Electronic Colloquium on Computational Complexity, Report No. 144 (2023)

Symmetric Exponential Time Requires Near-Maximum Circuit Size

Lijie Chen
UC Berkeley
lijiechen@berkeley.edu

Shuichi Hirahara
National Institute of Informatics
s_hirahara@nii.ac.jp

Hanlin Ren
University of Oxford
hanlin.ren@cs.ox.ac.uk

September 22, 2023

Abstract

We show that there is a language in $\mathsf{S}_2\mathsf{E}/_1$ (symmetric exponential time with one bit of advice) with circuit complexity at least $2^n/n$. In particular, the above also implies the same near-maximum circuit lower bounds for the classes $\Sigma_2\mathsf{E}$, $(\Sigma_2\mathsf{E} \cap \Pi_2\mathsf{E})/_1$, and $\mathsf{ZPE}^{\mathsf{NP}}/_1$. Previously, only "half-exponential" circuit lower bounds for these complexity classes were known, and the smallest complexity class known to require exponential circuit complexity was $\Delta_3\mathsf{E} = \mathsf{E}^{\Sigma_2\mathsf{P}}$ (Miltersen, Vinodchandran, and Watanabe COCOON'99).

Our circuit lower bounds are corollaries of an unconditional zero-error pseudodeterministic algorithm with an NP oracle and one bit of advice ($\mathsf{FZPP}^{\mathsf{NP}}/_1$) that solves the range avoidance problem infinitely often. This algorithm also implies unconditional infinitely-often pseudodeterministic $\mathsf{FZPP}^{\mathsf{NP}}/_1$ constructions for Ramsey graphs, rigid matrices, two-source extractors, linear codes, and $\mathrm{K}^{\mathrm{poly}}$-random strings with nearly optimal parameters.

Our proofs relativize. The two main technical ingredients are (1) Korten's P^{NP} reduction from the range avoidance problem to constructing hard truth tables (FOCS'21), which was in turn inspired by a result of Jeřábek on provability in Bounded Arithmetic (Ann. Pure Appl. Log. 2004); and (2) the recent iterative win-win paradigm of Chen, Lu, Oliveira, Ren, and Santhanam (FOCS'23).

Contents

ISSN 1433-8092

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 have $\mathrm{pr}\mathsf{P} = \mathrm{pr}\mathsf{BPP}$ [NW94, IW97], while super-polynomial lower bounds such as $\mathsf{EXP} \not\subset \mathsf{P}/_{\mathrm{poly}}$ only imply sub-exponential time derandomization of $\mathrm{pr}\mathsf{BPP}$.[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 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] emphasized our lack of knowledge regarding the circuit complexity of $\Sigma_2\mathsf{EXP}$, even with respect to *relativizing* 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 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 relativization 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], $\mathsf{S_2EXP}$ [CCHO05, Cai07], PEXP [Vin05, Aar06], and MAEXP [BFT98, San09].

[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, in this paper, 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 paper 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.

[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.

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 1.4.1 for a detailed discussion.

1.1 Our Results

1.1.1 New near-maximum circuit lower bounds

In this work, we *overcome* the half-exponential barrier mentioned above and resolve Open Problem 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 *relativizes*:

Theorem 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 relativized 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 2.1 for a formal definition), again with a relativizing proof:

Theorem 1.3. $\mathsf{S}_2\mathsf{E}/_1 \not\subset \mathsf{SIZE}[2^n/n]$. *Moreover, this holds in every relativized world.*

The symmetric time class $\mathsf{S}_2\mathsf{E}$. $\mathsf{S}_2\mathsf{E}$ can be seen as a "randomized" version of E^{NP} since it is sandwiched between E^{NP} and $\mathsf{ZPE}^{\mathsf{NP}}$: it is easy to show that $\mathsf{E}^{\mathsf{NP}} \subseteq \mathsf{S}_2\mathsf{E}$ [RS98], and it is also known that $\mathsf{S}_2\mathsf{E} \subseteq \mathsf{ZPE}^{\mathsf{NP}}$ [Cai07]. We also note that under plausible derandomization assumptions (e.g., E^{NP} requires $2^{\Omega(n)}$-size SAT-oracle circuits), all three classes simply collapse to E^{NP} [KvM02].

Hence, our results also imply a near-maximum circuit lower bound for the class $\mathsf{ZPE}^{\mathsf{NP}}/_1 \subseteq (\Sigma_2\mathsf{E} \cap \Pi_2\mathsf{E})/_1$. This vastly improves the previous lower bound for $\Delta_3\mathsf{E} = \mathsf{E}^{\Sigma_2\mathsf{P}}$.

Corollary 1.4. $\mathsf{ZPE}^{\mathsf{NP}}/_1 \not\subset \mathsf{SIZE}[2^n/n]$. *Moreover, this holds in every relativized world.*

1.1.2 New algorithms for the range avoidance problem

Background on Avoid. Actually, our circuit lower bounds are implied by our new algorithms for solving the range avoidance problem (Avoid) [KKMP21, Kor21, RSW22], which is defined as follows: given a circuit $C : \{0,1\}^n \to \{0,1\}^{n+1}$ as input, find a string outside the range of C (we define $\mathrm{Range}(C) \coloneqq \{C(z) : z \in \{0,1\}^n\}$). That is, output any string $y \in \{0,1\}^{n+1}$ such that for every $x \in \{0,1\}^n$, $C(x) \neq y$.

There is a trivial $\mathsf{FZPP}^{\mathsf{NP}}$ 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 $y \notin \mathrm{Range}(C)$). The class APEPP (Abundant Polynomial Empty Pigeonhole Principle) [KKMP21] is the class of total search problems reducible to Avoid.

As demonstrated by Korten [Kor21, Section 3], APEPP captures the complexity of explicit construction problems whose solutions are guaranteed to exist by the probabilistic method (more precisely, the dual weak pigeonhole principle [Kra01, Jeř04]), in the sense that constructing such objects reduces to the range avoidance problem. 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, derandomizing the trivial $\mathsf{FZPP}^{\mathsf{NP}}$ 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 $\mathsf{FZPP}^{\mathsf{NP}}$ algorithm for Avoid can be made *pseudodeterministic* on infinitely many input lengths. A *pseudodeterministic* algorithm [GG11] is a randomized algorithm that outputs the same *canonical* answer on most computational paths. In particular, we have:

Theorem 1.5. *For every constant $d \geq 1$, there is a randomized 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 relativized world.*

As a corollary, for every problem in APEPP, we obtain zero-error pseudodeterministic constructions with an NP oracle and one bit of advice ($\mathsf{FZPP}^{\mathsf{NP}}/_1$) that works infinitely often[4]:

Corollary 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.*

Actually, we obtain single-valued $\mathsf{FS_2P}/_1$ algorithms for the explicit construction problems above (see Definition 2.2), and the pseudodeterministic $\mathsf{FZPP}^{\mathsf{NP}}/_1$ algorithms follow from Cai's theorem that $\mathsf{S_2P} \subseteq \mathsf{ZPP}^{\mathsf{NP}}$ [Cai07]. We stated them as pseudodeterministic $\mathsf{FZPP}^{\mathsf{NP}}/_1$ algorithms since this notion is better known than the notion of single-valued $\mathsf{FS_2P}/_1$ algorithms.

