, or $\oplus$) and two children gates g_l, g_r, we have a constraint

$$v_g = v_{g_l} \circ_g v_{g_r}$$

in $\tau_b(G)$. Similarly, for each $i \in [\ell]$ representing an output gate g_i, we have a constraint

$$v_{g_i} = b_i$$

in $\tau_b(G)$. Note that since every constraint only depends on at most 3 variables, it can be written as a 3-CNF of size at most $2^3 = 8$, and we can add every clause in this 3-CNF into $\tau_b(G)$. We assume that the $\oplus$ gate of fan-in 2 is included in our basis (looking ahead, it will be used to implement the extractor). The 3-CNF $\tau_b(G)$ is simply the union of (3-CNFs generated from) these constraints over every internal and output gate $g \in G$.

Now we define the notion of *simple parity reductions* between two CNFs. This is a technical notion that we need in Claim 9.3.4.

Definition 9.3.3. Let $F(x)$ and $G(y)$ be CNF formulas over variables $x = (x_1, \ldots, x_n)$ and $y = (y_1, \ldots, y_m)$. We say that there is a *simple parity* reduction from F to G, denoted as $F \leq^{\oplus} G$, if:

- **Variables.** The reduction is computed by a GF(2)-linear mapping $\mathsf{redu} : \{0,1\}^n \to \{0,1\}^m$ (that is, every output bit of redu is the XOR of a subset of its input bits).
- **Axioms.** For any clause $g \in G$, one of the following happens:
  - $g \circ \mathsf{redu} \equiv \mathsf{True}$;

– $g \circ \mathsf{redu}$ is equal to some axiom in F; or
– g is a width-1 clause (i.e., one that consists of a *single* literal) and $g \circ \mathsf{redu}$ is the XOR of a subset of axioms in F (in which case these axioms in F are also width-1 clauses).

We say a proof system $\mathcal{P}$ is *closed under simple parity reductions* if there is a polynomial p such that the following holds. For every CNF F and G, if there is a simple parity reduction from F to G and there is a length-ℓ $\mathcal{P}$-proof of G, then there is a length-$p(\ell)$ $\mathcal{P}$-proof of F.

We note that this notion is weaker than that of (degree-1) algebraic reductions in [BGIP01, dRGN+21]. It follows from [dRGN+21, Lemma 8.3] that many algebraic proof systems (such as Nullstellensatz and Polynomial Calculus) over GF(2) are closed under simple parity reductions when the complexity measure is degree. While we do not know if $\mathsf{Res}[\oplus]$ (resolution over linear equations modulo 2 [IS20]) is closed under low-degree algebraic reductions, it is straightforward to prove that $\mathsf{Res}[\oplus]$ is closed under simple parity reductions (see Theorem 9.3.6).

Recall that $G : \{0,1\}^n \to \{0,1\}^N$ is a purported demi-bits generator, $\mathsf{Ext} : \{0,1\}^N \times \{0,1\}^d \to \{0,1\}^m$ is an extractor, and for a fixed $r \in \{0,1\}^d$ we define the circuit $C_r : \{0,1\}^n \to \{0,1\}^m$ as

$$C_r(s) := \mathsf{Ext}(G(s), r).$$

We say that Ext is *linear* if for every fixed randomness r, the function $\mathsf{Ext}(\cdot, r) : \mathrm{GF}(2)^N \to \mathrm{GF}(2)^m$ is GF(2)-linear. For every r we fix a circuit Ext_r for computing $\mathsf{Ext}(\cdot, r)$ using $\oplus$ gates of fan-in 2 only.

Claim 9.3.4. *Suppose that* Ext *is a linear extractor. For every* $y \in \{0,1\}^N$ *and* $r \in \{0,1\}^d$*, there is a simple parity reduction from* $\tau_y(G)$ *to* $\tau_z(C_r)$*, where* $z := \mathsf{Ext}(y, r)$*.*

First, as a sanity check, we show that $\tau_y(G)$ follows from $\tau_z(C_r)$ logically: Suppose that $\tau_y(G)$ is false and that $y = G(s)$ for some $s \in \{0,1\}^n$, then

$$C_r(s) = \mathsf{Ext}(G(s), r) = \mathsf{Ext}(y, r) = z,$$

meaning that $\tau_z(C_r)$ is also false. Now we show that if Ext is a linear extractor, then the above deduction is actually a simple parity reduction under our formalisation of $\tau_b(G)$:

Proof of Claim 9.3.4. Recall that the variables of $\tau_y(G)$ consist of $s \in \{0,1\}^n$ and $\mathsf{hist}_G \in \{0,1\}^{|G|}$, where $|G|$ denotes the number of internal gates in G. Also, recall the variables of $\tau_z(C_r)$ consist of $s \in \{0,1\}^n$ and $\mathsf{hist}_{C_r} \in \{0,1\}^{|C_r|}$. Since $C_r(s) = \mathsf{Ext}(G(s), r)$, hist_{C_r} consists of hist_G as well as the internal gates of $\mathsf{Ext}(\cdot, r)$. Since Ext is linear, each internal gate in $\mathsf{Ext}(\cdot, r)$ is an XOR of variables in hist_G. Therefore, one can compute a GF(2)-linear map $\mathsf{redu} : \{0,1\}^{n+|G|} \to \{0,1\}^{n+|C_r|}$ that maps (s, hist_G) to (s, hist_{C_r}).

Now we show that for every clause $c \in \tau_z(C_r)$, one of the three cases in Definition 9.3.3 happens. Note that c comes from an internal gate or an output gate of C_r.

- If c comes from an internal gate of G, then $c \circ \mathsf{redu}$ (which is equal to c itself) is an axiom in $\tau_y(G)$.
- If c comes from an internal gate in $\mathsf{Ext}(\cdot, r)$, then $c \circ \mathsf{redu} \equiv \mathsf{True}$ by the definition of redu.

- The only remaining case is that c comes from an output gate. Suppose this is the i-th output gate of C_r (where $i \in [m]$), and let v'_i denote the variable (of $\tau_z(C_r)$) representing the i-th output of C_r. Note that c is a width-1 axiom stating that $v'_i = z_i$.

  Let $S_i \subseteq [N]$ be such that $\mathsf{Ext}(y,r)_i = \bigoplus_{j \in S_i} y_j$. Then redu maps v'_i to $\bigoplus_{j \in S_i} v^G_j$, where v^G_j is the variable in $\tau_y(G)$ that represents the j-th output gate of G. We also have that $z_i = \bigoplus_{j \in S_i} y_j$. Hence $c \circ \mathsf{redu}$ is the XOR of the axioms $v^G_j = y_j$ over all $j \in S_i$. Since each $v^G_j = y_j$ is an axiom in $\tau_y(G)$, this concludes the proof. □

Theorem 9.3.5. *Let $\mathcal{P}$ be a proof system closed under parity reductions. Let $G : \{0,1\}^n \to \{0,1\}^N$ be a demi-bits generator secure against $\mathcal{P}$, and $\mathsf{Ext} : \{0,1\}^N \times \{0,1\}^d \to \{0,1\}^m$ be an $(N-2, 2^{-m-1})$-strong linear seeded extractor. Then there is a non-uniform proof complexity generator secure against $\mathcal{P}$.*

Proof. Suppose for contradiction that for every $r \in \{0,1\}^d$, there exists a string $z(r) \in \{0,1\}^m$ such that $\mathcal{P}$ admits a length-ℓ proof of $\tau_{z(r)}(C_r)$, where $\ell \leq \mathrm{poly}(|G|)$. For $r \in \{0,1\}^d$ and $z \in \{0,1\}^m$, let $\mathcal{A}'(r,z)$ be the adversary that outputs 1 if $\mathcal{P}$ admits a length-ℓ proof of $\tau_z(C_r)$ and outputs 0 otherwise. Since for every r, $\mathcal{A}'(r, z(r)) = 1$, we have

$$\Pr_{\substack{r \leftarrow \{0,1\}^d \\ z \leftarrow \{0,1\}^m}} [\mathcal{A}'(r,z) = 1] \geq 2^{-m}.$$

Since Ext is a $(n-2, 2^{-m-1})$-strong extractor, for at least a $3/4$ fraction of $y \in \{0,1\}^N$, we have

$$\Pr_{r \leftarrow \{0,1\}^d} [\mathcal{A}'(r, \mathsf{Ext}(y,r)) = 1] \geq \Pr_{\substack{r \leftarrow \{0,1\}^d \\ z \leftarrow \{0,1\}^m}} [\mathcal{A}'(r,z) = 1] - 2^{-m-1} > 0.$$

Hence, for such $y \in \{0,1\}^N$, there exists some $r := r(y)$ such that $\mathcal{P}$ admits a length-ℓ proof of $\tau_z(C_r)$ where $z := \mathsf{Ext}(y,r)$. Since there is a parity reduction from $\tau_y(G)$ to $\tau_z(C_r)$ and $\mathcal{P}$ is closed under parity reductions, it follows that $\mathcal{P}$ admits a length-$\mathrm{poly}(\ell)$ proof of $\tau_y(G)$ as well, contradicting the security of G as a demi-bits generator against $\mathcal{P}$. □

Although super-polynomial lower bounds for $\mathsf{Res}[\oplus]$ remain open, it seems conceivable that we will eventually prove such lower bounds sooner or later. Our results suggest a potential approach for designing proof complexity generators against $\mathsf{Res}[\oplus]$: it suffices to design a *demi-bits generator* against $\mathsf{Res}[\oplus]$ (which might be an easier task) and then apply Theorem 9.3.5.

We end this section by proving that $\mathsf{Res}[\oplus]$ is closed under simple parity reductions:

Theorem 9.3.6. *$\mathsf{Res}[\oplus]$ is closed under simple parity reductions. That is, let $F(x_{1 \sim n}) = f_1 \wedge f_2 \wedge \cdots \wedge f_m$ and $G(y_{1 \sim n'}) = g_1 \wedge g_2 \wedge \cdots \wedge g_{m'}$ be CNF formulas such that $F \leq^{\oplus} G$. If there exists $\mathsf{Res}[\oplus]$ refutation of G in s steps, then there exists a $\mathsf{Res}[\oplus]$ refutation of F in $2nm' + s$ steps.*

Proof. We assume familiarity with $\mathsf{Res}[\oplus]$ (the definition can be found in [IS20]).

Let $C_1, C_2, \ldots, C_s$ be a $\mathsf{Res}[\oplus]$ refutation of G where each C_i is a disjunction of linear equations modulo 2, $C_i = g_i$ for every $1 \leq i \leq m$, and $C_s = \bot$. Let $\mathsf{redu} : \{0,1\}^n \to \{0,1\}^m$ be the simple parity reduction from F to G. Define $C'_i = C_i \circ \mathsf{redu}$, then C'_i is still a disjunction of

linear equations modulo 2. It is easy to see that $C'_1, C'_2, \ldots, C'_s$ is still a valid $\mathsf{Res}[\oplus]$ derivation, and that $C'_s = \bot$. Hence there is an s-step $\mathsf{Res}[\oplus]$ refutation from the axioms $\{g_i \circ \mathsf{redu}\}_{1 \le i \le m}$.

It suffices to show that each $g_i \circ \mathsf{redu}$ can be proved from the axioms of F. This is easy to see when $g_i \circ \mathsf{redu} \equiv \mathsf{True}$ or $g_i \circ \mathsf{redu}$ is equal to some f_i, hence we only need to consider the third case in Definition 9.3.3 where $g_i \circ \mathsf{redu}$ is the XOR of some axioms in f_i. Note that one can derive $(a \oplus b = 0)$ from $(a = 0)$ and $(b = 0)$ in 2 steps[10], hence $g_i \circ \mathsf{redu}$ can be derived from F in $2n$ steps. Since there are at most m' linear clauses of the form $g_i \circ \mathsf{redu}$ that need to be derived, the total number of steps is at most $2nm' + s$. □

9.4 Lower Bounds for Student-Teacher Games

In this section, we show that the Range Avoidance problem is hard for Student-Teacher games. In Section 9.4.1 we prove lower bounds against uniform, polynomial-time Students, which implies a conditional separation between bounded arithmetic theories PV_1 and APC_1. In Section 9.4.2, we show that demi-bits generators can be transformed into proof complexity generators that are pseudo-surjective.

9.4.1 Separating PV_1 from APC_1

As discussed in Section 9.2.5, to separate PV_1 from APC_1, it suffices to show that there is no polynomial-time Student that wins the Student-Teacher game in $O(1)$ rounds. In fact, we will show something stronger: Let $k = k(n)$ be a parameter, assuming the existence of demi-bits generators secure against $\mathsf{AM}/_{O(\log k)}$, there is no polynomial-time Student that wins the Student-Teacher game in $k(n)$ rounds.

Theorem 9.4.1. *Let m, n, k be parameters such that $m > n$. Assume there exists a demi-bits generator $G : \{0,1\}^n \to \{0,1\}^N$ secure against $\mathsf{AM}/_{O(\log k)}$. Let $\mathsf{Ext} : \{0,1\}^N \times \{0,1\}^d \to \{0,1\}^m$ be an $(N-1, 2^{-10km})$-strong extractor. Then for every deterministic polynomial-time Student A, there is a string $r \in \{0,1\}^d$ and a Teacher such that A fails to solve* Avoid *on C_r in k rounds, where $C_r : \{0,1\}^n \to \{0,1\}^m$ is the circuit*

$$C_r(s) := \mathsf{Ext}(G(s), r).$$

Proof. Let A denote the Student algorithm where $A(i, C, z_1, \ldots, z_{i-1})$ outputs the i-th candidate solution. For strings $s_1, \ldots, s_j \in \{0,1\}^n$ (where $j \le k$), we say that $(s_1, \ldots, s_j)$ is a *valid trace* for A on the input C_r if all of the following are true:

- $C_r(s_1) = A(1, C_r)$ (that is, s_1 is a valid counterexample for $A(1, -)$);
- $C_r(s_2) = A(2, C_r, s_1)$ (that is, s_2 is a valid counterexample for $A(2, -)$);
- ...
- and $C_r(s_j) = A(j, C_r, s_1, s_2, \ldots, s_{j-1})$ (that is, s_j is a valid counterexample for $A(j, -)$).

We prove the following stronger claim that implies Theorem 9.4.1:

[10] First weaken $(b = 0)$ to derive $(a = 1 \vee a \oplus b = 0)$, then resolve $(a = 0)$ and $(a = 1 \vee a \oplus b = 0)$ to derive $(a \oplus b = 0)$.

Claim 9.4.2. *For every $j \le k$, there exist $s_1, s_2, \ldots, s_j \in \{0,1\}^n$ such that*

$$\Pr_{r \leftarrow \{0,1\}^d}[(s_1, \ldots, s_j) \text{ is a valid trace for } A \text{ on the input } C_r] \ge 2^{-2jm}.$$

Clearly, Claim 9.4.2 implies Theorem 9.4.1 by setting $j := k$ and noticing that $2^{-2jm} > 0$.

We prove Claim 9.4.2 by induction on j. The base case $j = 0$ is trivially true. Now we assume the claim is true for $j - 1$, which gives strings $s_1, \ldots, s_{j-1}$ such that

$$\Pr_{r \leftarrow \{0,1\}^d}[(s_1, \ldots, s_{j-1}) \text{ is a valid trace for } A \text{ on the input } C_r] \ge 2^{-2(j-1)m}.$$

Consider the following $\mathsf{AM}/_{O(\log k)}$ protocol that attempts to break the demi-bits generator G. This protocol has the index j hardwired as advice but is otherwise uniform.

Algorithm 9.4.1: The $\mathsf{AM}/_{O(\log k)}$ protocol $\mathcal{P}$ breaking demi-bits generator G

Input: A string $y \in \{0,1\}^N$.

