Symmetric Exponential Time Requires Near-Maximum Circuit Size*

(Draft of Journal Version)

Lijie Chen
UC Berkeley
lijiechen@berkeley.edu

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

Zeyong Li
National University of Singapore
li.zeyong@u.nus.edu

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

July 8, 2024

Abstract

We show that there is a language in $\mathsf{S}_2\mathsf{E}$ (symmetric exponential time) that requires circuit complexity at least $2^n/n$ on every input length. In particular, the above also implies the same near-maximum circuit lower bounds for $\Sigma_2\mathsf{E} \cap \Pi_2\mathsf{E}$ and $\mathsf{ZPE}^{\mathsf{NP}}$. Our proofs relativise. Previously, only "half-exponential" circuit lower bounds for the aforementioned 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 that solves the range avoidance problem. This algorithm also implies unconditional pseudodeterministic $\mathsf{FZPP}^{\mathsf{NP}}$ constructions for Ramsey graphs, rigid matrices, two-source extractors, linear codes, and $\mathrm{K}^{\mathrm{poly}}$-random strings with nearly optimal parameters.

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

*This submission is merged from the two extended abstracts [CHR24, Li24], both presented at the 56th Annual ACM Symposium on Theory of Computing (STOC 2024).

[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 $2^n/n$.

$\mathsf{prP} = \mathsf{prBPP}$ [NW94, IW97], while super-polynomial lower bounds such as $\mathsf{EXP} \not\subset \mathsf{P}/_{\mathrm{poly}}$ only imply sub-exponential time derandomisation of prBPP.[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] emphasised our lack of knowledge regarding the circuit complexity of $\Sigma_2\mathsf{EXP}$, even with respect to *relativising* proof techniques. In particular, the following question has been open for at least 20 years (indeed, if we count from [Kan82], it would be at least 40 years):

Open Problem 1.1. *Can we prove that $\Sigma_2\mathsf{EXP} \not\subset \mathsf{SIZE}[2^{\varepsilon n}]$ for some absolute constant $\varepsilon > 0$, or at least show a relativisation barrier for proving such a lower bound?*

The half-exponential barrier. There is a richer literature regarding super-polynomial lower bounds than exponential lower bounds. Kannan [Kan82] proved that the class $\Sigma_2\mathsf{E} \cap \Pi_2\mathsf{E}$ does not have polynomial-size circuits. Subsequent works proved super-polynomial circuit lower bounds for exponential-time complexity classes such as $\mathsf{ZPEXP}^{\mathsf{NP}}$ [KW98, BCG+96], $\mathsf{S}_2\mathsf{EXP}$ [CCHO05, Cai07], PEXP [Vin05, Aar06], and MAEXP [BFT98, San09].

Unfortunately, all these works fail to prove exponential lower bounds. All of their proofs go through certain *Karp–Lipton* collapses [KL80]; such a proof strategy runs into a so-called "half-exponential barrier" [MVW99], preventing us from getting exponential lower bounds. See Appendix A 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 providing a proof of Kannan's original claim that $\Sigma_2\mathsf{E} \cap \Pi_2\mathsf{E}$ requires near-maximum $(2^n/n)$ circuit complexity. Moreover, our proof indeed *relativises*:

Theorem 1.2. $\Sigma_2\mathsf{E} \cap \Pi_2\mathsf{E} \not\subset$ i.o.-$\mathsf{SIZE}[2^n/n]$.[4] *Moreover, this holds in every relativised world.*

With some more work, we extend our lower bounds to the smaller complexity class $\mathsf{S}_2\mathsf{E}$ (see Definition 2.1 for a formal definition), again with a relativising proof:

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

[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.

[4] We use i.o.-$\mathsf{SIZE}[s(n)]$ to denote the set of languages L such that there are *infinitely many* input lengths n where $L_n = L \cap \{0,1\}^n$ can be computed by a circuit of size $s(n)$. If a language L is not in i.o.-$\mathsf{SIZE}[s(n)]$, then we have a circuit lower bound for L on *almost every* input length. This is a stronger statement than $L \notin \mathsf{SIZE}[s(n)]$ as the latter statement only asserts a lower bound on *infinitely many* input lengths.

The symmetric time class $\mathsf{S_2E}$. $\mathsf{S_2E}$ can be seen as a "randomised" version of $\mathsf{E^{NP}}$ since it is sandwiched between $\mathsf{E^{NP}}$ and $\mathsf{ZPE^{NP}}$: it is easy to show that $\mathsf{E^{NP}} \subseteq \mathsf{S_2E}$ [RS98], and it is also known that $\mathsf{S_2E} \subseteq \mathsf{ZPE^{NP}}$ [Cai07]. We also note that under plausible derandomisation assumptions (e.g., $\mathsf{E^{NP}}$ requires $2^{\Omega(n)}$-size SAT-oracle circuits), all three classes simply collapse to $\mathsf{E^{NP}}$ [KvM02].