Theorem 1.5 is tantalizingly close to an infinitely-often $\mathsf{FP}^{\mathsf{NP}}$ algorithm for Avoid (with the only caveat of being *zero-error* instead of being completely *deterministic*). However, since an $\mathsf{FP}^{\mathsf{NP}}$ algorithm for range avoidance would imply near-maximum circuit lower bounds for E^{NP}, we expect that it would require fundamentally new ideas to completely derandomize our algorithm. Previously, Hirahara, Lu, and Ren [HLR23, Theorem 36] presented an infinitely-often pseudodeterministic $\mathsf{FZPP}^{\mathsf{NP}}$ 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 relativize 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 derandomization [CT21b].

Corollary 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 $\mathsf{ZPE}^{\mathsf{NP}}/_1$ in place of $\mathsf{S_2E}/_1$. Moreover, this holds in every relativized world.*

1.2 Intuitions

In the following, we present some high-level intuitions for our new circuit lower bounds.

[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).

1.2.1 Perspective: single-valued constructions

A key perspective in this paper 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 emphasized 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 $f : \{0,1\}^n \to \{0,1\}$ 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 summarized 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 1.1.2).

In the rest of Section 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 $|C_n| \leq \mathrm{poly}(n)$.[7] Our goal is to find an algorithm A such that for infinitely many n, $A(1^n) \in \{0,1\}^{2n} \setminus \mathrm{Range}(C_n)$; see Section 5.3 and Section 5.4 for how to turn

[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

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.

1.2.2 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]

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 (randomized) 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 $\mathrm{poly}(n_{i+1}) \le \mathrm{poly}(n_i)$ 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 paper, we will implement a similar mechanism using Korten's P^{NP} reduction from the range avoidance problem to constructing hard truth tables [Kor21].

Getting polynomial time. Now we briefly explain why (Win-or-Improve) implies a *polynomial-time* construction algorithm. Let α be an absolute constant such that we always have $T_{i+1} \le T_i^\alpha$; we now set $\beta := 2\alpha$. Recall that $n_i = n_{i-1}^\beta$ for every i. The crucial observation is the following:

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 $1/\mathrm{poly}(n)$-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] and the prime number theorem, the set of n-bit primes is such a property.

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 $T_t \le \mathrm{poly}(n_t)$; see [CLO+23, Section 1.3.1].

For each $0 \le i < t$, if $R(1^{n_i})$ successfully simulates ALG_i, then we obtain an algorithm for input length n_i running in $\mathrm{poly}(n_{i+1}) \le \mathrm{poly}(n_i)$ 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 $T_t \le \mathrm{poly}(n_t)$, 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 $n_0, n_1, \ldots, n_t$.

1.2.3 Algorithms for range-avoidance via Korten's 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 Korten's reduction [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 1.3.2). 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.

Korten's reduction. We will apply Korten's recent work [Kor21] to make use of the "gain" above. So it is worth taking a detour to review the main result of [Kor21]. Roughly speaking, [Kor21] gives **an algorithm that uses a hard truth table f to solve a derandomization task: finding a non-output of the given circuit (that has more output bits than input bits).**[10]

Formally, [Kor21] gives a P^{NP}-computable algorithm $\mathsf{Korten}(C, f)$ 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 $m = n^\beta$, $\mathsf{Korten}(C_m, h_{\mathsf{BF}})$ solves AVOID for C_m since the circuit complexity of h_{BF}, n^α, is sufficiently larger than the size of C_m. Moreover, $\mathsf{Korten}(C_m, h_{\mathsf{BF}})$ 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.

[9] Korten's result was inspired by [Jeř04], which 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 derandomization task: estimating the acceptance probability of the given circuit. This explains why one may want to use Korten's algorithm to replace the Chen–Tell targeted generator construction [CT21a] from [CLO+23], as they are both hardness-vs-randomness connections.

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 1.2.2. Therefore, we only need to iterate the argument above to obtain a polynomial-time algorithm.

For this purpose, we need to consider the computational history of not only BF, but also algorithms of the form $\mathsf{Korten}(C, f)$.[11] For any circuit C and "hard" truth table f, there is a *unique* "computational history" h of $\mathsf{Korten}(C, f)$, 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 $\mathsf{Korten}(C, f)$ can be simulated by a single-valued $\mathsf{F\Sigma_2P}$ algorithm in time $\mathrm{poly}(s)$; see Section 1.3.2 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 $\mathsf{Korten}(C_{n_i}, -)$ 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 $\mathsf{ALG}_i := \mathsf{Korten}(C_{n_i}, f_i)$.

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 $\mathsf{Korten}(C_{n_0}, f_0)$ is equivalent to $\mathsf{BF} = \mathsf{ALG}_0$; see Fact 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 Korten's reduction to solve AVOID for $C_{n_{i+1}}$. That is, we take $\mathsf{ALG}_{i+1} := \mathsf{Korten}(C_{n_{i+1}}, f_{i+1})$ 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).

We note that $\mathsf{Korten}(\mathsf{TT}_{n,2n}, f)$ 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.

[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 $\mathsf{Korten}(C, f)$ (including ALG_0), so in what follows we only consider the computational history of $\mathsf{Korten}(C, f)$.

[12] With an "encoded" version of history and more effort, we are able to simulate $\mathsf{Korten}(C, f)$ 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 1.3.3 for details.

[13] A hardness condenser takes a long truth table f with certain hardness and outputs a shorter truth table with similar hardness.

At the stage i, we consider the algorithm $\mathsf{ALG}_i := \mathsf{Korten}(\mathsf{TT}_{n_i,2n_i}, f_i)$, 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 $\mathsf{ALG}_{i+1} := \mathsf{Korten}(\mathsf{TT}_{n_{i+1},2n_{i+1}}, f_{i+1})$, 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.

1.3 Proof Overview

In this section, we elaborate on the computational history of Korten and how the easy-witness-style argument gives us $\mathsf{F\Sigma_2P}$ and $\mathsf{FS_2P}$ algorithms.

1.3.1 Korten's reduction

We first review the key concepts and results from [Kor21] that are needed for us. Given a circuit $C\colon \{0,1\}^n \to \{0,1\}^{2n}$ and a parameter $T \ge 2n$, Korten 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 1 for an example with $k = 3$.
- For every $i \in \{0,1,\ldots,k-1\}$ and $j \in \{0,1,\ldots,2^i-1\}$, 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 $v_{k,0}, v_{k,1}, \ldots, v_{k,2^k-1}$.