1 Prover sends $s_1, \ldots, s_{j-1}$;

2 Prover and Verifier run the Goldwasser–Sipser protocol to estimate

$$p := \Pr_{r \leftarrow \{0,1\}^d}\left[\begin{array}{c}(s_1, \ldots, s_{j-1}) \text{ is a valid trace for } A \text{ on the input } C_r \\ \text{and } \mathsf{Ext}(y, r) = A(j, C_r, s_1, \ldots, s_{j-1}).\end{array}\right];$$

3 If $p \ge 2^{-(2j-1)m-1}$ then Verifier accepts; if $p \le 2^{-(2j-1)m-2}$ then Verifier rejects;

Completeness of $\mathcal{P}$. We show that for $\ge 1/2$ fraction of y, there is a Prover such that the Verifier accepts w.p. $\ge 2/3$ in $\mathcal{P}$. In the first round, the honest Prover sends $(s_1, \ldots, s_{j-1})$ as guaranteed by the induction hypothesis. Recall that this means

$$\Pr_{r \leftarrow \{0,1\}^d}[(s_1, \ldots, s_{j-1}) \text{ is a valid trace for } A \text{ on the input } C_r] \ge 2^{-2(j-1)m}.$$

Let $\mathsf{Test}(r, z) = 1$ if $(s_1, \ldots, s_{j-1})$ is a valid trace for A on the input C_r and $z = A(j, C_r, s_1, \ldots, s_{j-1})$, and $\mathsf{Test}(r, z) = 0$ otherwise. Clearly, we have

$$\Pr_{r \leftarrow \{0,1\}^d, z \leftarrow \{0,1\}^m}[\mathsf{Test}(r, z) = 1] \ge 2^{-2(j-1)m}/2^m = 2^{-(2j-1)m}.$$

Since Ext is an $(N - 1, 2^{-10km})$-strong extractor, for $\ge 1/2$ fraction of y's, it holds that

$$\Pr_{r \leftarrow \{0,1\}^d}[\mathsf{Test}(r, \mathsf{Ext}(y, r)) = 1] \ge \Pr_{r \leftarrow \{0,1\}^d, z \leftarrow \{0,1\}^m}[\mathsf{Test}(r, z) = 1] - 2^{-10km} \ge 2^{-(2j-1)m-1}.$$

It follows that there is a Prover for the Goldwasser–Sipser protocol in Line 2 of Algorithm 9.4.1 such that the verifier accepts with probability at least 2/3.

Employing the lack of soundness. Since G is a demi-bits generator that is secure against $\mathsf{AM}/_{O(\log k)}$ adversaries, $\mathcal{P}$ does not have the soundness for all sufficiently large n. In other words, there is a Prover* that makes the Verifier accepts some $y \in \text{Range}(G)$ with probability $> 1/3$.

Fix such a string y, let s_j be any n-bit string such that $G(s_j) = y$, and let $s_1, \ldots, s_{j-1} \in \{0,1\}^n$ be the message sent in Line 1 (of Algorithm 9.4.1) by Prover^* on the input y.

Since $\mathsf{Verifier}$ accepts with probability $> 1/3$, by the soundness of the Goldwasser–Sipser Protocol (Lemma 9.2.4), we have that

$$\Pr_{r \leftarrow \{0,1\}^d} \left[\begin{array}{c} (s_1, \ldots, s_{j-1}) \text{ is a valid trace for } A \text{ on the input } C_r \\ \text{and } \mathsf{Ext}(y, r) = A(j, C_r, s_1, \ldots, s_{j-1}). \end{array} \right] \geq 2^{-j(2m+1)-2}.$$

Note that $\mathsf{Ext}(y, r) = C_r(s_j)$, hence the above condition inside $\Pr_{r \leftarrow \{0,1\}^d}[\cdot]$ means exactly that $(s_1, \ldots, s_j)$ is a valid trace for $A(C_r)$. This implies Claim 9.4.2 for j. □

We remark that the parameters we obtained in Theorem 9.4.1 are (almost) tight in the following sense. Theorem 9.4.1 showed that (under plausible assumptions) for every parameter $k \leq \text{poly}(n)$, there is no deterministic polynomial-time Student that wins the Student-Teacher game for AVOID in k rounds, when given an AVOID instance of size $s = \text{poly}(k, n) > k$. On the other hand, under plausible derandomisation assumptions, for every size parameter s, there exists a deterministic polynomial-time Student that wins the game on size-s circuits within $k = \text{poly}(s, n) > s$ rounds [ILW23, Appendix A].

Finally, setting $k := O(1)$ in Theorem 9.4.1, we obtain the following separation:

Theorem 9.1.5. *Assume there exists a demi-bits generator $G : \{0,1\}^n \to \{0,1\}^{\omega(n)}$ secure against $\mathsf{AM}/_{O(1)}$. Then, the Dual Weak Pigeonhole Principle for polynomial-time functions* (dwPHP(PV)) *is not provable in* PV. *(In particular, APC_1 is a strict extension of PV_1.)*

9.4.2 From Demi-Bits to Pseudo-Surjectivity

Theorem 9.4.3. *Let $G : \{0,1\}^n \to \{0,1\}^N$ be a demi-bits generator secure against $\mathsf{NP}/_{\text{poly}}$, $k \in \mathbb{N}$, and $\mathsf{Ext} : \{0,1\}^N \times \{0,1\}^d \to \{0,1\}^m$ be an $(N-1, \varepsilon)$-strong linear extractor for $\varepsilon := 2^{-10km}$. $(k, d, N \leq \text{poly}(m).)$ Then for every non-uniform propositional proof system $\mathcal{P}$, there is a string $r \in \{0,1\}^d$ such that the circuit $C_r : \{0,1\}^n \to \{0,1\}^m$ defined as*

$$C_r(s) := \mathsf{Ext}(G(s), r)$$

is a non-uniform k-round pseudo-surjective proof complexity generator secure for $\mathcal{P}$.

Proof. Suppose, for contradiction, that such an $r \in \{0,1\}^d$ does not exist. Then, for any $r \in \{0,1\}^d$, there exist student circuits

$$B^{(r)} = \left\{ B_i^{(r)} : \{0,1\}^{n(i-1)} \to \{0,1\}^m \right\}_{i \in [k+1]}$$

such that $\mathcal{P}$ admits a proof of

$$\bigvee_{i=1}^{k+1} \tau(C_r)_{B_i(q_1, q_2, \ldots, q_{i-1})}(q_i).$$

(i.e., $\mathcal{P}$ can prove that $B^{(r)}$ wins the Student-Teacher game on C_r.)

Now we attempt to break the demi-bits generator G. For $j = 0, 1, \ldots, k+1$, define

$$\Phi_j := \max_{(s_1,s_2,\ldots,s_j)\in\{0,1\}^{nj}} \Pr_{r\leftarrow\{0,1\}^m} \begin{bmatrix} \exists \text{ Student } B \text{ such that:} \\ (1)\ \mathcal{P} \text{ proves that } B \text{ wins the Student-Teacher game on } C_r; \\ (2)\ B_i(s_1,\ldots,s_{i-1}) = C_r(s_i) \text{ for all } i \in [j]. \end{bmatrix}.$$

(Item (2) says that the history of the Student-Teacher game in the first i rounds is exactly $s_1, \ldots, s_j$.) We make the following claims on the values of Φ_0 and Φ_{k+1}:

- $\Phi_0 = 1$: When $j = 0$, item (2) obviously holds, and item (1) holds by our assumption that C_r is not a pseudo-surjective proof complexity generator;
- $\Phi_{k+1} = 0$: When $j = k+1$, for any r, B items (1) and (2) cannot hold simultaneously. This is because (2) implies that B loses the Student-Teacher game, which contradicts (1).

Simple calculations show that there exists $j \in \{0, 1, \ldots, k\}$ such that

$$\Phi_j \cdot 2^{-m} - \varepsilon > 2 \cdot \Phi_{j+1}.$$

We use such j to break the demi-bits generator G. Let $(s_1^*, \ldots, s_j^*)$ be the tuple such that the maximum is achieved in the definition of Φ_j, i.e.,

$$\Phi_j = \Pr_{r\leftarrow\{0,1\}^m} \begin{bmatrix} \exists \text{ Student } B \text{ such that:} \\ (1)\ \mathcal{P} \text{ proves that } B \text{ wins the Student-Teacher game on } C_r; \\ (2)\ B_i(s_1^*,\ldots,s_{i-1}^*) = C_r(s_i^*) \text{ for all } i \in [j]. \end{bmatrix}.$$

Consider the following algorithm: on an input $y \in \{0,1\}^n$, let

$$p(y) := \Pr_{r\leftarrow\{0,1\}^m} \begin{bmatrix} \exists \text{ Student } B \text{ such that:} \\ (1)\ \mathcal{P} \text{ proves that } B \text{ wins the Student-Teacher game on } C_r; \\ (2)\ B_i(s_1^*,\ldots,s_{i-1}^*) = C_r(s_i^*) = \mathsf{Ext}(G(s_i^*), r) \text{ for all } i \in [j], \\ \quad \text{and } B_{j+1}(s_1^*,\ldots,s_j^*) = \mathsf{Ext}(y, r). \end{bmatrix}.$$

Our algorithm accepts y if $p(y) \geq \Phi_j \cdot 2^{-m} - \varepsilon$, and rejects if $p(y) \leq \Phi_{j+1}$. This can be implemented by the Goldwasser–Sipser set lower bound protocol since $\Phi_j \cdot 2^{-m} - \varepsilon > 2 \cdot \Phi_{j+1}$, and the condition inside $\Pr_{r\leftarrow\{0,1\}^m}[\cdot]$ is certifiable in polynomial time with the help of a prover. Finally, we prove that this algorithm breaks demi-bits generator G:

- **For any $y \in \mathrm{Range}(G)$, we have $p(y) \leq \Phi_{j+1}$:**

  Suppose $y = G(s)$. Then

$$p(y) = \Pr_{r\leftarrow\{0,1\}^m} \begin{bmatrix} \exists \text{ Student } B \text{ such that:} \\ (1)\ \mathcal{P} \text{ Proves that } B \text{ wins the Student-Teacher game on } \mathsf{Ext}(G(\cdot), r); \\ (2)\ B_i(s_1^*,\ldots,s_{i-1}^*) = \mathsf{Ext}(G(s_i^*), r) \text{ for all } i \in [j], \\ \quad \text{and } B_{j+1}(s_1^*,\ldots,s_j^*) = \mathsf{Ext}(G(s), r). \end{bmatrix}$$

$\leq \Phi_{j+1}$.

Where the $\leq$ in the second line follows from the definition of Φ_{j+1}.

- **For half of $y \in \{0,1\}^n$, we have $p(y) \geq \Phi_j \cdot 2^{-m} - \varepsilon$:**

  For simplicity, we use "$\mathsf{Test}(r, \mathsf{Ext}(y,r))$" to denote the condition inside $\Pr_{r \leftarrow \{0,1\}^m}[\cdot]$ in the definition of $p(y)$. We have:

$$\Pr_{\substack{r \leftarrow \{0,1\}^m \\ z \leftarrow \{0,1\}^m}} [\mathsf{Test}(r,z)] = \Phi_j \cdot 2^{-m}.$$

  By the definition of strong extractors, for half of $y \in \{0,1\}^n$, we have

$$\Pr_{r \leftarrow \{0,1\}^m} [\mathsf{Test}(r, \mathsf{Ext}(y,r))] \geq \Pr_{\substack{r \leftarrow \{0,1\}^m \\ z \leftarrow \{0,1\}^m}} [\mathsf{Test}(r,z)] - \varepsilon = \Phi_j \cdot 2^{-m} - \varepsilon. \qquad \square$$

Corollary 9.4.4. *Suppose for any parameter $N \leq \mathrm{poly}(n)$, there exists a demi-bits generator $G : \{0,1\}^n \to \{0,1\}^N$ secure against $\mathsf{NP}/_{\mathrm{poly}}$. Then for every non-uniform propositional proof system $\mathcal{P}$ and any parameters $k \leq \mathrm{poly}(n)$ and $n < m < \mathrm{poly}(n)$, there is a circuit $C : \{0,1\}^n \to \{0,1\}^m$ of size $\mathrm{poly}(n)$ such that C is a k-round pseudo-surjective proof complexity generator secure against $\mathcal{P}$.*

Proof. In Theorem 9.4.3, let $N := 100km$ and $\mathsf{Ext}\colon \{0,1\}^N \times \{0,1\}^{O(km)} \to \{0,1\}^m$ be an $(N - 1, 2^{-10km})$-strong seeded extractor guaranteed by Theorem 9.2.19. Then there exists $r \in \{0,1\}^{O(km)}$ such that $C_r := \mathsf{Ext}(G(\cdot), r)$ is a k-round pseudo-surjective proof complexity generator secure against $\mathcal{P}$. $\square$

9.5 Candidate Demi-Bits Generators

9.5.1 Demi-Bits Generators with Polynomial Stretch

Assumption 9.2.2 (demi-bits generators with polynomial stretch) follows from Rudich's conjecture on the unprovability of circuit lower bounds [Rud97], with the *truth table generator* TT being a candidate demi-bits generator.

For a Boolean function $f : \{0,1\}^n \to \{0,1\}$ and a parameter $s(n) \leq \mathrm{poly}(n)$, let $\mathsf{lb}(f, s)$ denote the sentence stating that "f requires circuit complexity at least $s(n)$". This sentence can be written as a CNF of size $2^{O(n)}$ and, if true, admits a trivial proof of length $2^{\tilde{O}(s(n))}$ in most reasonable proof systems. *Rudich's conjecture* asserts that there is no (non-uniform) propositional proof system that has length-$2^{O(n)}$ proofs of $\mathsf{lb}(f, s)$ for a large fraction of Boolean functions f. This can be equivalently formulated as the non-existence of "$\mathsf{NP}/_{\mathrm{poly}}$-natural proofs" against polynomial-size circuits (see [Rud97] for more details). Rudich's conjecture was further investigated in [PS19, ST21].

It is easy to see that Rudich's conjecture is equivalent to the demi-hardness of the "truth table generator" $\mathsf{TT} : \{0,1\}^{O(s \log s)} \to \{0,1\}^{2^n}$: Given as input the description of a size-s circuit C, $\mathsf{TT}(C)$ outputs the truth table of C. As we only want demi-bits generators with polynomial stretch, we can set the parameter $s(n)$ to be $2^{\varepsilon n}$ for some constant $\varepsilon > 0$.

Rudich's conjecture follows from the existence of any *super-bits* generator $g : \{0,1\}^\ell \to \{0,1\}^{\ell+1}$ [Rud97, GGM86].[11] It is open whether Rudich's conjecture also follows from the existence of any demi-bits generator $g : \{0,1\}^\ell \to \{0,1\}^{\ell+1}$, i.e., whether TT is the "most secure" demi-bits generator (see Open Problem 4 of [Rud97]). Hence, we have:

Proposition 9.5.1. *Assumption 9.2.2 follows from either:*

- *Rudich's conjecture on the unprovability of random circuit lower bounds; or*
- *the existence of super-bits generators.*

9.5.2 Constant-Degree Demi-Bits Generators from LPN

Learning Parity with Noise (LPN) is the assumption that noisy linear equations over $\mathbb{F}_2$ are hard to solve. Let $n \in \mathbb{N}$ be the number of variables, $m := m(n) \in \mathbb{N}$ be the number of equations, and $\mu := \mu(n) \in (0,1)$ be the noise rate. Here we work in the regime where $m = n^{1+\varepsilon}$ and $\mu = n^{-\varepsilon}$ for some constant $\varepsilon > 0$. Let $A \leftarrow \mathbb{F}_2^{m \times n}$ be a random matrix, $\vec{s} \in \mathbb{F}_2^n$ be a hidden random vector (i.e., the solution), and $\vec{e} \leftarrow \mathsf{Ber}(\mu)^m$ be a hidden noise vector where each entry is equal to 1 w.p. μ independently. The LPN assumption asserts that the following two distributions are computationally indistinguishable:

$$(A, A\vec{s} + \vec{e}) \quad \text{v.s.} \quad (A, \mathcal{U}_m).$$

Roughly speaking, we will assume the above indistinguishability holds even for nondeterministic adversaries, in the sense that no non-uniform proof system can efficiently prove a vector $\vec{v} \in \mathbb{F}^m$ is *not* of the form $A\vec{s} + \vec{e}$ when $\vec{e}$ is a $(\mu \cdot m)$-sparse vector. That is:

Assumption 9.5.2. For some (public) matrix $A \in \mathbb{F}_2^{m \times n}$, there is no polynomial-size non-uniform nondeterministic circuit $C : \{0,1\}^m \to \{0,1\}$ such that C accepts a constant fraction of random strings but rejects every string of the form $A\vec{s} + \vec{e}$, where $\vec{e} \in \mathbb{F}_2^m$ is $(\mu \cdot m)$-sparse and $\vec{s} \in \mathbb{F}_2^n$.

Fact 9.5.3. *Assumption 9.5.2 implies Assumption 9.2.3.*

Proof. Let $d := \lceil 2/\varepsilon \rceil$. Since $\mu \cdot m < \frac{m^{1-1/d}}{d}$, by [GGNS23, Lemma 3.1], there exists a polynomial-time computable function $f : \mathbb{F}_2^{O(\mu m^{1+1/d})} \to \mathbb{F}_2^m$ whose range contains all vectors of sparsity at most s, such that each output of f is a degree-d polynomial.

Now consider the following generator $g : \mathbb{F}_2^{O(\mu m^{1+1/d})+n} \to \mathbb{F}_2^m$. The input of g consists of $\overrightarrow{e_{\mathsf{enc}}} \in \mathbb{F}_2^{O(\mu m^{1+1/d})}$ and $\vec{s} \in \mathbb{F}_2^n$. The output is $A\vec{s} + f(\overrightarrow{e_{\mathsf{enc}}})$ (where A is hardwired in the circuit computing g). It is easy to see that every output bit of g is computable by a degree-d polynomial over $\mathbb{F}_2$, and the range of g contains every vector of the form $A\vec{s} + \vec{e}$ where $\vec{e}$ is $(\mu \cdot m)$-sparse and $\vec{s} \in \mathbb{F}_2^n$. The input length of g is $O(\mu m^{1+1/d}) + n \le O(n^{1+\varepsilon/2})$, which is polynomially smaller than the output length $m = n^{1+\varepsilon}$. □

[11] This is true for $s(n) = 2^{\Omega(n)}$. If super-bits generators with *subexponential* security exist, then TT is a secure demi-bits generator even for $s(n) = \mathrm{poly}(n)$.

9.5.3 Constant-Degree Demi-Bits Generators from Goldreich's Generator

Goldreich's generator is an influential candidate pseudorandom generator with large stretch that is computable with constant locality (i.e., in NC^0) [Gol11a]. To define the generator $G : \{0,1\}^n \to \{0,1\}^m$, fix a d-uniform hypergraph with n vertices and m (ordered) hyperedges each of size d, and a predicate $P : \{0,1\}^d \to \{0,1\}$. For each $i \in [m]$, the i-th output bit is obtained by applying P to the input bits on the i-th hyperedge. That is, suppose the i-th hyperedge contains vertices $v_{i,1}, v_{i,2}, \ldots, v_{i,d}$, then on input $x \in \{0,1\}^n$, the i-th output bit is

$$G(x)_i := P(x_{v_{i,1}}, x_{v_{i,2}}, \ldots, x_{v_{i,d}}).$$

It seems plausible to conjecture that Goldreich's generator is a secure demi-bits generator when instantiated with a suitable hypergraph and a suitable predicate P:

Assumption 9.5.4. There exist constants $c, d > 1$, a predicate $P : \{0,1\}^d \to \{0,1\}$, and a non-uniform family of d-hypergraphs $\{\mathcal{G}_n\}_{n\in\mathbb{N}}$ with n^c hyperedges such that Goldreich's PRG $G : \{0,1\}^n \to \{0,1\}^{n^c}$ instantiated with P and $\{\mathcal{G}_n\}$ is a secure demi-bits generator.

Clearly, Assumption 9.5.4 implies Assumption 9.2.3.

In fact, the following predicate $P : \mathbb{F}_2^5 \to \mathbb{F}_2$ is frequently considered in the literature:

$$P_{\mathrm{MST06}}(x_{1\sim 5}) := x_1 + x_2 + x_3 + x_4 x_5.$$

Note that P_{MST06} is a degree-2 function over $\mathbb{F}_2$. It was shown in [MST06] that some instantiation of Goldreich's PRG with P_{MST06} fools linear tests. Instantiations using this predicate were also conjectured to be secure against polynomial-time adversaries in [BKR23]. Instantiating Goldreich's PRG with the predicate P_{MST06} and a suitable family of 5-hypergraphs gives us a candidate demi-bits generator computable by *degree-2 polynomials over* $\mathbb{F}_2$.

To summarise, we have:

Proposition 9.5.5 (Informal)**.** *Assumption 9.2.3 follows from either:*

- *the demi-hardness of Learning Parity with Noise in certain parameter regimes; or*
- *the demi-hardness of certain suitably instantiated Goldreich's generator.*

9.6 Hardness of Range Avoidance from Predictable Arguments

Ilango, Li, and Williams [ILW23] proved Avoid $\notin$ FP assuming the existence of JLS-secure $i\mathcal{O}$ and NP $\neq$ coNP. Given our main result that the existence of demi-bits generators implies the hardness of Avoid, it is natural to ask whether the $i\mathcal{O}$ assumption used in [ILW23] can be weakened to a "Minicrypt" assumption. In particular, we conjecture:

Conjecture 9.6.1. Avoid $\notin$ FP *follows from the existence of* one-way functions *and* NP $\neq$ coNP.

Unfortunately, we are not able to prove Conjecture 9.6.1. Instead, in this section, we present an alternative interpretation of [ILW23]. Although the results and proofs are not new, we hope

that our new perspective helps make progress towards proving Conjecture 9.6.1, or in general, basing Avoid $\notin$ FP from minimal assumptions.

Witness encryptions. Instead of $i\mathcal{O}$, what [ILW23] actually needs is a primitive called *witness encryption* [GGSW13]. Let $L \in \mathsf{NP}$ (usually we take L to be an NP-complete language such as SAT). A *witness encryption scheme* for L is a pair of algorithms $(\mathsf{Enc}, \mathsf{Dec})$ with the following interface:[12]

- Given an instance $x \in \{0,1\}^n$, a message m, a security parameter 1^λ, and some random coins r, $\mathsf{Enc}(x, b, 1^\lambda; r)$ outputs the encryption of m under x.
- Given an instance $x \in \{0,1\}^n$, a witness w that $x \in L$, a ciphertext ct, and the security parameter 1^λ, $\mathsf{Dec}(x, w, ct, 1^\lambda)$ outputs the encrypted message m. (We assume that Dec is deterministic.)

We require $(\mathsf{Enc}, \mathsf{Dec})$ to be *correct* and $2^{-\lambda^\varepsilon}$*-secure* for some constant $\varepsilon > 0$; here we say

- $(\mathsf{Enc}, \mathsf{Dec})$ is *(perfectly) correct* if for every $x \in L$, witness w for x, bit b, and randomness r, it is the case that
$$\mathsf{Dec}(x, w, \mathsf{Enc}(x, b, 1^\lambda; r), 1^\lambda) = b.$$
- $(\mathsf{Enc}, \mathsf{Dec})$ is $\delta(\cdot)$*-secure* if for every $x \notin L$, message m, and every non-uniform adversary $\mathcal{A}$ of size $\mathrm{poly}(n)$,
$$\left|\Pr[\mathcal{A}(\mathsf{Enc}(x, m, 1^\lambda)) = 1] - \Pr[\mathcal{A}(\mathcal{U}) = 1]\right| < \delta(n).$$

Hardness of Avoid from predictable arguments. Witness encryption implies the following *predictable argument system* [FNV17] for L. Let $x \in \{0,1\}^n$ be an instance known to both the prover and the verifier. Recall that in an argument system, the prover is computationally bounded. If $x \in L$, then the prover also has access to a witness $w \in \{0,1\}^m$. Let $\ell > m$ be a parameter and $\lambda := \ell^{2/\varepsilon}$.

- First, the Verifier picks a random message $m \leftarrow \{0,1\}^\ell$, encrypts m as $ct \leftarrow \mathsf{Enc}(x, m, 1^\lambda)$, and sends ct to the Prover.
- Then the Prover sends a message m'. In particular, the honest Prover sends $m' \leftarrow \mathsf{Dec}(x, w, ct, 1^\lambda)$.
- The Verifier accepts if and only if $m' = m$.

Now, suppose that there is a deterministic algorithm $\mathcal{A}$ solving Avoid, we show that $L \in$ coNP. The idea is to use $\mathcal{A}$ as the prover in the above argument system. In particular, let $C_{x,ct} : \{0,1\}^m \to \{0,1\}^\ell$ denote the circuit that given $w \in \{0,1\}^m$ as input, outputs $\mathsf{Dec}(x, w, ct, 1^\lambda)$. Upon receiving ct, Prover always sends $\mathcal{A}(C_{x,ct})$ to Verifier.

[12] Witness encryption schemes as defined in [GGSW13] can only encrypt a one-bit message $m \in \{0,1\}$. To obtain a witness encryption scheme that can encrypt an arbitrarily long message, one can simply encrypt each message bit independently. The security of this new scheme follows easily from a hybrid argument.

Claim 9.6.2. *If $x \in L$, then the* Prover *will never convince the* Verifier.

Proof. The only message that convinces the verifier is $\tilde{m} := \mathsf{Dec}(x, w, ct, 1^\lambda)$, where w is a witness of $x \in L$. Clearly, $\tilde{m}$ is in the range of $C_{x,ct}$. □

Claim 9.6.3. *If $x \notin L$, then the* Prover *has a non-zero probability of convincing the* Verifier.

Proof. Let $m^\star \in \{0,1\}^\ell$ be any string such that $\mathcal{A}(ct) = m^\star$ with probability at least $2^{-\ell}$ over a truly random ct. Since $\mathcal{A}$ runs in polynomial time (which means the security of witness encryption holds for $\mathcal{A}$), the probability over $ct \leftarrow \mathsf{Enc}(x, m^\star, 1^\lambda)$ that $\mathcal{A}(ct) = m^\star$ is at least $2^{-\ell} - 2^{-\lambda^\varepsilon} > 0$. □

The above claims imply a nondeterministic algorithm for deciding the complement of L. On input $x \in \{0,1\}^n$, guess the Verifier's first message m and randomness r, compute $ct := \mathsf{Enc}(x, m, 1^\lambda; r)$, and accept if the Verifier accepts when the Prover replies with $\mathcal{A}(C_{x,ct})$.

In summary, witness encryption implies that NP has a *special type of predictable argument system*. Moreover, if Avoid ∈ FP, then plugging the Avoid algorithm as the prover results in the following intriguing situation: if $x \in L$, then the Verifier *never* accepts, while if $x \notin L$, then the Verifier accepts with *non-zero* probability! This allows us to put every language with such argument systems in coNP.

An interesting question is to identify the weakest possible argument system for NP such that the above situation happens. Can we build such argument systems using only one-way functions? Such an argument system would make progress towards proving Conjecture 9.6.1.

Chapter 10

Conclusions and Future Directions

In this thesis, we studied explicit construction problems through the lens of complexity theory. Our results reveal an intimate and bidirectional connection between the two: techniques from complexity theory yield new algorithms for explicit construction problems, and conversely, understanding the complexity of explicit construction problems has important consequences back to complexity theory as well.

A central computational problem arising from the study of explicit construction problems is the *Range Avoidance problem* (Avoid). Our findings suggest that understanding the complexity of Avoid is a key step towards resolving many important open questions, both in explicit constructions and in complexity theory more broadly.

We conclude by highlighting several future research directions that emerge from this thesis:

Algorithmic Methods for small-stretch Avoid. A drawback of the framework developed in Chapter 3 is that, even given the best possible Satisfying-Pairs algorithms, it could only solve Avoid instances with *large* stretch, i.e., circuits $C : \{0,1\}^n \to \{0,1\}^\ell$ where $\ell \geq n^{1+\Omega(1)}$. Many (if not most) interesting explicit construction problems reduce to Avoid with *small stretch*, where our Algorithmic Method does not appear to be helpful for obtaining $\mathsf{FP}^{\mathsf{NP}}$-explicit constructions for them.

Consider, for instance, the problem of constructing Ramsey graphs:

Example 10.0.1. A graph $G = (V, E)$ over $n = |V|$ vertices is a *Ramsey graph* if it does not contain cliques or independent sets of size at least (say) $10 \log n$. It requires $N := \binom{n}{2}$ bits to encode an n-vertex graph G. However, if G is not Ramsey and $S \subseteq V$ is a clique or independent set of size $10 \log n$ in G, then G can be described more succinctly: specify S, a bit $b \in \{0,1\}$ (indicating whether S is a clique or an independent set), and the $\binom{n}{2} - \binom{10 \log n}{2}$ edges outside S. This description uses $M := (10 \log n) \cdot \log n + 1 + \binom{n}{2} - \binom{10 \log n}{2} \leq N - 30 \log n$ bits.

Let $C_{\mathsf{Ramsey}} : \{0,1\}^M \to \{0,1\}^N$ be the circuit that maps a length-M succinct encoding of a non-Ramsey graph G into its adjacency matrix. Then, explicitly constructing a Ramsey graph is equivalent to solving the Range Avoidance problem on the instance C_{Ramsey}.

Note that every output bit of C_{Ramsey} can be computed in polylog(n) time, so C_{Ramsey} has very low circuit complexity. However, its stretch is very small (there are only $\Theta(\log N)$ more output bits than input bits), preventing us from obtaining $\mathsf{FP}^{\mathsf{NP}}$-explicit constructions of Ramsey graphs using Chapter 3. What kind of circuit-analysis algorithms are sufficient for handling such

small-stretch Avoid instances? Is there a generic Algorithmic Method for solving Avoid when the stretch is small?

Toward optimal win-win arguments in complexity theory. The results in Chapter 6 and Chapter 7 suggest that many complexity-theoretic results based on win-win arguments can be improved to near-optimal by an *iterative win-win* approach.

An immediate open problem in this direction is to prove an *exponential* circuit lower bound for $\mathsf{MA\text{-}E}/_1$, the exponential-time analogue of MA with one-bit advice. Buhrman, Fortnow, and Thierauf [BFT98] proved a super-polynomial lower bound for this class using a single-step win-win argument. Can we obtain a near-optimal lower bound via an iterative win-win argument?

Another example comes from Impagliazzo and Wigderson [IW01], who showed that if $\mathsf{EXP} \neq \mathsf{BPP}$, then BPP admits subexponential-time heuristic derandomisation on infinitely many input lengths. It is unclear whether a "high-end" analogue of this result holds: if $\mathsf{EXP} \not\subseteq \mathsf{BPTIME}[2^{n^{o(1)}}]$, does it follow that BPP admits infinitely-often *quasi-polynomial time* heuristic derandomisation? A related question is to prove the prRP-hardness of `Implicit-Heavy-Avoid` (Theorem 8.4.8) in the polynomial-time regime, i.e., showing that a polynomial-time algorithm for `Implicit-Heavy-Avoid` implies an infinitely-often* *polynomial-time* algorithm for Gap-SAT (instead of only a subexponential-time algorithm).

We briefly list additional win-win arguments whose current bounds are non-optimal, with the hope that the iterative win-win method may strengthen them to optimal results:

- **Easy-witness lemma.** Impagliazzo, Kabanets, and Wigderson [IKW02] showed that if $\mathsf{NEXP} \subseteq \mathsf{P}/_{\mathrm{poly}}$, then NEXP has "easy witnesses" in the sense that every accepting NEXP computation has an (exponentially-long) witness that is the truth table of a polynomial-size circuit. Is there an analogue for subexponential size bounds? That is, if $\mathsf{NEXP} \subseteq \mathsf{SIZE}[2^{n^{o(1)}}]$, is it true that every accepting NEXP computation has a witness with circuit complexity $\leq 2^{n^{o(1)}}$?