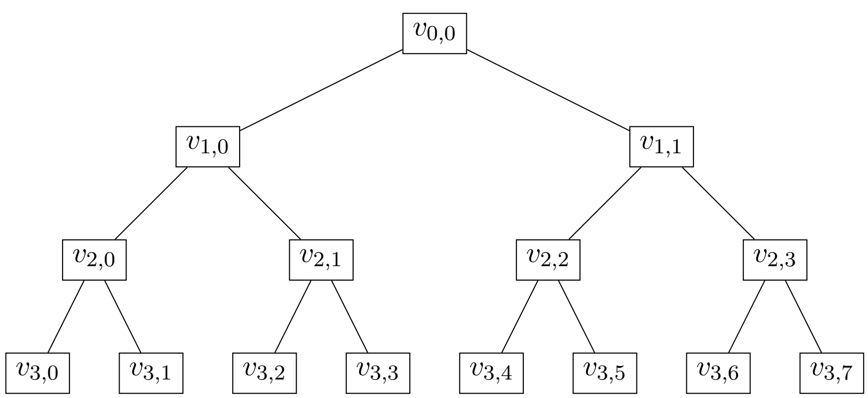

Figure 1: An illustration of the GGM Tree, in which, for instance, it holds that $(v_{3,4}, v_{3,5}) = C(v_{2,2})$.

The following two properties of $\mathsf{GGM}_T[C]$ are established in [Kor21], which will be useful for us:

[14] We use the name GGM because the construction is similar to the pseudorandom function generator of Goldreich, Goldwasser, and Micali [GGM86].

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 at most $\mathrm{poly}(\mathsf{SIZE}(C), \log T)$.

2. There is a P^{NP} algorithm $\mathsf{Korten}(C, f)$ that takes an input $f \in \{0,1\}^T \setminus \mathrm{Range}(\mathsf{GGM}_T[C])$ 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, it is not in $\mathrm{Range}(\mathsf{GGM}_T[C])$, and therefore $\mathsf{Korten}(C, f)$ solves Avoid for C. This confirms our description of Korten in Section 1.1.2.

1.3.2 Computational history of Korten and an easy-witness argument for $\mathsf{F}\Sigma_2\mathsf{P}$ algorithms

The algorithm $\mathsf{Korten}(C, f)$ 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 3.3 for details.

The computational history of Korten. The algorithm described above induces a natural description of the computational history of 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 $\mathsf{Korten}(C, f)$ (for the intermediate nodes with no label assigned, we set their labels to a special symbol $\perp$); see Figure 2 for an illustration. This history has length at most $5T$, and for convenience, we pad additional zeros at the end of it so that its length is exactly $5T$.

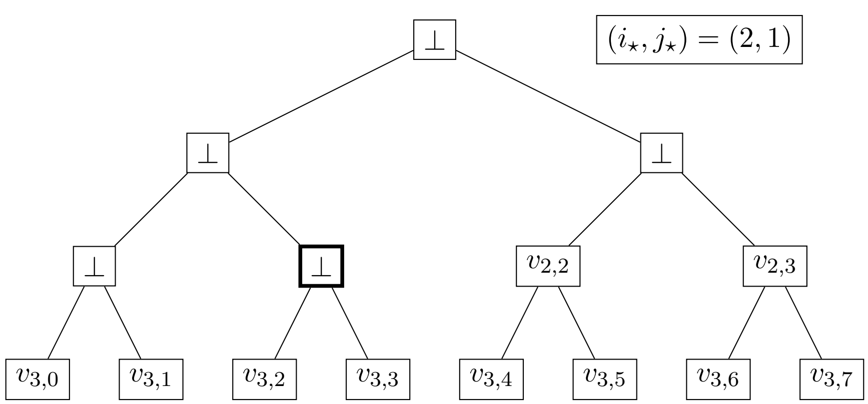

Figure 2: An illustration of the history of $\mathsf{Korten}(C, f)$. Here we have $\mathsf{History}(C, f) = (2,1) \circ \perp\perp\perp\perp\perp \circ v_{2,2} \circ v_{2,3} \circ v_{3,0} \circ \ldots \circ v_{3,7}$ and $\mathsf{Korten}(C, f) = v_{3,2} \circ v_{3,3}$.

A local characterization of $\mathsf{History}(C, f)$**.** The crucial observation we make on $\mathsf{History}(C, f)$ is that it admits a local characterization 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 $V_{\mathsf{History}}^{h,f}(x) = \psi_x(h \circ f)$. 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.

$\mathsf{F\Sigma_2P}$ **algorithms.** Before we proceed, we give a formal definition of a single-valued $\mathsf{F\Sigma_2P}$ 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]

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 1.1.2. Recall that at stage i, we have $\mathsf{ALG}_i = \mathsf{Korten}(C_{n_i}, f_i)$ 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 $\mathsf{Korten}(C_{n_i}, f_i)$ 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 simply 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 $C_1, \ldots, C_i$ that are supposed to compute $f_1, \ldots, f_i$, respectively. (Also, from Fact 3.4, one can easily construct a $\mathrm{poly}(n_0)$-size circuit C_0 computing f_0.) Next, for every $t \in \{0, 1, \ldots, i\}$, A checks whether $\mathtt{tt}(C_{t+1}) \circ \mathtt{tt}(C_t)$ satisfies all the local constraints ψ_x's from the characterization 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 $x \in \{0,1\}^{\mathrm{poly}(n_t)}$.

[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 2.2. But from this definition, it is easier to see why $\mathsf{F\Sigma_2P}$ algorithms for constructing hard truth tables imply circuit lower bounds for $\mathsf{\Sigma_2E}$.

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 $t \in \{0, 1, \ldots, i\}$. Finally, A checks whether z corresponds to the answer described in $\mathtt{tt}(C_{i+1}) = f_{i+1}$.

1.3.3 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 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 randomized selectors. Before we proceed, we give a formal definition of a single-valued $\mathsf{FS_2BPP}$ algorithm A. We implement A by a randomized 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 randomized *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 correct proof.[18]

Recap. Revising the algorithm in Section 1.2.3, our goal now is to give an $\mathsf{FS_2BPP}$ simulation of $\mathsf{Korten}(C_{n_i}, f_i)$, 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 randomized 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 $\mathsf{Korten}(C_{n_i}, f_i)$: 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 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

[17] $\mathsf{FS_2P}$ algorithms are the special case of $\mathsf{FS_2BPP}$ algorithms where the algorithm V_A is *deterministic*.

[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.

since all (i,j) are processed by $\mathsf{Korten}(C,f)$ 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 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 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.

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 randomized polynomial time. See the proof of Lemma 5.3 for the details. This part is similar to the selector construction from [Hir15].

1.4 Discussions

We conclude the introduction by discussing some related works.

1.4.1 Previous approach: 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

[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 sub-exponential 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.

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 diagonalization, 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 $\mathsf{EXP} \not\subseteq \mathsf{SIZE}[f(n)]$, then of course $\mathcal{C}$-$\mathsf{EXP} \supseteq \mathsf{EXP}$ does not have circuits of size $f(n)$;
- 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 diagonalization. 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.

1.4.2 Implications for the Missing-String problem?

In the Missing-String problem, we are given a list of m strings $x_1, x_2, \ldots, x_m \in \{0,1\}^n$ where $m < 2^n$, and the goal is to output any length-n string y that does not appear in $\{x_1, x_2, \ldots, x_m\}$. Vyas and Williams [VW23] connected the circuit complexity of Missing-String with the (relativized) circuit complexity of $\Sigma_2\mathsf{E}$:

Theorem 1.8 ([VW23, Theorem 32], Informal)**.** *The following are equivalent:*

- $\Sigma_2\mathsf{E}^A \not\subseteq$ i.o.-$\mathsf{SIZE}^A[2^{\Omega(n)}]$ *for every oracle A;*
- *for $m = 2^{\Omega(n)}$, the* Missing-String *problem can be solved by a uniform family of size-$2^{O(n)}$ depth-3 AC^0 circuits.*

The intuition behind Theorem 1.8 is roughly as follows. For every oracle A, the set of truth tables with low A-oracle circuit complexity induces an instance for Missing-String, and solving this instance gives us a hard truth table relative to A. If the algorithm for Missing-String is a uniform AC^0 circuit of depth 3, then the hard function is inside $\Sigma_2\mathsf{E}^A$.

However, despite our Theorem 1.2 being completely relativizing, it does not seem to imply any non-trivial depth-3 AC^0 circuit for Missing-String. The reason is the heavy win-win analysis *across multiple input lengths*: for each $0 \le i < t$, we have a single-valued $\mathsf{F}\Sigma_2\mathsf{P}$ construction algorithm for hard truth tables relative to oracle A on input length n_i, but this algorithm needs access to $A_{n_{i+1}}$, *a higher input length of A*. Translating this into the language of Missing-String, we obtain a weird-looking depth-3 AC^0 circuit that takes as input a *sequence* of Missing-String instances $\mathcal{I}_{n_0}, \mathcal{I}_{n_1}, \ldots, \mathcal{I}_{n_t}$ (where *each* $\mathcal{I}_{n_i} \subseteq \{0,1\}^{n_i}$ is a set of strings), looks at all of the instances

[21] There are some evidences that Karp–Lipton collapses are essential for proving circuit lower bounds [CMMW19].

(or, at least $\mathcal{I}_{n_i}$ and $\mathcal{I}_{n_{i+1}}$), and outputs a purportedly missing string of $\mathcal{I}_{n_i}$. It is guaranteed that for at least one input length i, the output string is indeed a missing string of $\mathcal{I}_{n_i}$. However, if our algorithm is only given one instance $\mathcal{I} \subseteq \{0,1\}^n$, without assistance from a larger input length, it does not know how to find any missing string of $\mathcal{I}$.

It remains an intriguing open problem whether the bullets in Theorem 1.8 are true or not. In other words, is there an oracle A relative to which $\Sigma_2\mathsf{E}$ has small circuits *on infinitely many input lengths*?

Organization

In Section 2, we introduce the necessary technical preliminaries for this paper. In Section 3, we review Korten's reduction from solving range avoidance to generating hard truth tables [Kor21], together with some new properties required by our new results. In Section 4, we prove the near-maximum circuit lower bound for $\Sigma_2\mathsf{E}$; although this lower bound is superseded by the later $\mathsf{S}_2\mathsf{E}/_1$ lower bound, we nonetheless include it in the paper since its proof is much more elementary. In Section 5, we extend the near-maximum circuit lower bound to $\mathsf{S}_2\mathsf{E}/_1$, and also present our new algorithms for solving the range avoidance problem.

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 $A(x) \in \Pi_x$.

2.1 Complexity Classes

We assume basic familiarity with computation complexity theory (see, e.g., [AB09, Gol08] for references). Below we recall the definition of $\mathsf{S}_2\mathsf{TIME}[T(n)]$ [RS98, Can96].

Definition 2.1. Let $T\colon \mathbb{N} \to \mathbb{N}$. We say a language $L \in \mathsf{S}_2\mathsf{TIME}[T(n)]$, 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 $L \in \mathsf{S}_2\mathsf{E}$ if $L \in \mathsf{S}_2\mathsf{TIME}[T(n)]$ for some $T(n) \le 2^{O(n)}$, and $L \in \mathsf{S}_2\mathsf{P}$ if $L \in \mathsf{S}_2\mathsf{TIME}[p(n)]$ for some polynomial p.

It is known that $\mathsf{S}_2\mathsf{P}$ contains MA and P^{NP} [RS98], and $\mathsf{S}_2\mathsf{P}$ is contained in $\mathsf{ZPP}^{\mathsf{NP}}$ [Cai07]. From its definition, it is also clear that $\mathsf{S}_2\mathsf{P} \subseteq \Sigma_2\mathsf{P} \cap \Pi_2\mathsf{P}$.

2.2 Single-valued $\mathsf{F}\Sigma_2\mathsf{P}$ and $\mathsf{FS}_2\mathsf{P}$ Algorithms

We consider the following definitions of single-valued algorithms which correspond to circuit lower bounds for $\Sigma_2\mathsf{E}$ and $\mathsf{S}_2\mathsf{E}$.

Definition 2.2 (Single-valued $\mathsf{F}\Sigma_2\mathsf{P}$ and $\mathsf{FS}_2\mathsf{P}$ algorithms)**.** A single-valued $\mathsf{F}\Sigma_2\mathsf{P}$ 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 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]

2.2.1 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 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 $f(x) := B(x, A(x))$ 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 passes 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 $\{\phi_i\}_{i\in[\ell]}$. □

2.3 The Range Avoidance Problem

The *range avoidance* problem [KKMP21, Kor21, RSW22] is the following problem: Given as input a circuit $C\colon \{0,1\}^n \to \{0,1\}^\ell$ where $\ell > n$, find any string $y \in \{0,1\}^\ell \setminus \mathrm{Range}(C)$. 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 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 $L \in \mathsf{S_2E}/_{\alpha(n)}$. □

3 Korten's Reduction

Our results crucially rely on a reduction in [Kor21] showing that proving circuit lower bounds is "the hardest explicit construction" under $\mathsf{P^{NP}}$ 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.

3.1 GGM Tree and the Reduction

We first recall the GGM tree construction from [GGM86], which is used in a crucial way by [Kor21].

Definition 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 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. $\square$

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.

Algorithm 3.1: $\mathsf{Korten}(C, f)$: Korten's 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 $\mathsf{Korten}(C, f)$: 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 $v_{i',j'} \leftarrow \perp$;
      Set $i_\star := i$, and $j_\star := j$;
      return $v_{i+1,2j} \circ v_{i+1,2j+1}$;
return $\perp$

Lemma 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, $\mathsf{Korten}(C, f)$ (as defined in Algorithm 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 $\mathsf{Korten}(C, f)$ follows directly from its description. Also, note that whenever $\mathsf{Korten}(C, f)$ 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 $f \in \{0,1\}^T \setminus \mathrm{Range}(\mathsf{GGM}_T[C])$, $\mathsf{Korten}(C, f)$ does not return $\perp$.