- **ACC^0 lower bounds.** Williams [Wil14] proved $\mathsf{NEXP} \not\subseteq \mathsf{ACC}^0$ using non-trivial circuit-analysis algorithms for ACC^0. These circuit-analysis algorithms are able to handle ACC^0 circuits of subexponential size, but converting this into a subexponential ACC^0 circuit lower bound for NEXP remains open.

  As shown in [MW20, Che19], ACC^0 lower bounds for NEXP follow from (variants of) the easy-witness lemma, which in turn follows from (refined versions of) circuit lower bounds for $\mathsf{MA}/_1$ and $\mathsf{MA\text{-}E}/_1$ [BFT98, San09]. Thus, progress on any one of these fronts could unlock progress on the others.

- **Circuit lower bounds for $\mathsf{BPEXP}^{\mathsf{MCSP}}$.** Impagliazzo, Kabanets, and Volkovich [IKV18] proved that $\mathsf{ZPEXP}^{\mathsf{MCSP}} \not\subseteq \mathsf{P}/_{\mathrm{poly}}$. Partially inspired by this result, Hirahara, Lu, and Ren [HLR23] obtained a near-maximum circuit lower bound for $\mathsf{BPEXP}^{\mathsf{MCSP}}$ with $2^{\varepsilon n}$ bits of advice, where $\varepsilon > 0$ is an arbitrary constant. Can we prove an exponential lower bound for $\mathsf{BPEXP}^{\mathsf{MCSP}}$ (possibly with one bit of advice), improving the circuit lower bound in [IKV18] to near-optimal and getting rid of the $2^{\varepsilon n}$ advice bits in [HLR23]?

- **RP vs. ZPP.** Kabanets [Kab01] showed that every RP algorithm can be simulated by a subexponential-time ZPP algorithm infinitely often that "appears correct" to every efficient adversary. Williams [Wil16] later gave a worst-case simulation but with a fixed polynomial amount of advice bits, i.e., he showed that there exists a constant $c > 0$ such that $\mathsf{RP} \subseteq$ i.o.-$\mathsf{ZPSUBEXP}/_{n^c}$. Can we improve these simulations to only have a polynomial overhead? In particular, can we show that $\mathsf{RP} \subseteq$ i.o.-$\mathsf{ZPP}/_{n^c}$?

Bibliography

[Aar06] Scott Aaronson. Oracles are subtle but not malicious. In *CCC*, pages 340–354. IEEE Computer Society, 2006. doi:10.1109/CCC.2006.32. 199

[AB09] Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, 2009. doi:10.1017/CBO9780511804090. 1, 2, 137, 138, 166, 177, 212, 284

[AB18] Amir Abboud and Karl Bringmann. Tighter connections between Formula-SAT and shaving logs. In *ICALP*, volume 107 of *LIPIcs*, pages 8:1–8:18, 2018. doi:10.4230/LIPIcs.ICALP.2018.8. 102

[ABN+92] Noga Alon, Jehoshua Bruck, Joseph Naor, Moni Naor, and Ron M. Roth. Construction of asymptotically good low-rate error-correcting codes through pseudo-random graphs. *IEEE Transactions on Information Theory*, 38(2):509–516, 1992. doi:10.1109/18.119713. 2

[ABRW04] Michael Alekhnovich, Eli Ben-Sasson, Alexander A. Razborov, and Avi Wigderson. Pseudo-random generators in propositional proof complexity. *SIAM J. Comput.*, 34(1):67–88, 2004. doi:10.1137/S0097539701389944. 277, 278, 286

[AC19] Josh Alman and Lijie Chen. Efficient construction of rigid matrices using an NP oracle. In *FOCS*, pages 1034–1055, 2019. doi:10.1109/FOCS.2019.00067. 3, 4, 5, 20

[Adl78] Leonard M. Adleman. Two theorems on random polynomial time. In *FOCS*, pages 75–83, 1978. doi:10.1109/SFCS.1978.37. 104, 107, 235, 285

[AG91] Eric Allender and Vivek Gore. On strong separations from AC^0 (extended abstract). In *Fundamentals of Computation Theory, 8th International Symposium, FCT '91*, volume 529 of *Lecture Notes in Computer Science*, pages 1–15. Springer, 1991. doi:10.1007/3-540-54458-5_44. 25, 80

[AGHP92] Noga Alon, Oded Goldreich, Johan Håstad, and René Peralta. Simple constructions of almost k-wise independent random variables. *Random Structures Algorithms*, 3(3):289–304, 1992. doi:10.1002/rsa.3240030308. 2, 125

[AHWW16] 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. In *STOC*, pages 375–388. ACM, 2016. doi:10.1145/2897518.2897653. 20, 102

[Ajt83] Miklós Ajtai. Σ^1_1-formulae on finite structures. *Ann. Pure. Appl. Log.*, 24(1):1–48, 1983. doi:10.1016/0168-0072(83)90038-6. 24

[Ajt90] Miklós Ajtai. Approximate counting with uniform constant-depth circuits. In *Advances In Computational Complexity Theory*, volume 13 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 1–20. DIMACS/AMS, 1990. doi:10.1090/DIMACS/013/01. 249, 252, 253

[Ajt96] Miklós Ajtai. Generating hard instances of lattice problems (extended abstract). In *STOC*, pages 99–108, 1996. doi:10.1145/237814.237838. 282

[AKS04] Manindra Agrawal, Neeraj Kayal, and Nitin Saxena. PRIMES is in P. *Annals of Mathematics*, 160(2):781–793, 2004. doi:10.4007/annals.2004.160.781. 3, 151, 153, 202

[ALM+98] Sanjeev Arora, Carsten Lund, Rajeev Motwani, Madhu Sudan, and Mario Szegedy. Proof verification and the hardness of approximation problems. *Journal of the ACM*, 45(3):501–555, 1998. doi:10.1145/278298.278306. 269

[Alo98] Noga Alon. The shannon capacity of a union. *Comb.*, 18(3):301–310, 1998. doi:10.1007/PL00009824. 1

[ALR99] Eric Allender, Michael C. Loui, and Kenneth W. Regan. Reducibility and completeness. In *Algorithms and Theory of Computation Handbook*, Chapman & Hall/CRC Applied Algorithms and Data Structures series. CRC Press, 1999. URL: https://dl.acm.org/doi/10.5555/1882757.1882780. 275

[APY09] Noga Alon, Rina Panigrahy, and Sergey Yekhanin. Deterministic approximation algorithms for the nearest codeword problem. In *APPROX-RANDOM*, volume 5687 of *Lecture Notes in Computer Science*, pages 339–351. Springer, 2009. doi:10.1007/978-3-642-03685-9_26. 22

[AS10] Vikraman Arvind and Srikanth Srinivasan. Circuit lower bounds, help functions, and the remote point problem. In *Innovations in Computer Science - ICS 2010, Tsinghua University, Beijing, China, January 5-7, 2010. Proceedings*, pages 383–396, 2010. URL: http://conference.iiis.tsinghua.edu.cn/ICS2010/content/papers/30.html. 22, 23, 24, 26, 82

[AW17] Josh Alman and R. Ryan Williams. Probabilistic rank and matrix rigidity. In *STOC*, pages 641–652. ACM, 2017. doi:10.1145/3055399.3055484. 2

[Bab85] László Babai. Trading group theory for randomness. In *STOC*, pages 421–429. ACM, 1985. doi:10.1145/22145.22192. 284, 285

[Bar89] David A. Mix Barrington. Bounded-width polynomial-size branching programs recognize exactly those languages in NC^1. *J. Comput. Syst. Sci.*, 38(1):150–164, 1989. doi:10.1016/0022-0000(89)90037-8. 102

[Bar06] Boaz Barak. A simple explicit construction of an $n^{\tilde{O}(\log n)}$-Ramsey graph. *arXiv preprint*, math/0601651, 2006. doi:10.48550/arXiv.math/0601651. 1

[BB18] Ben Berger and Zvika Brakerski. Zero-knowledge protocols for search problems. In *International Conference on Security and Cryptography for Networks* (SCN), pages 292–309, 2018. doi:10.1007/978-3-319-98113-0_16. 152

[BBP95] Maria Luisa Bonet, Samuel R. Buss, and Toniann Pitassi. Are there hard examples for Frege systems? In *Feasible Mathematics II*, pages 30–56, Boston, MA, 1995. Birkhäuser Boston. doi:10.1007/978-1-4612-2566-9_3. 278

[BCG+96] Nader H. Bshouty, Richard Cleve, Ricard Gavaldà, Sampath Kannan, and Christino Tamon. Oracles and queries that are sufficient for exact learning. *J. Comput. Syst. Sci.*, 52(3):421–433, 1996. doi:10.1006/jcss.1996.0032. 199, 211

[BCGT13] Eli Ben-Sasson, Alessandro Chiesa, Daniel Genkin, and Eran Tromer. On the concrete efficiency of probabilistically-checkable proofs. In *STOC*, pages 585–594. ACM, 2013. doi:10.1145/2488608.2488681. 258

[BDT16] Avraham Ben-Aroya, Dean Doron, and Amnon Ta-Shma. Explicit two-source extractors for near-logarithmic min-entropy. *Electron. Colloquium Comput. Complex.*, TR16-088, 2016. URL: https://eccc.weizmann.ac.il/report/2016/088. 1

[BF99] Harry Buhrman and Lance Fortnow. One-sided versus two-sided error in probabilistic computation. In *STACS*, volume 1563 of *Lecture Notes in Computer Science*, pages 100–109. Springer, 1999. doi:10.1007/3-540-49116-3_9. 240, 260, 261, 266

[BFL91] László Babai, Lance Fortnow, and Carsten Lund. Non-deterministic exponential time has two-prover interactive protocols. *Computational Complexity*, 1:3–40, 1991. doi:10.1007/BF01200056. 221, 255, 258

[BFNW93] László Babai, Lance Fortnow, Noam Nisan, and Avi Wigderson. BPP has subexponential time simulations unless EXPTIME has publishable proofs. *Computatioanl Complexity*, 3:307–318, 1993. doi:10.1007/BF01275486. 211

[BFT98] Harry Buhrman, Lance Fortnow, and Thomas Thierauf. Nonrelativizing separations. In *CCC*, pages 8–12, 1998. doi:10.1109/CCC.1998.694585. 199, 304

[BGH+05] Eli Ben-Sasson, Oded Goldreich, Prahladh Harsha, Madhu Sudan, and Salil P. Vadhan. Short PCPs verifiable in polylogarithmic time. In *CCC*, pages 120–134, 2005. doi:10.1109/CCC.2005.27. 10, 27, 114, 115, 116, 117

[BGH+06] Eli Ben-Sasson, Oded Goldreich, Prahladh Harsha, Madhu Sudan, and Salil Vadhan. Robust PCPs of proximity, shorter PCPs and applications to coding. *SIAM J. Comput.*, 36(4):889–974, 2006. doi:10.1137/S0097539705446810. 11, 13, 26, 96, 112, 113, 114, 115, 117, 124, 126, 131, 144, 258, 269

[BGI+12] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. *Journal of the ACM*, 59(2):6:1–6:48, 2012. doi:10.1145/2160158.2160159. 6

[BGIP01] Samuel R. Buss, Dima Grigoriev, Russell Impagliazzo, and Toniann Pitassi. Linear gaps between degrees for the polynomial calculus modulo distinct primes. *J. Comput. Syst. Sci.*, 62(2):267–289, 2001. doi:10.1006/JCSS.2000.1726. 292

[BH92] Harry Buhrman and Steven Homer. Superpolynomial circuits, almost sparse oracles and the exponential hierarchy. In *FSTTCS*, volume 652 of *Lecture Notes in Computer Science*, pages 116–127. Springer, 1992. doi:10.1007/3-540-56287-7_99. 256

[BHP01] R. C. Baker, G. Harman, and J. Pintz. The difference between consecutive primes. II. *Proc. London Math. Soc. (3)*, 83(3):532–562, 2001. doi:10.1112/plms/83.3.532. 1, 151

[BHPT24] Amey Bhangale, Prahladh Harsha, Orr Paradise, and Avishay Tal. Rigid matrices from rectangular PCPs. *SIAM J. Comput.*, 53(2):480–523, 2024. doi:10.1137/22M1495597. 4, 13, 18, 20, 27, 112, 113, 114, 115, 117, 126, 131, 132, 134

[BI94] David A. Mix Barrington and Neil Immerman. Time, hardware, and uniformity. In *SCT*, pages 176–185. IEEE Computer Society, 1994. doi:10.1109/SCT.1994.315806. 275

[BIS90] David A. Mix Barrington, Neil Immerman, and Howard Straubing. On uniformity within NC^1. *J. Comput. Syst. Sci.*, 41(3):274–306, 1990. doi:10.1016/0022-0000(90)90022-D. 240, 253, 254

[BK95] Manuel Blum and Sampath Kannan. Designing programs that check their work. *J. ACM*, 42(1):269–291, 1995. doi:10.1145/200836.200880. 247

[BKKS23] Vladimir Braverman, Robert Krauthgamer, Aditya Krishnan, and Shay Sapir. Lower bounds for pseudo-deterministic counting in a stream. In *ICALP*, volume 261 of *LIPIcs*, pages 30:1–30:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICS.ICALP.2023.30. 152

[BKR23] Andrej Bogdanov, Pravesh K. Kothari, and Alon Rosen. Public-key encryption, local pseudorandom generators, and the low-degree method. In *TCC (1)*, volume 14369 of *Lecture Notes in Computer Science*, pages 268–285. Springer, 2023. doi:10.1007/978-3-031-48615-9_10. 300

[BKS+10] Boaz Barak, Guy Kindler, Ronen Shaltiel, Benny Sudakov, and Avi Wigderson. Simulating independence: New constructions of condensers, Ramsey graphs, dispersers, and extractors. *J. ACM*, 57(4):20:1–20:52, 2010. doi:10.1145/1734213.1734214. 1

[BKT14] Samuel R. Buss, Leszek Aleksander Kolodziejczyk, and Neil Thapen. Fragments of approximate counting. *J. Symb. Log.*, 79(2):496–525, 2014. doi:10.1017/JSL.2013.37. 280

[BLS85] Ronald V. Book, Timothy J. Long, and Alan L. Selman. Qualitative relativizations of complexity classes. *J. Comput. Syst. Sci.*, 30(3):395–413, 1985. doi:10.1016/0022-0000(85)90053-4. 201

[Blu84] Norbert Blum. A boolean function requiring $3n$ network size. *Theor. Comput. Sci.*, 28:337–345, 1984. doi:10.1016/0304-3975(83)90029-4. 2

[BM84] Manuel Blum and Silvio Micali. How to generate cryptographically strong sequences of pseudo-random bits. *SIAM J. Comput.*, 13(4):850–864, 1984. doi:10.1137/0213053. 164, 177, 282

[Bou05] Jean Bourgain. More on the sum-product phenomenon in prime fields and its applications. *International Journal of Number Theory*, 1(01):1–32, 2005. doi:10.1142/S1793042105000108. 1

[BRSW12] Boaz Barak, Anup Rao, Ronen Shaltiel, and Avi Wigderson. 2-source dispersers for $n^{o(1)}$ entropy, and Ramsey graphs beating the Frankl–Wilson construction. *Annals of Mathematics*, 176:1483–1544, 2012. doi:10.4007/annals.2012.176.3.3. 1

[BS06] Joshua Buresh-Oppenheim and Rahul Santhanam. Making hard problems harder. In *CCC*, pages 73–87. IEEE Computer Society, 2006. doi:10.1109/CCC.2006.26. 206

[BS08] Eli Ben-Sasson and Madhu Sudan. Short PCPs with polylog query complexity. *SIAM J. Comput.*, 38(2):551–607, 2008. doi:10.1137/050646445. 27, 96, 258