Assume, towards a contradiction, that $\mathsf{Korten}(C, f)$ 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 $f \in \{0,1\}^T \setminus \mathrm{Range}(\mathsf{GGM}_T[C])$. $\square$

In addition, we observe the following trivial fact:

Fact 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 3.4 with Lemma 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 tests every possible length-$2n$ string against the range of the circuit. It will be the basis of our win-win analysis in Section 4.

Finally, we give the following remark, showing that Korten's reduction relativizes.

Remark 3.5. Algorithm 3.1 and Lemma 3.3 *relativizes*, 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 $C^{-1}(v_{i+1,2j} \circ v_{i+1,2j+1})$.

3.2 Π_1 Verification of the History of $\mathsf{Korten}(C, f)$

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 3.1 processes all the pairs (i, j) in the reverse lexicographic order.

Definition 3.6 (The computational history of $\mathsf{Korten}(C, f)$)**.** 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 $f \in \{0,1\}^T$ be a "hard truth table" in the sense that $f \notin \mathrm{Range}(\mathsf{GGM}_T[C])$. The *computational history* of $\mathsf{Korten}(C, f)$, denoted as

$$\mathsf{History}(C, f),$$

consists of $(i_\star, j_\star)$, as well as the concatenation of $v_{i,j}$ for every $0 \leq i < k$ and $0 \leq j < 2^i$, in the lexicographical order of (i, j) ($(i_\star, j_\star)$ and the $v_{i,j}$ are defined in Algorithm 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 $v_{i,j} = \perp$ then $\mathsf{enc}(v_{i,j}) = 1^{n+1}$. The length of this history is at most $(2^{k+1} - 1)(n + 1) + 2 \log T \leq 5T$, and for convenience we always pad zeros at the end so that its length becomes exactly $5T$.

The following lemma summarizes the properties of the computational history construction above required for the $\Sigma_2\mathsf{E}$ lower bound in the next section.

Lemma 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 $h := \mathsf{History}(C, f)$ and $z := \mathsf{Korten}(C, f)$.*

1. **(history contains input/output)** *There is a* $\mathrm{poly}(\log T)$*-time one-query oracle algorithm* Input *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,* $\mathsf{Input}_{T,n}$ *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* $z = \mathsf{Korten}(C, f)$.

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:*

$$V^{f,h}(C, w) = 1 \qquad \textit{for every } w \in \{0,1\}^{5\cdot(\log T+n)}.$$

Proof. From the definition of $\mathsf{History}(C, f)$, the construction of $\mathsf{Input}_{T,n}$ 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 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 3.6 and Lemma 3.7 relativize.

Remark 3.8. Definition 3.6 and Lemma 3.7 *relativize*, in the sense that if C is an oracle circuit C^O for some arbitrary oracle, Definition 3.6 needs no modification since Algorithm 3.1 relativizes, and Lemma 3.7 holds with the only modification that V now also need to take O as an oracle (since it needs to evaluate C).

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 $\{C_n : \{0,1\}^n \to \{0,1\}^{2n}\}_{n\in\mathbb{N}}$ 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 4.1. *Let $\{C_n : \{0,1\}^n \to \{0,1\}^{2n}\}_{n\in\mathbb{N}}$ 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 $1 \le i \le t^{(\ell)}$, 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 $i \in \{0,1,\dots,t^{(\ell)}\}$ 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 $i \in \{0,1,\dots,t\}$.

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 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 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 $T_{t+1} \le 4n_{t+1}$. Note that we have $T_i = 5^i \cdot T_0 \le 2^{3n_0 + i\cdot\log 5}$ and $n_i = (n_0)^{10^i} = 2^{\log n_0 \cdot 10^i}$. Hence, we have that $t \le O(\log n_0)$. (Also note that $n_t^{(\ell)} < n_0^{(\ell+1)}$.)

Description of our $\mathsf{F}\Sigma_2\mathsf{P}$ 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 $z := \mathsf{Korten}(C_{n_k}, f_k)$. It follows from Lemma 3.3 that z is not in the range of C_{n_k}. Our single-valued $\mathsf{F}\Sigma_2\mathsf{P}$ algorithm A computes z on input 1^{n_k} (see Definition 2.2). That is, for some $\ell_1, \ell_2 \le \mathrm{poly}(n_k)$:

- 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 $\bot$.

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 $\ell_1 := n_{k+1} \le \mathrm{poly}(n_k)$. If $k = t$, then V_A defines $\hat{f}_{k+1} := \pi_1$ and $\ell_1 := T_{t+1} \le \mathrm{poly}(n_k)$. 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 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 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 Input described in Item 1 of Lemma 3.7. Therefore, for each i from k downto 1, we can recursively define $\hat{f}_i$ such that $(\hat{f}_i)_j = \mathsf{Input}^{\hat{f}_{i+1}}_{T_i,n_i}(j)$. 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 Input 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 Input 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 $\ell_2 := \log(k + 1) + 5(\log T_k + n_k) \le \mathrm{poly}(n_k)$. Now, let V_{History} be the oracle algorithm in Item 2 of Lemma 3.7, we let $V_A(1^{n_k}, \pi_1, \pi_2)$ check whether the following holds:

$$V^{\hat{f}_i,\hat{f}_{i+1}}_{\mathsf{History}}(C_{n_i}, w) = 1.\text{[24]} \tag{1}$$

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 3.7. Otherwise, V_A outputs $\perp$.

The correctness of A. Before establishing the correctness of A, we need the following claim:

Claim 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 $w \in \{0,1\}^{5(\log T_i + n_i)}$.*

Proof. First, assume that $f_{k+1} = \hat{f}_{k+1}$. By Item 1a of Lemma 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 3.7, we have that for every $i \in \{0, 1, \ldots, k\}$, and for every $w \in \{0,1\}^{5(\log T_i + n_i)}$, $V^{\hat{f}_i,\hat{f}_{i+1}}_{\mathsf{History}}(C_{n_i}, w) = 1$ 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 $V^{\hat{f}_i,\hat{f}_{i+1}}_{\mathsf{History}}(C_{n_i}, w) = 1$ 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 3.7, it follows that $f_i = \hat{f}_i$ for every $i \in \{0, 1, \ldots, k + 1\}$. In particular, $f_{k+1} = \hat{f}_{k+1}$. $\diamond$

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 $z = \mathsf{Korten}(C_{n_k}, f_k)$. 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 4.2, we know that V_A will output $z = \mathsf{Korten}(C_{n_k}, f_k)$ 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, $V_A(1^{n_k}, \pi_1, \pi_2) \ne \perp$. In this case, every invocation of Equation 1 holds, and thus by Claim 4.2 we know that $f_{k+1} = \hat{f}_{k+1}$. It follows that $\mathsf{Korten}(C_{n_k}, f_k) = z$ and V_A will output z regardless of π_2.