[BSVW03] Eli Ben-Sasson, Madhu Sudan, Salil P. Vadhan, and Avi Wigderson. Randomness-efficient low degree tests and short PCPs via epsilon-biased sets. In *STOC*, pages 612–621, 2003. doi:10.1145/780542.780631. 125

[BT94] Richard Beigel and Jun Tarui. On ACC. *Comput. Complex.*, 4:350–366, 1994. doi:10.1007/BF01263423. 25, 80

[BT06a] Andrej Bogdanov and Luca Trevisan. Average-case complexity. *Found. Trends Theor. Comput. Sci.*, 2(1), 2006. doi:10.1561/0400000004. 282

[BT06b] Andrej Bogdanov and Luca Trevisan. On worst-case to average-case reductions for NP problems. *SIAM Journal of Computing*, 36(4):1119–1159, 2006. doi:10.1137/S0097539705446974. 282

[BT11] Avraham Ben-Aroya and Amnon Ta-Shma. A combinatorial construction of almost-Ramanujan graphs using the zig-zag product. *SIAM J. Comput.*, 40(2):267–290, 2011. doi:10.1137/080732651. 2

[BT13] Avraham Ben-Aroya and Amnon Ta-Shma. Constructing small-bias sets from algebraic-geometric codes. *Theory Comput.*, 9:253–272, 2013. doi:10.4086/TOC.2013.V009A005. 2

[Bus87] Samuel R. Buss. The Boolean formula value problem is in ALOGTIME. In *STOC*, pages 123–131. ACM, 1987. doi:10.1145/28395.28409. 247

[BV14] Eli Ben-Sasson and Emanuele Viola. Short PCPs with projection queries. In *ICALP*, volume 8572 of *Lecture Notes in Computer Science*, pages 163–173, 2014. doi:10.1007/978-3-662-43948-7_14. 19, 96, 258

[BW05] Amos Beimel and Enav Weinreb. Monotone circuits for weighted threshold functions. In *CCC*, pages 67–75. IEEE Computer Society, 2005. doi:10.1109/CCC.2005.12. 247

[Cai07] Jin-yi Cai. $\mathsf{S}_2^p \subseteq \mathsf{ZPP}^{\mathsf{NP}}$. *J. Comput. Syst. Sci.*, 73(1):25–35, 2007. doi:10.1016/j.jcss.2003.07.015. 5, 199, 200, 201, 211, 212, 232

[Can96] Ran Canetti. More on BPP and the polynomial-time hierarchy. *Inf. Process. Lett.*, 57(5):237–241, 1996. doi:10.1016/0020-0190(96)00016-6. 209, 212, 227, 230

[CCHO05] Jin-yi Cai, Venkatesan T. Chakaravarthy, Lane A. Hemaspaandra, and Mitsunori Ogihara. Competing provers yield improved Karp–Lipton collapse results. *Inf. Comput.*, 198(1):1–23, 2005. doi:10.1016/j.ic.2005.01.002. 5, 199

[CDM23] Arkadev Chattopadhyay, Yogesh Dahiya, and Meena Mahajan. Query complexity of search problems. In *MFCS*, volume 272 of *LIPIcs*, pages 34:1–34:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023. doi:10.1145/3519935.3520043. 152

[CdRN+23] Jonas Conneryd, Susanna F. de Rezende, Jakob Nordström, Shuo Pang, and Kilian Risse. Graph colouring is hard on average for Polynomial Calculus and Nullstellensatz. In *FOCS*, pages 1–11. IEEE Computer Society, 2023. doi:10.1109/FOCS57990.2023.00007. 282

[CG88] Benny Chor and Oded Goldreich. Unbiased bits from sources of weak randomness and probabilistic communication complexity. *SIAM J. Comput.*, 17(2):230–261, 1988. doi:10.1137/0217015. 1

[CGL+19] Lijie Chen, Shafi Goldwasser, Kaifeng Lyu, Guy N. Rothblum, and Aviad Rubinstein. Fine-grained complexity meets IP = PSPACE. In *SODA*, pages 1–20. SIAM, 2019. doi:10.1137/1.9781611975482.1. 102

[Che18] Lijie Chen. Toward super-polynomial size lower bounds for depth-two threshold circuits. *arXiv preprint*, abs/1805.10698, 2018. doi:10.48550/arXiv.1805.10698. 103

[Che19] Lijie Chen. Non-deterministic quasi-polynomial time is average-case hard for ACC circuits. In *FOCS*, pages 1281–1304, 2019. doi:10.1109/FOCS.2019.00079. 19, 22, 304

[Che23] Lijie Chen. New lower bounds and derandomization for ACC, and a derandomization-centric view on the algorithmic method. In *ITCS*, volume 251 of *LIPIcs*, pages 34:1–34:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICS.ITCS.2023.34. 19, 238, 239, 247, 248, 258, 259, 271, 272, 273, 274, 275

[CHLR23] Yeyuan Chen, Yizhi Huang, Jiatu Li, and Hanlin Ren. Range avoidance, remote point, and hard partial truth table via satisfying-pairs algorithms. In *STOC*, pages 1058–1066. ACM, 2023. doi:10.1145/3564246.3585147. 7

[CHLR25] Lijie Chen, Shuichi Hirahara, Zeyong Li, and Hanlin Ren. Symmetric exponential time requires near-maximum circuit size. *J. ACM*, November 2025. doi:10.1145/3778166. 7

[CHR24] Lijie Chen, Shuichi Hirahara, and Hanlin Ren. Symmetric exponential time requires near-maximum circuit size. In *STOC*, pages 1990–1999. ACM, 2024. doi:10.1145/3618260.3649624. ii, 7, 155

[CHR26] Lijie Chen, Yang Hu, and Hanlin Ren. New algebrization barriers to circuit lower bounds via communication complexity of Missing-String. In *ITCS*, 2026. To appear. doi:10.48550/arXiv.2511.14038. 8

[CIKK16] Marco L. Carmosino, Russell Impagliazzo, Valentine Kabanets, and Antonina Kolokolova. Learning algorithms from natural proofs. In *CCC*, volume 50 of *LIPIcs*, pages 10:1–10:24. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPIcs.CCC.2016.10. 86

[CL16] Eshan Chattopadhyay and Xin Li. Explicit non-malleable extractors, multi-source extractors, and almost optimal privacy amplification protocols. In *FOCS*, pages 158–167. IEEE Computer Society, 2016. doi:10.1109/FOCS.2016.25. 1

[CL21] Lijie Chen and Xin Lyu. Inverse-exponential correlation bounds and extremely rigid matrices from a new derandomized XOR lemma. In *STOC*, pages 761–771, 2021. doi:10.1145/3406325.3451132. 4, 19, 20, 22, 23

[CL24] Yilei Chen and Jiatu Li. Hardness of range avoidance and remote point for restricted circuits via cryptography. In *STOC*, pages 620–629, 2024. doi:10.1145/3618260.3649602. 6, 276, 277, 280, 285

[CLLO21] Lijie Chen, Zhenjian Lu, Xin Lyu, and Igor C. Oliveira. Majority vs. approximate linear sum and average-case complexity below NC^1. In *ICALP*, volume 198 of *LIPIcs*, pages 51:1–51:20, 2021. doi:10.4230/LIPIcs.ICALP.2021.51. 19, 91, 108, 109

[CLO+23] Lijie Chen, Zhenjian Lu, Igor C. Oliveira, Hanlin Ren, and Rahul Santhanam. Polynomial-time pseudodeterministic construction of primes. In *FOCS*, pages 1261–1270. IEEE, 2023. doi:10.1109/FOCS57990.2023.00074. ii, 7, 202, 203, 204, 205, 238, 240, 241, 259, 260

[CLO24] Lijie Chen, Jiatu Li, and Igor C. Oliveira. Reverse mathematics of complexity lower bounds. In *FOCS*, pages 505–527. IEEE, 2024. doi:10.1109/FOCS61266.2024.00040. 287

[CLW20] Lijie Chen, Xin Lyu, and R. Ryan Williams. Almost-everywhere circuit lower bounds from non-trivial derandomization. In *FOCS*, pages 1–12, 2020. doi:10.1109/FOCS46700.2020.00009. 4, 5, 19, 20, 22, 23, 25, 28, 30, 31, 32, 33, 35, 54, 81, 88, 89, 91, 99, 108, 111, 235

[CMMW19] Lijie Chen, Dylan M. McKay, Cody D. Murray, and R. Ryan Williams. Relations and equivalences between circuit lower bounds and Karp–Lipton theorems. In *CCC*, volume 137 of *LIPIcs*, pages 30:1–30:21. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.CCC.2019.30. 211

[CN10] Stephen A. Cook and Phuong Nguyen. *Logical Foundations of Proof Complexity*, volume 11. Cambridge University Press, 2010. doi:10.1017/CBO9780511676277. 287

[Cob64] Alan Cobham. The intrinsic computational difficulty of functions. In *Proc. Logic, Methodology, and the Philosophy of Science*, pages 24–30, 1964. 287

[Coh16a] Gil Cohen. Making the most of advice: New correlation breakers and their applications. In *FOCS*, pages 188–196. IEEE Computer Society, 2016. doi:10.1109/FOCS.2016.28. 1

[Coh16b] Gil Cohen. Two-source extractors for quasi-logarithmic min-entropy and improved privacy amplification protocols. *Electron. Colloquium Comput. Complex.*, TR16-114, 2016. URL: https://eccc.weizmann.ac.il/report/2016/114. 1

[Coh17] Gil Cohen. Towards optimal two-source extractors and Ramsey graphs. In *STOC*, pages 1157–1170. ACM, 2017. doi:10.1145/3055399.3055429. 1

[Coo75] Stephen A. Cook. Feasibly constructive proofs and the propositional calculus (preliminary version). In *STOC*, pages 83–97. ACM, 1975. doi:10.1145/800116.803756. 280, 287

[Cop82] Don Coppersmith. Rapid multiplication of rectangular matrices. *SIAM J. Comput.*, 11(3):467–471, 1982. doi:10.1137/0211037. 79

[COS18] Ruiwen Chen, Igor C. Oliveira, and Rahul Santhanam. An average-case lower bound against ACC^0. In *LATIN*, volume 10807 of *Lecture Notes in Computer Science*, pages 317–330. Springer, 2018. doi:10.1007/978-3-319-77404-6_24. 19, 22

[CPS99] Jin-yi Cai, Aduri Pavan, and D. Sivakumar. On the hardness of permanent. In *STACS*, volume 1563 of *Lecture Notes in Computer Science*, pages 90–99. Springer, 1999. doi:10.1007/3-540-49116-3_8. 282

[CPW23] Suvradip Chakraborty, Manoj Prabhakaran, and Daniel Wichs. A map of witness maps: New definitions and connections. In *Public Key Cryptography (2)*, volume 13941 of *Lecture Notes in Computer Science*, pages 635–662. Springer, 2023. doi:10.1007/978-3-031-31371-4_22. 152

[CR79] Stephen A. Cook and Robert A. Reckhow. The relative efficiency of propositional proof systems. *J. Symb. Log.*, 44(1):36–50, 1979. doi:10.2307/2273702. 285

[CR22] Lijie Chen and Hanlin Ren. Strong average-case circuit lower bounds from nontrivial derandomization. *SIAM J. Comput.*, 51(3):20–115, 2022. doi:10.1137/20M1364886. ii, 19, 22, 23, 54, 86, 88, 89, 90, 108, 109

[Cra36] Harald Cramér. On the order of magnitude of the difference between consecutive prime numbers. *Acta Arithmetica*, 2:23–46, 1936. URL: http://eudml.org/doc/205441. 1, 151

[CRT22] Lijie Chen, Ron D. Rothblum, and Roei Tell. Unstructured hardness to average-case randomness. In *FOCS*, pages 429–437. IEEE, 2022. doi:10.1109/FOCS54457.2022.00048. 156, 262

[CT21a] Lijie Chen and Roei Tell. Hardness vs randomness, revised: Uniform, non-black-box, and instance-wise. In *FOCS*, pages 125–136, 2021. doi:10.1109/FOCS52979.2021.00021. 6, 155, 157, 158, 159, 160, 161, 169, 178, 187, 188, 189, 190, 191, 192, 193, 203, 204, 234, 237, 238, 240, 242, 246, 259, 260, 263, 264, 267, 282

[CT21b] Lijie Chen and Roei Tell. Simple and fast derandomization from very hard functions: eliminating randomness at almost no cost. In *STOC*, pages 283–291, 2021. doi:10.1145/3406325.3451059. 201

[CTW23] Lijie Chen, Roei Tell, and Ryan Williams. Derandomization vs refutation: A unified framework for characterizing derandomization. In *FOCS*, pages 1008–1047. IEEE, 2023. doi:10.1109/FOCS57990.2023.00062. 238, 259

[CW19a] Lijie Chen and Ruosong Wang. Classical algorithms from quantum and Arthur-Merlin communication protocols. In *ITCS*, volume 124 of *LIPIcs*, pages 23:1–23:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.ITCS.2019.23. 20

[CW19b] Lijie Chen and R. Ryan Williams. Stronger connections between circuit analysis and circuit lower bounds, via PCPs of proximity. In *CCC*, volume 137 of *LIPIcs*, pages 19:1–19:43, 2019. doi:10.4230/LIPIcs.CCC.2019.19. 19, 39, 49, 54, 65, 87, 88, 89, 103, 144, 147

[CW21] Timothy M. Chan and R. Ryan Williams. Deterministic APSP, orthogonal vectors, and more: Quickly derandomizing Razborov-Smolensky. *ACM Trans. Algorithms*, 17(1):2:1–2:14, 2021. doi:10.1145/3402926. 3

[CZ19] Eshan Chattopadhyay and David Zuckerman. Explicit two-source extractors and resilient functions. *Annals of Mathematics*, 189(3):653–705, 2019. doi:10.4007/annals.2019.189.3.1. 1, 200

[Din07] Irit Dinur. The PCP theorem by gap amplification. *J. ACM*, 54(3):12, 2007. doi:10.1145/1236457.1236459. 27, 269

[DK11] Evgeny Demenkov and Alexander S. Kulikov. An elementary proof of a $3n-o(n)$ lower bound on the circuit complexity of affine dispersers. In *MFCS*, volume 6907 of *Lecture Notes in Computer Science*, pages 256–265. Springer, 2011. doi:10.1007/978-3-642-22993-0_25. 2

[DPV18] Peter Dixon, Aduri Pavan, and N. V. Vinodchandran. On pseudodeterministic approximation algorithms. In *MFCS*, pages 61:1–61:11, 2018. doi:10.4230/LIPIcs.MFCS.2018.61. 152

[DPV21] Peter Dixon, Aduri Pavan, and N. V. Vinodchandran. Complete problems for multi-pseudodeterministic computations. In *ITCS*, 2021. doi:10.4230/LIPIcs.ITCS.2021.66. 152

[DPWV22] Peter Dixon, Aduri Pavan, Jason Vander Woude, and N. V. Vinodchandran. Pseudodeterminism: promises and lowerbounds. In *STOC*, pages 1552–1565, 2022. doi:10.1145/3519935.3520043. 152

[dRGN+21] Susanna F. de Rezende, Mika Göös, Jakob Nordström, Toniann Pitassi, Robert Robere, and Dmitry Sokolov. Automating algebraic proof systems is NP-hard. In *STOC*, pages 209–222. ACM, 2021. doi:10.1145/3406325.3451080. 292

[dRPR23] Susanna F. de Rezende, Aaron Potechin, and Kilian Risse. Clique is hard on average for unary Sherali-Adams. In *FOCS*, pages 12–25. IEEE, 2023. doi:10.1109/FOCS57990.2023.00008. 282

[Erd59] Paul Erdős. Graph theory and probability. *Canadian Journal of Mathematics*, 11:34–38, 1959. doi:10.4153/CJM-1959-003-9. 1, 200

[ESY84] Shimon Even, Alan L. Selman, and Yacov Yacobi. The complexity of promise problems with applications to public-key cryptography. *Information and Control*, 61(2):159–173, 1984. doi:10.1016/S0019-9958(84)80056-X. 246

[FF93] Joan Feigenbaum and Lance Fortnow. Random-self-reducibility of complete sets. *SIAM J. Comput.*, 22(5):994–1005, 1993. doi:10.1137/0222061. 282

[FGHK16] Magnus Gausdal Find, Alexander Golovnev, Edward A. Hirsch, and Alexander S. Kulikov. A better-than-3n lower bound for the circuit complexity of an explicit function. In *FOCS*, pages 89–98, 2016. doi:10.1109/FOCS.2016.19. 2

[FGJ+26] Noah Fleming, Stefan Grosser, Siddhartha Jain, Jiawei Li, Hanlin Ren, Morgan Shirley, and Weiqiang Yuan. Total search problems in ZPP. In *ITCS*, 2026. To appear. doi:10.48550/arXiv.2512.01138. 8

[FHOS93] Stephen A. Fenner, Steven Homer, Mitsunori Ogiwara, and Alan L. Selman. On using oracles that compute values. In *STACS*, volume 665 of *Lecture Notes in Computer Science*, pages 398–407. Springer, 1993. doi:10.1007/3-540-56503-5_40. 201

[FM05] Gudmund Skovbjerg Frandsen and Peter Bro Miltersen. Reviewing bounds on the circuit size of the hardest functions. *Information Processing Letters*, 95(2):354–357, 2005. doi:10.1016/j.ipl.2005.03.009. 198, 201, 202, 214

[FNV17] Antonio Faonio, Jesper Buus Nielsen, and Daniele Venturi. Predictable arguments of knowledge. In *Public Key Cryptography (1)*, volume 10174 of *Lecture Notes in Computer Science*, pages 121–150. Springer, 2017. doi:10.1007/978-3-662-54365-8_6. 301

[Fri93] Joel Friedman. A note on matrix rigidity. *Comb.*, 13(2):235–239, 1993. doi:10.1007/BF01303207. 2

[FS16] Lance Fortnow and Rahul Santhanam. New non-uniform lower bounds for uniform classes. In *CCC*, volume 50 of *LIPIcs*, pages 19:1–19:14, 2016. doi:10.4230/LIPIcs.CCC.2016.19. 28, 32, 33

[FSS84] Merrick L. Furst, James B. Saxe, and Michael Sipser. Parity, circuits, and the polynomial-time hierarchy. *Math. Syst. Theory*, 17(1):13–27, 1984. doi:10.1007/BF01744431. 24

[FW81] Peter Frankl and Richard M. Wilson. Intersection theorems with geometric consequences. *Comb.*, 1(4):357–368, 1981. doi:10.1007/BF02579457. 1

[GG11] Eran Gat and Shafi Goldwasser. Probabilistic search algorithms with unique answers and their cryptographic applications. *Electron. Colloquium Comput. Complex.*, TR11-136, 2011. URL: https://eccc.weizmann.ac.il/report/2011/136/. 2, 152, 200, 202, 243

[GG17] Shafi Goldwasser and Ofer Grossman. Bipartite perfect matching in pseudo-deterministic NC. In *ICALP*, pages 87:1–87:13, 2017. doi:10.4230/LIPIcs.ICALP.2017.87. 152

[GG21] Sumanta Ghosh and Rohit Gurjar. Matroid intersection: A pseudo-deterministic parallel reduction from search to weighted-decision. In *APPROX-RANDOM*, pages 41:1–41:16, 2021. doi:10.4230/LIPIcs.APPROX/RANDOM.2021.41. 152

[GGH+07] Shafi Goldwasser, Dan Gutfreund, Alexander Healy, Tali Kaufman, and Guy N. Rothblum. Verifying and decoding in constant depth. In *STOC*, pages 440–449, 2007. doi:10.1145/1250790.1250855. 30

[GGH+16] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. *SIAM J. Comput.*, 45(3):882–929, 2016. doi:10.1137/14095772X. 6

[GGH18] Shafi Goldwasser, Ofer Grossman, and Dhiraj Holden. Pseudo-deterministic proofs. In *ITCS*, pages 17:1–17:18, 2018. doi:10.4230/LIPIcs.ITCS.2018.17. 152

[GGH19] Michel X. Goemans, Shafi Goldwasser, and Dhiraj Holden. Doubly-efficient pseudo-deterministic proofs. *arXiv preprint*, abs/1910.00994, 2019. doi:10.48550/arXiv.1910.00994. 152

[GGM86] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions. *J. ACM*, 33(4):792–807, 1986. doi:10.1145/6490.6503. 206, 214, 215, 299

[GGMW20] Shafi Goldwasser, Ofer Grossman, Sidhanth Mohanty, and David P. Woodruff. Pseudo-deterministic streaming. In *ITCS*, pages 79:1–79:25, 2020. doi:10.4230/LIPIcs.ITCS.2020.79. 152

[GGNS23] Karthik Gajulapalli, Alexander Golovnev, Satyajeet Nagargoje, and Sidhant Saraogi. Range avoidance for constant depth circuits: Hardness and algorithms. In *APPROX/RANDOM*, volume 275 of *LIPIcs*, pages 65:1–65:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPIcs.APPROX/RANDOM.2023.65. 200, 276, 299

[GGR13] Oded Goldreich, Shafi Goldwasser, and Dana Ron. On the possibilities and limitations of pseudodeterministic algorithms. In *ITCS*, pages 127–138, 2013. doi:10.1145/2422436.2422453. 152

[GGRT25] Michal Garlik, Svyatoslav Gryaznov, Hanlin Ren, and Iddo Tzameret. The weak rank principle: Lower bounds and applications. Manuscript, 2025. 8, 283

[GGSW13] Sanjam Garg, Craig Gentry, Amit Sahai, and Brent Waters. Witness encryption and its applications. In *STOC*, pages 467–476. ACM, 2013. doi:10.1145/2488608.2488667. 301

[Gil52] E. N. Gilbert. A comparison of signalling alphabets. *The Bell System Technical Journal*, 31(3):504–522, 1952. doi:10.1002/j.1538-7305.1952.tb01393.x. 2

[GIPS21] Shafi Goldwasser, Russell Impagliazzo, Toniann Pitassi, and Rahul Santhanam. On the pseudo-deterministic query complexity of NP search problems. In *CCC*, pages 36:1–36:22, 2021. doi:10.4230/LIPIcs.CCC.2021.36. 152

[GJS76] M. R. Garey, David S. Johnson, and Larry J. Stockmeyer. Some simplified NP-complete graph problems. *Theor. Comput. Sci.*, 1(3):237–267, 1976. doi:10.1016/0304-3975(76)90059-1. 147

[GKR15] Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. Delegating computation: interactive proofs for muggles. *J. ACM*, 62(4):27:1–27:64, 2015. doi:10.1145/2699436. 160

[GL89] Oded Goldreich and Leonid A. Levin. A hard-core predicate for all one-way functions. In *STOC*, pages 25–32, 1989. doi:10.1145/73007.73010. 177, 179

[GL19] Ofer Grossman and Yang P. Liu. Reproducibility and pseudo-determinism in Log-Space. In *SODA*, pages 606–620, 2019. doi:10.1137/1.9781611975482.38. 152

[GLW22] Venkatesan Guruswami, Xin Lyu, and Xiuhan Wang. Range avoidance for low-depth circuits and connections to pseudorandomness. In *APPROX/RANDOM*, volume 245 of *LIPIcs*, pages 20:1–20:21. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.APPROX/RANDOM.2022.20. 31, 200, 276

[GM84] Shafi Goldwasser and Silvio Micali. Probabilistic encryption. *J. Comput. Syst. Sci.*, 28(2):270–299, 1984. doi:10.1016/0022-0000(84)90070-9. 282

[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*, volume 6650 of *Lecture Notes in Computer Science*, pages 273–301. Springer, 2011. doi:10.1007/978-3-642-22670-0_23. 35

[Gol08] Oded Goldreich. *Computational complexity: a conceptual perspective*. Cambridge University Press, 2008. doi:10.1017/CBO9780511804106. 166, 212

[Gol11a] Oded Goldreich. Candidate one-way functions based on expander graphs. In *Studies in Complexity and Cryptography*, volume 6650 of *Lecture Notes in Computer Science*, pages 76–87. Springer, 2011. doi:10.1007/978-3-642-22670-0_10. 300

[Gol11b] Oded Goldreich. In a world of P = BPP. In *Studies in Complexity and Cryptography*, volume 6650 of *Lecture Notes in Computer Science*, pages 191–232. Springer, 2011. doi:10.1007/978-3-642-22670-0_20. 246

[Gol11c] Oded Goldreich. *A Sample of Samplers: A Computational Perspective on Sampling*, pages 302–332. Springer, Berlin, Heidelberg, 2011. doi:10.1007/978-3-642-22670-0_24. 132, 133

[Gol17] Oded Goldreich. On the doubly-efficient interactive proof systems of GKR. *Electron. Colloquium Comput. Complex.*, TR17-101, 2017. URL: https://eccc.weizmann.ac.il/report/2017/101. 160, 188, 189, 190

[Gol18] Oded Goldreich. On doubly-efficient interactive proof systems. *Foundations and Trends® in Theoretical Computer Science*, 13(3):158–246, 2018. doi:10.1561/0400000084. 188

[Gol25] Oded Goldreich. Multi-pseudodeterministic algorithms. In *Computational Complexity and Local Algorithms*, volume 15700 of *Lecture Notes in Computer Science*, pages 22–43. Springer, 2025. doi:10.1007/978-3-031-88946-2_2. 152

[GR08] Dan Gutfreund and Guy N. Rothblum. The complexity of local list decoding. In *APPROX-RANDOM*, volume 5171 of *Lecture Notes in Computer Science*, pages 455–468. Springer, 2008. doi:10.1007/978-3-540-85363-3_36. 30, 91, 106

[Gro00] Vince Grolmusz. Low rank co-diagonal matrices and ramsey graphs. *Electron. J. Comb.*, 7, 2000. doi:10.37236/1493. 1

[Gro15] Ofer Grossman. Finding primitive roots pseudo-deterministically. *Electron. Colloquium Comput. Complex.*, TR15-207, 2015. URL: https://eccc.weizmann.ac.il/report/2015/207. 152

[GS86] Shafi Goldwasser and Michael Sipser. Private coins versus public coins in interactive proof systems. In *STOC*, pages 59–68. ACM, 1986. doi:10.1145/12130.12137. 282, 284

[GS89] Yuri Gurevich and Saharon Shelah. Nearly linear time. In *Logic at Botik '89, Symposium on Logical Foundations of Computer Science, Pereslav-Zalessky, USSR, July 3-8, 1989, Proceedings*, volume 363 of *Lecture Notes in Computer Science*, pages 108–118, 1989. doi:10.1007/3-540-51237-3_10. 10

[GS92] Peter Gemmell and Madhu Sudan. Highly resilient correctors for polynomials. *Inf. Process. Lett.*, 43(4):169–174, 1992. doi:10.1016/0020-0190(92)90195-2. 178, 221

[GT18] Oded Goldreich and Avishay Tal. Matrix rigidity of random Toeplitz matrices. *Comput. Complex.*, 27(2):305–350, 2018. doi:10.1007/S00037-016-0144-9. 2

[HAB02] William Hesse, Eric Allender, and David A. Mix Barrington. Uniform constant-depth threshold circuits for division and iterated multiplication. *J. Comput. Syst. Sci.*, 65(4):695–716, 2002. doi:10.1016/S0022-0000(02)00025-9. 191, 194

[Hås89] Johan Håstad. Almost optimal lower bounds for small depth circuits. *Adv. Comput. Res.*, 5:143–170, 1989. doi:10.1145/12130.12132. 24

[Hir15] Shuichi Hirahara. Identifying an honest $\mathsf{EXP}^{\mathsf{NP}}$ oracle among many. In *CCC*, volume 33 of *LIPIcs*, pages 244–263. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2015. doi:10.4230/LIPIcs.CCC.2015.244. 211, 222, 240, 255, 256, 257, 258, 259

[Hir18] Shuichi Hirahara. Non-black-box worst-case to average-case reductions within NP. In *FOCS*, pages 247–258, 2018. doi:10.1109/FOCS.2018.00032. 282

[HIR23] Yizhi Huang, Rahul Ilango, and Hanlin Ren. NP-hardness of approximating meta-complexity: A cryptographic approach. In *STOC*, pages 1067–1075. ACM, 2023. doi:10.1145/3564246.3585154. 8

[HLR23] Shuichi Hirahara, Zhenjian Lu, and Hanlin Ren. Bounded relativization. In *CCC*, volume 264 of *LIPIcs*, pages 6:1–6:45. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPIcs.CCC.2023.6. 8, 199, 201, 304

[HMP+93] András Hajnal, Wolfgang Maass, Pavel Pudlák, Márió Szegedy, and György Turán. Threshold circuits of bounded depth. *J. Comput. Syst. Sci.*, 46(2):129–154, 1993. doi:10.1016/0022-0000(93)90001-D. 109

[HNOS96] Lane A. Hemaspaandra, Ashish V. Naik, Mitsunori Ogihara, and Alan L. Selman. Computing solutions uniquely collapses the polynomial hierarchy. *SIAM J. Comput.*, 25(4):697–708, 1996. doi:10.1137/S0097539794268315. 201

[HP10] Kristoffer Arnsfelt Hansen and Vladimir V. Podolskii. Exact threshold circuits. In *CCC*, pages 270–279. IEEE Computer Society, 2010. doi:10.1109/CCC.2010.33. 103, 104

[HV06] Alexander Healy and Emanuele Viola. Constant-depth circuits for arithmetic in finite fields of characteristic two. In *STACS*, volume 3884 of *Lecture Notes in Computer Science*, pages 672–683. Springer, 2006. doi:10.1007/11672142_55. 191, 194, 271, 272

[HV21] Xuangui Huang and Emanuele Viola. Average-case rigidity lower bounds. In *CSR*, volume 12730 of *Lecture Notes in Computer Science*, pages 186–205, 2021. doi:10.1007/978-3-030-79416-3_11. 4, 20

[IKV18] Russell Impagliazzo, Valentine Kabanets, and Ilya Volkovich. The power of natural properties as oracles. In *CCC*, volume 102 of *LIPIcs*, pages 7:1–7:20, 2018. doi:10.4230/LIPIcs.CCC.2018.7. 211, 304

[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. doi:10.1016/S0022-0000(02)00024-7. 85, 155, 204, 205, 236, 304

[Ila25] Rahul Ilango. The oracle derandomization hypothesis is false (and more) assuming no natural proofs. 2025. URL: https://eccc.weizmann.ac.il/report/2025/190. 8

[ILL89] Russell Impagliazzo, Leonid A. Levin, and Michael Luby. Pseudo-random generation from one-way functions. In *STOC*, page 12–24. ACM, 1989. doi:10.1145/73007.73009. 279, 289

[ILW23] Rahul Ilango, Jiatu Li, and R. Ryan Williams. Indistinguishability obfuscation, range avoidance, and bounded arithmetic. In *STOC*, pages 1076–1089. ACM, 2023. doi:10.1145/3564246.3585187. 6, 276, 277, 280, 281, 296, 300, 301

[IM02] Kazuo Iwama and Hiroki Morizumi. An explicit lower bound of 5n - o(n) for boolean circuits. In *MFCS*, volume 2420 of *Lecture Notes in Computer Science*, pages 353–364. Springer, 2002. doi:10.1007/3-540-45687-2_29. 2

[Imp95] Russell Impagliazzo. A personal view of average-case complexity. In *SCT*, pages 134–147. IEEE Computer Society, 1995. doi:10.1109/SCT.1995.514853. 6, 276, 282

[IS20] Dmitry Itsykson and Dmitry Sokolov. Resolution over linear equations modulo two. *Ann. Pure Appl. Log.*, 171(1), 2020. doi:10.1016/J.APAL.2019.102722. 292, 293