[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.

Finally, we generalize A and V_A to work on all inputs 1^n. On input 1^n, V_A calculates the largest ℓ such that $n^{(\ell)} \le n$, and also calculates the largest k' such that $n^{(\ell)}_{k'} \le n$. If $n^{(\ell)}_{k'} \ne n$, 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 $\mathsf{F\Sigma_2P}$ algorithm solving the range avoidance problem of $\{C_n\}_{n\in\mathbb{N}}$. $\square$

From Remark 3.5 and Remark 3.8, one can obverse that the proof above also relativizes. Hence we have the following as well.

Theorem 4.3 (Relativized version of Theorem 4.1)**.** *Let $\mathcal{O}\colon \{0,1\}^* \to \{0,1\}$ be any oracle. Let $\{C^{\mathcal{O}}_n \colon \{0,1\}^n \to \{0,1\}^{2n}\}_{n\in\mathbb{N}}$ 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 $y_n \in \{0,1\}^{2n} \setminus \mathrm{Range}(C^{\mathcal{O}}_n)$.*

We omit the proof of the following corollary since it is superseded by the results in the next section.

Corollary 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 relativize: for every oracle $\mathcal{O}$, $\Sigma_2\mathsf{E}^{\mathcal{O}} \not\subseteq \mathsf{SIZE}^{\mathcal{O}}[2^n/n]$ and $(\Sigma_2\mathsf{E}^{\mathcal{O}} \cap \Pi_2\mathsf{E}^{\mathcal{O}})/_1 \not\subseteq \mathsf{SIZE}^{\mathcal{O}}[2^n/n]$.*

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.

5.1 Reed–Muller Codes

To prove maximum circuit lower bounds for $\mathsf{S_2E}/_1$, 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 $\mathsf{BRM}_{\mathbb{F}_p,\Delta,m}(S) = \mathsf{Enc}_{\mathbb{F}_p}(P(w_1)) \circ \mathsf{Enc}_{\mathbb{F}_p}(P(w_2)) \circ \ldots \circ \mathsf{Enc}_{\mathbb{F}_p}(P(w_{p^m}))$. 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.

[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. Also, such P can be constructed by 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.

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 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,*

$$\Pr_{\vec{x} \leftarrow \mathbb{F}_p^m}[g(\vec{x}) \neq P(\vec{x})] \leq 1/4.$$

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 $2/3$.

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 5.2 (Low-max-degree tester [BFL91, Remark 5.15]). *Let $n, \Delta, p \in \mathbb{N}$ be such that $p \geq 20 \cdot (\Delta + 1)^2 \cdot n^2$ 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 $\deg_{\max}(g) \leq \Delta$, then $\mathsf{LDT}^g(p, n, \Delta)$ accepts with probability 1,*
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 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.

[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.

[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.

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 randomized 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 $f(w) \neq g(w)$. $\square$

5.2 Encoded History and $\mathsf{S_2BPP}$ Verification

Next, we define the following encoded history.

Definition 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 3.1. Let S be the concatenation of $\mathsf{enc}(v_{i,j})$ for every $i \in \{0, 1, \ldots, k-1\}$, $j \in \{0, 1, \ldots, 2^i - 1\}$, 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 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 $\mathsf{Korten}(C, f)$, 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

$$\lceil \log(40 \cdot \log^5 T) \rceil \cdot (40 \cdot \log^5 T)^{\log(5T)/\log\log T + 1} + 2\log T \leq T^6$$

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 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 $h \coloneqq \widetilde{\mathsf{History}}(C, f)$ and $z \coloneqq \mathsf{Korten}(C, f)$.*

[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.

[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 5.7 since there we only care about sufficiently large T.

1. **(history contains input/output)** *There is a* $\mathrm{poly}(\log T)$*-time oracle algorithm* Input *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) $\mathsf{Input}_{T,n}$ *makes a single query to its oracle; when given* h *as the oracle,* $\mathsf{Input}_{T,n}$ *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* $z = \mathsf{Korten}(C,f)$*.*

2. **($\mathsf{S_2BPP}$ verification of the history)** *There is a randomized 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.*

   (b) *For every* $\pi \in \{0,1\}^{T^6}$ *and every* $i \in \{0,1,\dots,T^6-1\}$*, we have that*

$$\Pr\left[V_{T,n}^{f,\pi,h}(C,i,\varepsilon) = h_i\right] \ge 1-\varepsilon \quad \textit{and} \quad \Pr\left[V_{T,n}^{f,h,\pi}(C,i,\varepsilon) = h_i\right] \ge 1-\varepsilon.$$

Proof. Again, the algorithms $\mathsf{Input}_{T,n}$ 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 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. We first run $\mathsf{LDT}^{g_1}(p,m,\Delta-1)$ and $\mathsf{LDT}^{g_2}(p,m,\Delta-1)$ for c_1 times, where c_1 is a sufficiently large constant. Recall that $p \ge 20 \cdot \log^5 T$, $m = \lceil \log(5T)/\log\log T \rceil$, and $\Delta = \log T$. It follows that $p \ge 20 \cdot ((\Delta-1)+1)^2 \cdot m^2$, which satisfies the condition of Lemma 5.2. We also note that $3m^2 \cdot ((\Delta-1)+1)/p < 1/4$. Hence, by Lemma 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 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 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.

[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 $2\log T + 2n \le 4n$ bits from h.

[32] Technically π_1 and π_2 are supposed to contain the RM codewords concatenated with $\mathsf{Enc}_{\mathbb{F}_p} : \mathbb{F}_p \to \{0,1\}^{\lceil \log p \rceil}$.

[33] As a minor detail, if both g_1 and g_2 are rejected by some runs, V selects π_2.

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] Note that $\mathsf{Comp}^{\widetilde{g}_1,\widetilde{g}_2}(p,m,\Delta-1)$ makes 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 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 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 $\Pr\left[V^{f,h,\pi}_{T,n}(i) = h(i)\right] \geq 2/3$. (The proof for $\Pr\left[V^{f,\pi,h}_{T,n}(i) = h(i)\right] \geq 2/3$ is symmetric.) 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 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 $\mathsf{Korten}(C,f)$.

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 $\widetilde{g}_1 \neq \widetilde{g}_2$.

[34] Recall that the $\{v_{i,j}\}$ is encoded in the reverse lexicographic order (Definition 5.4).

[35] From Lemma 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 $1/(mT)^{c_2}$. So 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.