[IW97] Russell Impagliazzo and Avi Wigderson. P = BPP if E requires exponential circuits: Derandomizing the XOR lemma. In *STOC*, pages 220–229. ACM, 1997. doi:10.1145/258533.258590. 1, 151, 198, 204, 235, 238

[IW01] Russell Impagliazzo and Avi Wigderson. Randomness vs time: Derandomization under a uniform assumption. *J. Comput. Syst. Sci.*, 63(4):672–688, 2001. doi:10.1006/jcss.2001.1780. 3, 85, 153, 155, 161, 236, 262, 304

[Jeř04] Emil Jeřábek. Dual weak pigeonhole principle, Boolean complexity, and derandomization. *Ann. Pure Appl. Log.*, 129(1-3):1–37, 2004. doi:10.1016/j.apal.2003.12.003. 3, 4, 200, 203, 204, 214, 280, 288

[Jeř05] Emil Jeřábek. *Weak pigeonhole principle and randomized computation.* PhD thesis, Charles University in Prague, 2005. 288

[Jer06] Emil Jerábek. The strength of sharply bounded induction. *Math. Log. Q.*, 52(6):613–624, 2006. doi:10.1002/MALQ.200610019. 287

[Jer07] Emil Jerábek. Approximate counting in bounded arithmetic. *J. Symb. Log.*, 72(3):959–993, 2007. doi:10.2178/JSL/1191333850. 280, 288

[JLS21] Aayush Jain, Huijia Lin, and Amit Sahai. Indistinguishability obfuscation from well-founded assumptions. In *STOC*, pages 60–73. ACM, 2021. doi:10.1145/3406325.3451093. 6, 277

[Jus72] Jørn Justesen. Class of constructive asymptotically good algebraic codes. *IEEE Trans. Inf. Theory*, 18(5):652–656, 1972. doi:10.1109/TIT.1972.1054893. 2

[Kab01] Valentine Kabanets. Easiness assumptions and hardness tests: Trading time for zero error. *J. Comput. Syst. Sci.*, 63(2):236–252, 2001. doi:10.1006/JCSS.2001.1763. 305

[Kan82] Ravi Kannan. Circuit-size lower bounds and non-reducibility to sparse sets. *Inf. Control.*, 55(1-3):40–56, 1982. doi:10.1016/S0019-9958(82)90382-5. 5, 155, 198, 199

[KC00] Valentine Kabanets and Jin-Yi Cai. Circuit minimization problem. In *STOC*, pages 73–79, 2000. doi:10.1145/335305.335314. 199

[Kha22] Erfan Khaniki. Nisan-Wigderson generators in proof complexity: New lower bounds. In *CCC*, volume 234 of *LIPIcs*, pages 17:1–17:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICS.CCC.2022.17. 278

[KKMP21] Robert Kleinberg, Oliver Korten, Daniel Mitropolsky, and Christos Papadimitriou. Total functions in the polynomial hierarchy. In *ITCS*, volume 185 of *LIPIcs*, pages 44:1–44:18, 2021. doi:10.4230/LIPIcs.ITCS.2021.44. 3, 22, 104, 202

[KL80] Richard M. Karp and Richard J. Lipton. Some connections between nonuniform and uniform complexity classes. In *STOC*, pages 302–309, 1980. doi:10.1145/800141.804678. 11, 155, 199, 211, 256

[KM65] Boris M Kloss and Vadim A Malyshev. Estimates of the complexity of certain classes of functions. *Vestn. Moskov. Univ. Ser.*, 20:44–51, 1965. 2

[Kor21] Oliver Korten. The hardest explicit construction. In *FOCS*, pages 433–444. IEEE, 2021. doi:10.1109/FOCS52979.2021.00051. 4, 5, 31, 85, 105, 200, 201, 202, 203, 204, 206, 214, 215, 231, 236, 276

[Kor22] Oliver Korten. Derandomization from time-space tradeoffs. In *CCC*, volume 234 of *LIPIcs*, pages 37:1–37:26. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICS.CCC.2022.37. 4, 259, 261

[KPT91] Jan Krajíček, Pavel Pudlák, and Gaisi Takeuti. Bounded arithmetic and the polynomial hierarchy. *Annals of Pure and Applied Logic*, 52(1):143–153, 1991. doi:10.1016/0168-0072(91)90043-L. 281, 287, 288

[Kra95] Jan Krajíček. *Bounded Arithmetic, Propositional Logic, and Complexity Theory*. Encyclopedia of Mathematics and its Applications. Cambridge University Press, 1995. doi:10.1017/CBO9780511529948. 287

[Kra01a] Jan Krajíček. On the weak pigeonhole principle. *Fundamenta Mathematicae*, 170:123–140, 2001. doi:10.4064/fm170-1-8. 277

[Kra01b] Jan Krajíček. Tautologies from pseudo-random generators. *Bull. Symb. Log.*, 7(2):197–212, 2001. doi:10.2307/2687774. 3, 200, 278, 286

[Kra04] Jan Krajíček. Dual weak pigeonhole principle, pseudo-surjective functions, and provability of circuit lower bounds. *J. Symb. Log.*, 69(1):265–286, 2004. doi:10.2178/jsl/1080938841. 278, 279, 281, 286, 287

[Kra09] Jan Krajíček. A proof complexity generator. In *Proc. from the 13th International Congress of Logic, Methodology and Philosophy of Science (Beijing, August 2007)*, Studies in Logic and the Foundations of Mathematics. King's College Publications, London, 2009. URL: https://www.karlin.mff.cuni.cz/~krajicek/generator.pdf. 283

[Kra11] Jan Krajíček. On the proof complexity of the Nisan-Wigderson generator based on a hard $\mathsf{NP} \cap \mathsf{coNP}$ function. *J. Math. Log.*, 11(1), 2011. doi:10.1142/S0219061311000979. 278

[Kra19] Jan Krajíček. *Proof Complexity*. Encyclopedia of Mathematics and its Applications. Cambridge University Press, 2019. doi:10.1017/9781108242066. 287

[Kra21] Jan Krajíček. Small circuits and dual weak PHP in the universal theory of p-time algorithms. *ACM Trans. Comput. Log.*, 22(2):11:1–11:4, 2021. doi:10.1145/3446207. 281

[Kra23] Jan Krajíček. A proof complexity conjecture and the Incompleteness theorem. *The Journal of Symbolic Logic*, page 1–5, 2023. doi:10.1017/jsl.2023.69. 279

[Kra24] Jan Krajíček. On the existence of strong proof complexity generators. *Bulletin of Symbolic Logic*, 30(1):20–40, 2024. doi:10.1017/bsl.2023.40. 279, 286

[Kra25] Jan Krajíček. *Proof complexity generators*. Cambridge University Press, 2025. doi:10.1017/9781009611664. 6, 278

[Kre88] Mark W. Krentel. The complexity of optimization problems. *J. Comput. Syst. Sci.*, 36(3):490–509, 1988. doi:10.1016/0022-0000(88)90039-6. 213

[KvM02] Adam R. Klivans and Dieter van Melkebeek. Graph nonisomorphism has subexponential size proofs unless the polynomial-time hierarchy collapses. *SIAM J. Comput.*, 31(5):1501–1526, 2002. doi:10.1137/S0097539700389652. 200, 284

[KW98] Johannes Köbler and Osamu Watanabe. New collapse consequences of NP having small circuits. *SIAM J. Comput.*, 28(1):311–324, 1998. doi:10.1137/S0097539795296206. 199

[Lev73] Leonid A. Levin. Universal sequential search problems. *Problemy peredachi informatsii*, 9(3):115–116, 1973. URL: https://mathnet.ru/eng/ppi914. 237, 266

[Lev87] Leonid A. Levin. One-way functions and pseudorandom generators. *Comb.*, 7(4):357–363, 1987. doi:10.1007/BF02579323. 35, 91

[Lew19] Mark Lewko. An explicit two-source extractor with min-entropy rate near 4/9. *Mathematika*, 65(4):950–957, 2019. doi:10.1112/S0025579319000238. 1

[LFKN92] Carsten Lund, Lance Fortnow, Howard J. Karloff, and Noam Nisan. Algebraic methods for interactive proof systems. *Journal of the ACM*, 39(4):859–868, 1992. doi:10.1145/146585.146605. 211

[Li12] Xin Li. Non-malleable extractors, two-source extractors and privacy amplification. In *FOCS*, pages 688–697. IEEE Computer Society, 2012. doi:10.1109/FOCS.2012.26. 1

[Li16] Xin Li. Improved two-source extractors, and affine extractors for polylogarithmic entropy. In *FOCS*, pages 168–177. IEEE Computer Society, 2016. doi:10.1109/FOCS.2016.26. 1

[Li17] Xin Li. Improved non-malleable extractors, non-malleable codes and independent source extractors. In *STOC*, pages 1144–1156. ACM, 2017. doi:10.1145/3055399.3055486. 1

[Li19] Xin Li. Non-malleable extractors and non-malleable codes: Partially optimal constructions. In *CCC*, volume 137 of *LIPIcs*, pages 28:1–28:49. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPICS.CCC.2019.28. 1

[Li23] Xin Li. Two source extractors for asymptotically optimal entropy, and (many) more. In *FOCS*, pages 1271–1281. IEEE, 2023. doi:10.1109/FOCS57990.2023.00075. 1, 200

[Li24] Zeyong Li. Symmetric exponential time requires near-maximum circuit size: Simplified, truly uniform. In *STOC*, pages 2000–2007. ACM, 2024. doi:10.1145/3618260.3649615. 7, 155

[Li25] Jiatu Li. An introduction to feasible mathematics and bounded arithmetic for computer scientists. *Electron. Colloquium Comput. Complex.*, TR25-086, 2025. URL: https://eccc.weizmann.ac.il/report/2025/086. 287

[LLR24] Jiawei Li, Yuhao Li, and Hanlin Ren. Finding bugs in short proofs: The metamathematics of resolution lower bounds. *arXiv preprint*, abs/2411.15515, 2024. doi:10.48550/arXiv.2411.15515. iii, 8

[LO87] J. C. Lagarias and Andrew M. Odlyzko. Computing $\pi(x)$: An analytic method. *J. Algorithms*, 8(2):173–191, 1987. doi:10.1016/0196-6774(87)90037-X. 1, 151

[LO22] Zhenjian Lu and Igor C. Oliveira. Theory and applications of probabilistic Kolmogorov complexity. *Bull. EATCS*, 137, 2022. URL: http://bulletin.eatcs.org/index.php/beatcs/article/view/700. 153, 245, 270

[Lok01] Satyanarayana V. Lokam. Spectral methods for matrix rigidity with applications to size-depth trade-offs and communication complexity. *J. Comput. Syst. Sci.*, 63(3):449–473, 2001. doi:10.1006/JCSS.2001.1786. 2

[Lok09] Satyanarayana V. Lokam. Complexity lower bounds using linear algebra. *Found. Trends Theor. Comput. Sci.*, 4(1-2):1–155, 2009. doi:10.1561/0400000011. 2

[LORS24] Zhenjian Lu, Igor C. Oliveira, Hanlin Ren, and Rahul Santhanam. On the complexity of avoiding heavy elements. In *FOCS*, pages 2403–2412. IEEE, 2024. doi:10.1109/FOCS61266.2024.00140. 7

[LOS21] Zhenjian Lu, Igor C. Oliveira, and Rahul Santhanam. Pseudodeterministic algorithms and the structure of probabilistic time. In *STOC*, pages 303–316, 2021. doi:10.1145/3406325.3451085. 152, 153

[LP22] Yanyi Liu and Rafael Pass. Characterizing derandomization through hardness of Levin-Kolmogorov complexity. In *CCC*, pages 35:1–35:17, 2022. doi:10.4230/LIPICS.CCC.2022.35. 238, 246, 259

[LP23] Yanyi Liu and Rafael Pass. Leakage-resilient hardness vs randomness. In *CCC*, volume 264 of *LIPIcs*, pages 32:1–32:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023. 6, 237, 238, 242, 246, 259, 263, 264, 267

[LR01] Oded Lachish and Ran Raz. Explicit lower bound of $4.5n - o(n)$ for Boolean circuits. In *STOC*, pages 399–408. ACM, 2001. doi:10.1145/380752.380832. 2

[Lup58] Oleg B Lupanov. The synthesis of contact circuits. *Doklady Akademii Nauk SSSR*, 119(1):23–26, 1958. URL: https://www.mathnet.ru/eng/dan22802. 198

[LV19] Ming Li and Paul M. B. Vitányi. *An Introduction to Kolmogorov Complexity and Its Applications, 4th Edition*. Texts in Computer Science. Springer, 2019. doi:10.1007/978-3-030-11298-1. 245

[LY22] Jiatu Li and Tianqi Yang. $3.1n - o(n)$ circuit lower bounds for explicit functions. In *STOC*, pages 1180–1193. ACM, 2022. doi:10.1145/3519935.3519976. 2, 151

[Mek17] Raghu Meka. Explicit resilient functions matching Ajtai-Linial. In *SODA*, pages 1132–1148. SIAM, 2017. doi:10.1137/1.9781611974782.73. 1

[Mie09] Thilo Mie. Short PCPPs verifiable in polylogarithmic time with $O(1)$ queries. *Ann. Math. Artif. Intell.*, 56(3-4):313–338, 2009. doi:10.1007/s10472-009-9169-y. 115, 140

[MP24] Noam Mazor and Rafael Pass. Gap MCSP is not (Levin) NP-complete in Obfustopia. In *CCC*, volume 300 of *LIPIcs*, pages 36:1–36:21. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPICS.CCC.2024.36. 266

[MST06] Elchanan Mossel, Amir Shpilka, and Luca Trevisan. On ε-biased generators in NC^0. *Random Struct. Algorithms*, 29(1):56–81, 2006. doi:10.1002/RSA.20112. 300

[Mur71] Saburo Muroga. *Threshold logic and its applications*. Wiley, 1971. 104

[MV05] Peter Bro Miltersen and N. V. Vinodchandran. Derandomizing Arthur-Merlin games using hitting sets. *Comput. Complex.*, 14(3):256–279, 2005. doi:10.1007/S00037-005-0197-7. 284

[MVW99] Peter Bro Miltersen, N. V. Vinodchandran, and Osamu Watanabe. Super-polynomial versus half-exponential circuit size in the exponential hierarchy. In *COCOON*, volume 1627 of *Lecture Notes in Computer Science*, pages 210–220. Springer, 1999. doi:10.1007/3-540-48686-0_21. 5, 155, 199, 202

[MW20] Cody D. Murray and R. Ryan Williams. Circuit lower bounds for nondeterministic quasi-polytime from a new easy witness lemma. *SIAM J. Comput.*, 49(5), 2020. doi:10.1137/18M1195887. 19, 304

[NN93] Joseph Naor and Moni Naor. Small-bias probability spaces: Efficient constructions and applications. *SIAM J. Comput.*, 22(4):838–856, 1993. doi:10.1137/0222053. 2

[NW94] Noam Nisan and Avi Wigderson. Hardness vs randomness. *J. Comput. Syst. Sci.*, 49(2):149–167, 1994. doi:10.1016/S0022-0000(05)80043-1. 86, 110, 111, 156, 161, 198, 204, 235, 238, 278

[Oli19] Igor C. Oliveira. Randomness and intractability in Kolmogorov complexity. In *ICALP*, pages 32:1–32:14, 2019. doi:10.4230/LIPIcs.ICALP.2019.32. 152

[OS17a] Igor C. Oliveira and Rahul Santhanam. Conspiracies between learning algorithms, circuit lower bounds, and pseudorandomness. In *CCC*, volume 79 of *LIPIcs*, pages 18:1–18:49, 2017. doi:10.4230/LIPIcs.CCC.2017.18. 86

[OS17b] Igor C. Oliveira and Rahul Santhanam. Pseudodeterministic constructions in subexponential time. In *STOC*, pages 665–677, 2017. doi:10.1145/3055399.3055500. 2, 3, 5, 152, 153, 154, 156, 157, 158, 159, 164