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 element from $C^{-1}(\alpha)$, so again h is selected. In the last case: (1) if $v^1_{i_\circ,j_\circ} = \bot$, then $v^1_{i',j'}$ has to be $\bot$ as well, thus h is selected; (2) if $v^1_{i_\circ,j_\circ} \neq \bot$ and $v^1_{i',j'} = \bot$, 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 $v^1_{i_\circ,j_\circ} \neq \bot$ and $v^1_{i',j'} \neq \bot$, 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 $\bot$, V operates exactly the same. But when Comp returns $\bot$, 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 $v^b_{i^b_\star,j^b_\star} = \bot$, 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 $\bot$, 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. □

5.2.1 A remark on relativization

Perhaps surprisingly, although Lemma 5.5 heavily relies on arithmetization tools such as Reed–Muller encoding and low-degree tests, it in fact also relativizes. To see this, the crucial observation is that, similarly to Lemma 3.7, the verifier V from Lemma 5.5 only needs *black-box access* to the input circuit C, meaning that it only needs to evaluate C on certain chosen input. 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 5.6. Definition 5.4 and Lemma 5.5 *relativize*, in the sense that if C is an oracle circuit $C^{\mathcal{O}}$ for some arbitrary oracle, Definition 5.4 needs no modification since Definition 3.6 relativizes, and Lemma 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 *relativize*, 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 3.6) and in the construction of the encoded history (Definition 5.4). That is, the arithmetization in the encoded history *does not arithmetize* the circuit C itself.

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 5.7. *Let $\{C_n : \{0,1\}^n \to \{0,1\}^{2n}\}_{n\in\mathbb{N}}$ 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 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 $i \in \{0, 1, \ldots, t\}$.

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 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 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 $k \in \{0, 1, \ldots, t\}$ 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 $z := \mathsf{Korten}(C_{n_k}, f_k)$, it follows from Lemma 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 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 randomized 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 $\ell := n_{k+1} \le \mathrm{poly}(n_k)$. If $k = t$, then V defines $\hat{f}_{k+1} := \pi_1$, $\hat{g}_{k+1} := \pi_2$, and their lengths are $\ell := T_{t+1} \le n_{k+1} \le \mathrm{poly}(n_k)$. 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 $f_{k+1} \in \mathrm{Range}(\mathsf{GGM}_{T_{k+1}}[C_{n_{k+1}}])$, 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 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 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 Input described in Item 1 of Lemma 5.5. Therefore, for each i from k downto 1, we can recursively define $\hat{f}_i$ such that $(\hat{f}_i)_j = \mathsf{Input}_{T_i,n_i}^{\hat{f}_{i+1}}(j)$ (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 Input 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 Input 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 $i \in [k+1]$, we define

$$V_i(\alpha) := \mathsf{Select}_{T_{i-1},n_{i-1}}^{V_{i-1},\hat{f}_i,\hat{g}_i}(C_{n_{i-1}}, \alpha, \varepsilon_i)$$

for some $\varepsilon_i \in [0,1)$ to be specified later, where Select is the algorithm in Item 2 of Lemma 5.5. We note that since V_{i-1} is a randomized 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

$$V(1^{n_k}, \pi_1, \pi_2) := \mathsf{Output}_{T_k,n_k}^{V_{k+1}}$$

(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 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, $\mathsf{Select}_{T_{i-1},n_{i-1}}^{V_{i-1},\hat{f}_i,\hat{g}_i}(C_{n_{i-1}}, \alpha, \varepsilon_i)$ makes at most $(n_{i-1} \cdot \log 1/\varepsilon_i)^\tau$ many queries to V_{i-1}.

We say $\mathsf{Select}_{T,n}^{f,\pi_1,\pi_2}(C, \alpha, \varepsilon_i)$ makes an error if the following statements hold ($h = \widetilde{\mathsf{History}}(C, f)$ from Lemma 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 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

$$\varepsilon_i = \frac{\varepsilon_{i+1}}{4 \cdot (n_i \cdot \log 1/\varepsilon_{i+1})^\tau}.$$

[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.

To show the correctness of V, we prove the following claim by induction.

Claim 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 $\mathsf{Select}^{V_{i-1},\hat{f}_i,\hat{g}_i}_{T_{i-1},n_{i-1}}(C_{n_{i-1}}, \alpha, \varepsilon_i)$ 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 $1 - 2\varepsilon_i$. $\diamond$

Note that $\mathsf{Output}^{V_{k+1}}_{T_k,n_k}$ makes at most $4n_k$ queries to V_{k+1}. It follows from Claim 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 analyze 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

$$\log \varepsilon_i^{-1} \leq 2^k \cdot O\left(\sum_{\ell=0}^{k} \log n_\ell\right) \leq \mathrm{poly}(n_0) \cdot \log n_k.$$

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 $k \leq O(\log n_0)$, 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.

Derandomization 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 derandomize V into a deterministic $\mathsf{S_2P}$ verifier V_A that outputs z.

By repeating V $\mathrm{poly}(n_k)$ times and outputs 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

$$\Pr[\widetilde{V}(1^{n_k}, \omega, \pi) = z] \geq 1 - 2^{-n_k} \qquad \text{and} \qquad \Pr[\widetilde{V}(1^{n_k}, \pi, \omega) = z] \geq 1 - 2^{-n_k}. \tag{2}$$

Let $\ell = \mathrm{poly}(n_k)$ be an upper bound on the number of random coins used by $\widetilde{V}$. We also let $m := \mathrm{poly}(\ell, n_{k+1}) \leq \mathrm{poly}(n_k)$ 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 $\vec{\pi}_1, \vec{\pi}_2 \in \{0,1\}^{n_{k+1}} \times \left(\{0,1\}^{\ell}\right)^m$ 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 $\vec{\pi} \in \{0,1\}^{n_{k+1}} \times \left(\{0,1\}^{\ell}\right)^m$,

$$\Pr[V_A(1^{n_k}, \vec{\omega}, \vec{\pi}) = z] \quad \text{and} \quad \Pr[V_A(1^{n_k}, \vec{\pi}, \vec{\omega}) = z].$$

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 $\widetilde{V}(1^{n_k}, \omega, \pi; r_i \oplus u) \neq z$ is at most $2^{-\Omega(m)}$, and the same probability upper bound holds for the corresponding case of $\widetilde{V}(1^{n_k}, \pi, \omega; r_i \oplus u) \neq z$ 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 (2) holds. We simply 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 generalize 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 $\mathsf{FS_2P}$ algorithm solving the range avoidance problem of $\{C_n\}$. $\square$

Moreover, observe that in the proof of Lemma 5.5, all considered input lengths (the $n_i^{(\ell)}$) are indeed powers of 2. So we indeed have the following slightly stronger result.

Corollary 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 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 5.10 and Theorem 2.3.

Corollary 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 2.4 and Corollary 5.11.

Corollary 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 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 5.6 and noting that the derandomization of $\mathsf{S_2BPP}$ verifier V to $\mathsf{S_2P}$ verifier A_V also relativizes, we can see that all the results above relativize as well.

5.4 Infinitely Often Single-Valued $\mathsf{FS_2P}$ Algorithm for Arbitrary Input Range Avoidance

Theorem 5.7 and Corollary 5.11 only give single-valued $\mathsf{FS_2P}$ algorithms for solving range avoidance for P-uniform families of circuits. Applying Korten's reduction [Kor21], we show that it can be strengthened into a single-valued infinitely-often $\mathsf{FS_2P}$ algorithm solving range avoidance given an arbitrary input circuit.