[OS18] Igor C. Oliveira and Rahul Santhanam. Pseudo-derandomizing learning and approximation. In *RANDOM*, pages 55:1–55:19, 2018. doi:10.4230/LIPIcs.APPROX-RANDOM.2018.55. 152

[Pan21] Shuo Pang. Large clique is hard on average for resolution. In *CSR*, volume 12730 of *Lecture Notes in Computer Science*, pages 361–380. Springer, 2021. doi:10.1007/978-3-030-79416-3_22. 282

[Par21] Orr Paradise. Smooth and strong PCPs. *Comput. Complex.*, 30(1):1, 2021. doi:10.1007/S00037-020-00199-3. 15, 31

[Păt08] Mihai Pătraşcu. Succincter. In *FOCS*, pages 305–313, 2008. doi:10.1109/FOCS.2008.83. 31

[Pau77] Wolfgang J. Paul. A $2.5n$-lower bound on the combinational complexity of Boolean functions. *SIAM J. Comput.*, 6(3):427–443, 1977. doi:10.1137/0206030. 2

[Pic11] Ján Pich. Nisan-Wigderson generators in proof systems with forms of interpolation. *Math. Log. Q.*, 57(4):379–383, 2011. doi:10.1002/MALQ.201010012. 278

[Pic15] Ján Pich. Circuit lower bounds in bounded arithmetics. *Ann. Pure Appl. Log.*, 166(1):29–45, 2015. doi:10.1016/J.APAL.2014.08.004. 278

[Pip79] Nicholas Pippenger. On simultaneous resource bounds (preliminary version). In *FOCS*, pages 307–311. IEEE Computer Society, 1979. doi:10.1109/SFCS.1979.29. 11, 24

[PS94] Ramamohan Paturi and Michael E. Saks. Approximating threshold circuits by rational functions. *Inf. Comput.*, 112(2):257–272, 1994. doi:10.1006/inco.1994.1059. 104

[PS19] Ján Pich and Rahul Santhanam. Why are proof complexity lower bounds hard? In *FOCS*, pages 1305–1324. IEEE Computer Society, 2019. doi:10.1109/FOCS.2019.00080. 278, 298

[PS22] Ján Pich and Rahul Santhanam. Learning algorithms versus automatability of Frege systems. In *ICALP*, volume 229 of *LIPIcs*, pages 101:1–101:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICS.ICALP.2022.101. 278

[Pud94] Pavel Pudlák. Communication in bounded depth circuits. *Comb.*, 14(2):203–216, 1994. doi:10.1007/BF01215351. 2

[PWW88] Jeff B. Paris, A. J. Wilkie, and Alan R. Woods. Provability of the pigeonhole principle and the existence of infinitely many primes. *J. Symb. Log.*, 53(4):1235–1244, 1988. doi:10.1017/S0022481200028061. 4

[Ram20] C. Ramya. Recent progress on matrix rigidity - A survey. *arXiv preprint*, abs/2009.09460, 2020. doi:10.48550/arXiv.2009.09460. 2

[Raz89] Alexander A. Razborov. On rigid matrices (in Russian). Technical report, 1989. URL: https://people.cs.uchicago.edu/~razborov/files/rigid.pdf. 2

[Raz98] Alexander A. Razborov. Lower bounds for the polynomial calculus. *Comput. Complex.*, 7(4):291–324, 1998. doi:10.1007/s000370050013. 278

[Raz04] Alexander A. Razborov. Resolution lower bounds for perfect matching principles. *J. Comput. Syst. Sci.*, 69(1):3–27, 2004. doi:10.1016/J.JCSS.2004.01.004. 278

[Raz05] Ran Raz. Extractors with weak random seeds. In *STOC*, pages 11–20. ACM, 2005. doi:10.1145/1060590.1060593. 1

[Raz15] Alexander Razborov. Pseudorandom generators hard for k-DNF resolution and polynomial calculus resolution. *Annals of Mathematics*, 181(2):415–472, 2015. doi:10.4007/annals.2015.181.2.1. 278, 279

[RR97] Alexander A. Razborov and Steven Rudich. Natural proofs. *J. Comput. Syst. Sci.*, 55(1):24–35, 1997. doi:10.1006/jcss.1997.1494. 86

[RS98] Alexander Russell and Ravi Sundaram. Symmetric alternation captures BPP. *Comput. Complex.*, 7(2):152–162, 1998. doi:10.1007/s000370050007. 200, 209, 212, 227, 230

[RSW22] Hanlin Ren, Rahul Santhanam, and Zhikun Wang. On the range avoidance problem for circuits. In *FOCS*, pages 640–650. IEEE, 2022. doi:10.1109/FOCS54457.2022.00067. 6, 7, 200, 201, 236, 276, 286

[Rud97] Steven Rudich. Super-bits, demi-bits, and $\tilde{\mathsf{NP}}$/*qpoly*-natural proofs. In *RANDOM*, volume 1269 of *Lecture Notes in Computer Science*, pages 85–93, 1997. doi:10.1007/3-540-63248-4_8. 6, 277, 279, 298, 299

[RVW02] Omer Reingold, Salil Vadhan, and Avi Wigderson. Entropy waves, the zig-zag graph product, and new constant-degree expanders. *Annals of Mathematics*, 155:157–187, 2002. doi:10.2307/3062153. 2

[RWZ26] Hanlin Ren, Yichuan Wang, and Yan Zhong. Hardness of range avoidance and proof complexity generators from demi-bits. In *ITCS*, 2026. To appear. doi:10.48550/arXiv.2511.14061. iii, 7

[San09] Rahul Santhanam. Circuit lower bounds for Merlin–Arthur classes. *SIAM J. Comput.*, 39(3):1038–1061, 2009. doi:10.1137/070702680. 199, 304

[Sch74] Claus-Peter Schnorr. Zwei lineare untere Schranken für die Komplexität Boolescher Funktionen. *Computing*, 13(2):155–171, 1974. doi:10.1007/BF02246615. 2

[Sch76] Claus-Peter Schnorr. The combinational complexity of equivalence. *Theor. Comput. Sci.*, 1(4):289–295, 1976. doi:10.1016/0304-3975(76)90073-6. 2

[Sel94] Alan L. Selman. A taxonomy of complexity classes of functions. *J. Comput. Syst. Sci.*, 48(2):357–381, 1994. doi:10.1016/S0022-0000(05)80009-1. 201

[Sha49] Claude E. Shannon. The synthesis of two-terminal switching circuits. *Bell System technical journal*, 28(1):59–98, 1949. doi:10.1002/j.1538-7305.1949.tb03624.x. 2, 198, 201

[Sho92] Victor Shoup. Searching for primitive roots in finite fields. *Mathematics of Computation*, 58(197):369–380, January 1992. doi:10.1090/S0025-5718-1992-1106981-9. 163, 176

[Spi96] Daniel A. Spielman. Linear-time encodable and decodable error-correcting codes. *IEEE Transactions on Information Theory*, 42(6):1723–1731, 1996. doi:10.1109/18.556668. 11

[SSS97] Mohammad Amin Shokrollahi, Daniel A. Spielman, and Volker Stemann. A remark on matrix rigidity. *Inf. Process. Lett.*, 64(6):283–285, 1997. doi:10.1016/S0020-0190(97)00190-7. 2

[ST21] Rahul Santhanam and Iddo Tzameret. Iterated lower bound formulas: a diagonalization-based approach to proof complexity. In *STOC*, pages 234–247. ACM, 2021. doi:10.1145/3406325.3451010. 278, 298

[Sto77] Larry J. Stockmeyer. On the combinational complexity of certain symmetric boolean functions. *Math. Syst. Theory*, 10:323–336, 1977. doi:10.1007/BF01683282. 2

[STV01] Madhu Sudan, Luca Trevisan, and Salil P. Vadhan. Pseudorandom generators without the XOR lemma. *J. Comput. Syst. Sci.*, 62(2):236–266, 2001. doi:10.1006/jcss.2000.1730. 161, 238, 263

[SU05] Ronen Shaltiel and Christopher Umans. Simple extractors for all min-entropies and a new pseudorandom generator. *J. ACM*, 52(2):172–216, 2005. doi:10.1145/1059513.1059516. 156, 161, 162, 169, 178, 179, 180, 181, 284

[SU06] Ronen Shaltiel and Christopher Umans. Pseudorandomness for approximate counting and sampling. *Comput. Complex.*, 15(4):298–341, 2006. doi:10.1007/S00037-007-0218-9. 284

[Sud95] Madhu Sudan. *Efficient Checking of Polynomials and Proofs anf the Hardness of Approximation Problems*, volume 1001 of *Lecture Notes in Computer Science*. Springer, 1995. doi:10.1007/3-540-60615-7. 178, 221

[Sud97] Madhu Sudan. Decoding of Reed Solomon codes beyond the error-correction bound. *J. Complex.*, 13(1):180–193, 1997. doi:10.1006/jcom.1997.0439. 175

[SW13] Rahul Santhanam and R. Ryan Williams. On medium-uniformity and circuit lower bounds. In *CCC*, pages 15–23. IEEE Computer Society, 2013. doi:10.1109/CCC.2013.40. 19

[Ta-17] Amnon Ta-Shma. Explicit, almost optimal, epsilon-balanced codes. In *STOC*, pages 238–251. ACM, 2017. doi:10.1145/3055399.3055408. 2

[TCH12] Terence Tao, Ernest Croot, III, and Harald Helfgott. Deterministic methods to find primes. *Math. Comput.*, 81(278):1233–1246, 2012. doi:10.1090/S0025-5718-2011-02542-1. 1, 151

[TV07] Luca Trevisan and Salil P. Vadhan. Pseudorandomness and average-case complexity via uniform reductions. *Computational Complexity*, 16(4):331–364, 2007. doi:10.1007/s00037-007-0233-x. 3, 153, 154, 155, 247

[TZ24] Iddo Tzameret and Luming Zhang. Stretching demi-bits and nondeterministic-secure pseudorandomness. In *ITCS*, volume 287 of *LIPIcs*, pages 95:1–95:22. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPICS.ITCS.2024.95. 277

[Vad12] Salil P. Vadhan. Pseudorandomness. *Foundations and Trends in Theoretical Computer Science*, 7(1-3):1–336, 2012. doi:10.1561/0400000010. 137

[Val77] Leslie G. Valiant. Graph-theoretic arguments in low-level complexity. In *MFCS*, volume 53 of *Lecture Notes in Computer Science*, pages 162–176, 1977. doi:10.1007/3-540-08353-7_135. 2, 200

[Var64] R. Varshamov. Estimate of the number of signals in error correcting codes. *Avtomat. i Telemekh*, 25:1628–1629, 1964. URL: https://mathnet.ru/eng/at11783. 2

[Vin05] N. V. Vinodchandran. A note on the circuit complexity of PP. *Theor. Comput. Sci.*, 347(1-2):415–418, 2005. doi:10.1016/j.tcs.2005.07.032. 199

[Vio09] Emanuele Viola. On approximate majority and probabilistic time. *Comput. Complex.*, 18(3):337–375, 2009. doi:10.1007/S00037-009-0267-3. 249, 252, 253

[Vio20] Emanuele Viola. New lower bounds for probabilistic degree and AC^0 with parity gates. *Electron. Colloquium Comput. Complex.*, page 15, 2020. URL: https://eccc.weizmann.ac.il/report/2020/015. 19

[VK21] Ben Lee Volk and Mrinal Kumar. Lower bounds for matrix factorization. *Comput. Complex.*, 30(1):6, 2021. doi:10.1007/S00037-021-00205-2. 2

[VL99] Jacobus Hendricus Van Lint. *Introduction to coding theory*, volume 86. Springer-Verlag Berlin Heidelberg, 1999. doi:10.1007/978-3-642-58575-3. 165

[VV86] Leslie G. Valiant and Vijay V. Vazirani. NP is as easy as detecting unique solutions. *Theor. Comput. Sci.*, 47(3):85–93, 1986. doi:10.1016/0304-3975(86)90135-0. 240, 252, 254

[VW18] Emanuele Viola and Avi Wigderson. Local expanders. *Computational Complexity*, 27(2):225–244, 2018. doi:10.1007/s00037-017-0155-1. 136

[VW20] Nikhil Vyas and R. Ryan Williams. Lower bounds against sparse symmetric functions of ACC circuits: Expanding the reach of #SAT algorithms. In *STACS*, volume 154 of *LIPIcs*, pages 59:1–59:17, 2020. doi:10.4230/LIPIcs.STACS.2020.59. 19

[VW23] Nikhil Vyas and Ryan Williams. On oracles and algorithmic methods for proving lower bounds. In *ITCS*, volume 251 of *LIPIcs*, pages 99:1–99:26. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPIcs.ITCS.2023.99. 199

[WDP+22] Jason Vander Woude, Peter Dixon, A. Pavan, Jamie Radcliffe, and N. V. Vinodchandran. The geometry of rounding. *Electron. Colloquium Comput. Complex.*, TR22-160, 2022. URL: https://eccc.weizmann.ac.il/report/2022/160. 152

[Wil13a] R. Ryan Williams. Improving exhaustive search implies superpolynomial lower bounds. *SIAM Journal of Computing*, 42(3):1218–1244, 2013. doi:10.1137/10080703X. 3, 19, 86, 103

[Wil13b] R. Ryan Williams. Towards NEXP versus BPP? In *CSR*, volume 7913 of *Lecture Notes in Computer Science*, pages 174–182. Springer, 2013. doi:10.1007/978-3-642-38536-0_15. 235

[Wil14] R. Ryan Williams. Nonuniform ACC circuit lower bounds. *J. ACM*, 61(1):2:1–2:32, 2014. doi:10.1145/2559903. 3, 19, 79, 235, 304

[Wil16] R. Ryan Williams. Natural proofs versus derandomization. *SIAM Journal of Computing*, 45(2):497–529, 2016. doi:10.1137/130938219. 19, 85, 236, 305

[Wil18a] R. Ryan Williams. Faster all-pairs shortest paths via circuit complexity. *SIAM J. Comput.*, 47(5):1965–1985, 2018. doi:10.1137/15M1024524. 79

[Wil18b] R. Ryan Williams. Limits on representing Boolean functions by linear combinations of simple functions: Thresholds, ReLUs, and low-degree polynomials. In *CCC*, volume 102 of *LIPIcs*, pages 6:1–6:24, 2018. doi:10.4230/LIPIcs.CCC.2018.6. 19, 49, 54, 87, 89

[Wil18c] R. Ryan Williams. New algorithms and lower bounds for circuits with linear threshold gates. *Theory Comput.*, 14(1):1–25, 2018. doi:10.4086/toc.2018.v014a017. 5, 19, 25, 80

[Wil19] R. Ryan Williams. Some estimated likelihoods for computational complexity. In *Computing and Software Science*, volume 10000 of *Lecture Notes in Computer Science*, pages 9–26. Springer, 2019. doi:10.1007/978-3-319-91908-9_2. 235

[Wun12] Henning Wunderlich. On a theorem of Razborov. *Comput. Complex.*, 21(3):431–477, 2012. doi:10.1007/S00037-011-0021-5. 2

[Yao82] Andrew Chi-Chih Yao. Theory and applications of trapdoor functions (extended abstract). In *FOCS*, pages 80–91, 1982. doi:10.1109/SFCS.1982.45. 164, 177, 278

[Yao85] Andrew Chi-Chih Yao. Separating the polynomial-time hierarchy by oracles (preliminary version). In *FOCS*, pages 1–10. IEEE Computer Society, 1985. doi:10.1109/SFCS.1985.49. 24

[Zák83] Stanislav Zák. A Turing machine time hierarchy. *Theor. Comput. Sci.*, 26:327–333, 1983. doi:10.1016/0304-3975(83)90015-4. 26, 28

[Zwi91] Uri Zwick. A $4n$ lower bound on the combinational complexity of certain symmetric Boolean functions over the basis of unate dyadic Boolean functions. *SIAM J. Comput.*, 20(3):499–505, 1991. doi:10.1137/0220032. 2