We need the following reduction from [Kor21].

Lemma 5.14 ([Kor21, Theorem 7]). *There is an $\mathsf{FP}^{\mathsf{NP}}$ algorithm A_{Korten} satisfying the following:*

1. *A_{Korten} takes an s-size circuit $C\colon \{0,1\}^n \to \{0,1\}^{n+1}$ and a truth table $f \in \{0,1\}^{2^m}$ such that $2^m \geq s^3$ and $n \leq s$ as input.*
2. *If the circuit complexity of f is at least $c_1 \cdot m \cdot s$ for a sufficiently large universal constant $c_1 \in \mathbb{N}$, then $A_{\mathsf{Korten}}(C, f)$ outputs a string $y \in \{0,1\}^{n+1} \setminus \mathrm{Range}(C)$.*

Theorem 5.15. *There is a single-valued $\mathsf{FS_2P}$ 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 $y_C \in \{0,1\}^{n+1} \setminus \mathrm{Range}(C)$.*

Proof Sketch. By Corollary 5.11, there is a single-valued $\mathsf{FS_2P}$ 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 $\mathsf{FS_2P}$ 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 $A_{\mathsf{Korten}}(C, f_m)$. It follows from Theorem 2.3 that A is a single-valued $\mathsf{FS_2P}$ algorithm with one bit of advice (the advice of A is given to W). $\square$

Finally, $\mathsf{S_2P} \subseteq \mathsf{ZPP}^{\mathsf{NP}}$ [Cai07] implies that every single-valued $\mathsf{FS_2P}$ 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 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 $n \le s$, $A(C)$ outputs $y_C \in \{0,1\}^{n+1} \setminus \mathrm{Range}(C)$. That is, for all those s, there is a string $y_C \in \{0,1\}^{n+1} \setminus \mathrm{Range}(C)$ 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 $2/3$.*

Acknowledgments

Part of the work was done when all authors were participating in the Meta-Complexity program at the Simons Institute. Lijie Chen is supported by a Miller Research Fellowship. Shuichi Hirahara is supported by JST, PRESTO Grant Number JPMJPR2024, Japan. Hanlin Ren received support from DIMACS through grant number CCF-1836666 from the National Science Foundation. We thank Oliver Korten, Zhenjian Lu, Igor C. Oliveira, Rahul Santhanam, Roei Tell, and Ryan Williams for helpful discussions. We also want to thank Jiatu Li, Igor C. Oliveira, and Roei Tell for comments on an early draft of the paper.

References

[Aar06] Scott Aaronson. Oracles are subtle but not malicious. In *CCC*, pages 340–354. IEEE Computer Society, 2006. doi:10.1109/CCC.2006.32. 1

[AB09] Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach.* Cambridge University Press, 2009. 14

[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. 5

[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. 1, 13

[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. 23

[BFNW93] László Babai, Lance Fortnow, Noam Nisan, and Avi Wigderson. BPP has subexponential time simulations unless $\mathsf{EXPTIME}$ has publishable proofs. *Computational Complexity*, 3:307–318, 1993. doi:10.1007/BF01275486. 13

[BFT98] Harry Buhrman, Lance Fortnow, and Thomas Thierauf. Nonrelativizing separations. In *CCC*, pages 8–12, 1998. doi:10.1109/CCC.1998.694585. 1

[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. 4

[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. 7

[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. 1, 2, 3, 13, 14, 33

[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. 11, 14, 28, 31

[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. 1

[CLO+23] Lijie Chen, Zhenjian Lu, Igor C. Oliveira, Hanlin Ren, and Rahul Santhanam. Polynomial-time pseudodeterministic construction of primes. In *FOCS*, 2023. To appear. doi:10.48550/arXiv.2305.15140. 5, 6, 7

[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. 13

[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. 5, 6

[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. 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. 2

[Erd59] Paul Erdős. Graph theory and probability. *Canadian Journal of Mathematics*, 11:34–38, 1959. doi:10.4153/CJM-1959-003-9. 2

[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. 4

[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. 1, 4, 16

[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. 3, 4

[GGM86] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions. *Journal of the ACM*, 33(4):792–807, 1986. doi:10.1145/6490.6503. 8, 16, 17

[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. 2

[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. 2

[Gol08] Oded Goldreich. *Computational complexity: a conceptual perspective*. Cambridge University Press, 2008. doi:10.1017/CBO9780511804106. 14

[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. 23

[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. 12, 23

[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. 1, 3

[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. 4

[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. 13

[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. 6, 7

[IW97] Russell Impagliazzo and Avi Wigderson. $\mathsf{P} = \mathsf{BPP}$ if E requires exponential circuits: Derandomizing the XOR lemma. In *STOC*, pages 220–229. ACM, 1997. doi:10.1145/258533.258590. 1, 6

[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. 2, 6

[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. 1, 2

[KC00] Valentine Kabanets and Jin-Yi Cai. Circuit minimization problem. In *STOC*, pages 73–79, 2000. doi:10.1145/335305.335314. 1

[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. 2, 4, 16

[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. 2, 12, 13

[Kor21] Oliver Korten. The hardest explicit construction. In *FOCS*, pages 433–444. IEEE, 2021. doi:10.1109/FOCS52979.2021.00051. 2, 4, 5, 6, 8, 14, 16, 17, 32

[Kra01] Jan Krajíček. Tautologies from pseudo-random generators. *Bull. Symb. Log.*, 7(2):197–212, 2001. doi:10.2307/2687774. 2

[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. 15

[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. 2

[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. 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. 13

[Li23] Xin Li. Two source extractors for asymptotically optimal entropy, and (many) more. In *FOCS*, 2023. To appear. doi:10.48550/arXiv.2303.06802. 2

[Lup58] Oleg B Lupanov. On the synthesis of switching circuits. *Doklady Akademii Nauk SSSR*, 119(1):23–26, 1958. 1

[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. 1, 4

[NW94] Noam Nisan and Avi Wigderson. Hardness vs randomness. *Journal of Computer and System Sciences*, 49(2):149–167, 1994. doi:10.1016/S0022-0000(05)80043-1. 1, 6

[RS98] Alexander Russell and Ravi Sundaram. Symmetric alternation captures BPP. *Comput. Complex.*, 7(2):152–162, 1998. doi:10.1007/s000370050007. 2, 11, 14, 28, 31

[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. 2, 4, 16

[San09] Rahul Santhanam. Circuit lower bounds for Merlin–Arthur classes. *SIAM J. Comput.*, 39(3):1038–1061, 2009. doi:10.1137/070702680. 1

[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. 4

[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. 1, 4

[Sud95] Madhu Sudan. *Efficient Checking of Polynomials and Proofs and the Hardness of Approximation Problems*, volume 1001 of *Lecture Notes in Computer Science*. Springer, 1995. doi:10.1007/3-540-60615-7. 23

[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

[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. 1

[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. 1, 13

ECCC ISSN 1433-8092

https://eccc.weizmann.ac.il