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

Corollary 1.4. $\mathsf{ZPE^{NP}} \not\subseteq$ i.o.-$\mathsf{SIZE}[2^n/n]$. *Moreover, this holds in every relativised 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 $\mathrm{Range}(C) := \{C(z) : z \in \{0,1\}^n\}$). 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 $\mathsf{FZPP^{NP}}$ 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, derandomising the trivial $\mathsf{FZPP^{NP}}$ algorithm for Avoid would imply explicit constructions for all these important objects.

Our results: new pseudodeterministic algorithms for Avoid. We show that the trivial $\mathsf{FZPP^{NP}}$ algorithm for Avoid can be *unconditionally* made *pseudodeterministic*. A *pseudodeterministic* algorithm [GG11] is a randomised algorithm that outputs the same *canonical* answer on most computational paths. In particular, we have:

Theorem 1.5. *There is a randomised algorithm $\mathcal{A}$ with an* NP *oracle such that the following holds. Given a circuit $C\colon \{0,1\}^n \to \{0,1\}^{n+1}$ as input, there is a string $y_C \in \{0,1\}^n \setminus \mathrm{Range}(C)$, which only depends on C (and, importantly, not on the internal randomness of $\mathcal{A}$), such that $\mathcal{A}(C)$ either outputs y_C or $\perp$, and the probability (over the internal randomness of $\mathcal{A}$) that $\mathcal{A}(C)$ outputs y_C is at least $2/3$. Moreover, this theorem holds in every relativised world.*

As a corollary, we obtain a zero-error pseudodeterministic construction with an NP oracle for every problem in APEPP:

Corollary 1.6 (Informal)**.** *There are zero-error pseudodeterministic constructions for the following objects with an* NP *oracle for every input size n: Ramsey graphs, rigid matrices, two-source extractors, linear codes, hard truth tables, and $\mathrm{K}^{\mathrm{poly}}$-random strings.*

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

Theorem 1.5 is tantalisingly close to an $\mathsf{FP^{NP}}$ algorithm for Avoid (with the only caveat of being *zero-error* instead of being completely *deterministic*). However, since an $\mathsf{FP^{NP}}$ algorithm for range avoidance would imply near-maximum circuit lower bounds for E^{NP}, we expect that it would require fundamentally new ideas to completely derandomise our algorithm. Previously, Hirahara, Lu, and Ren [HLR23, Theorem 36] presented an infinitely-often pseudodeterministic $\mathsf{FZPP}^{\mathsf{NP}}$ algorithm for Avoid using n^{ε} bits of advice, for any small constant $\varepsilon > 0$. Our result improves the above in a few aspects: our algorithm works for *every* input length and needs *no* advice, and our techniques relativise while theirs do not.

Lower bounds against non-uniform computation with maximum advice length. Finally, our results also imply lower bounds against non-uniform computation with maximum advice length. We mention this corollary because it is a stronger statement than circuit lower bounds, and similar lower bounds appeared recently in the literature of super-fast derandomisation [CT21].

Corollary 1.7. *For every $\alpha(n) \geq \omega(1)$ and any constant $k \geq 1$, $\mathsf{S}_2\mathsf{E} \not\subset$ i.o.-$\mathsf{TIME}[2^{kn}]/_{2^n-\alpha(n)}$. Moreover, this holds in every relativised world.*

1.1.3 Depth-3 circuits for the Missing-String problem

Our results also imply a family of new depth-3 circuits for the Missing-String problem. The Missing-String problem is defined as follows: given a list of m strings $x_1, \ldots, x_m \in \{0,1\}^n$ where $m < 2^n$, the goal is to output any string $y \in \{0,1\}^n$ not in the list. Vyas and Williams [VW23] connected the circuit complexity of the Missing-String (in the regime where $m < 2^n/2$) with the (relativised) circuit complexity of $\Sigma_2\mathsf{E}$.

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

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

As a corollary of our circuit lower bound which relativises, we can conclusively claim that:

Corollary 1.9. *For $m = 2^{\Omega(n)}$, the Missing-String problem can be solved by a uniform family of size-$2^{\mathrm{poly}(n)}$ depth-3 AC^0 circuits.*

1.2 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 emphasised in recent papers on the range avoidance problem [Kor21, RSW22].

Let $\Pi \subseteq \{0,1\}^{\star}$ be an *ε-dense* property, i.e., for every integer $N \in \mathbb{N}$, $|\Pi_N| \geq \varepsilon \cdot 2^N$. (In what follows, we use $\Pi_N := \Pi \cap \{0,1\}^N$ to denote the length-N slice of Π.) As a concrete example, let Π_{hard} be the set of hard truth tables, i.e., a string $tt \in \Pi_{\mathrm{hard}}$ 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 $\mathsf{BPE} \not\subset$ i.o.-$\mathsf{SIZE}[2^n/n]$.[6] In this regard, the previous near-maximum circuit lower bound for $\Delta_3\mathsf{E} := \mathsf{E}^{\Sigma_2\mathsf{P}}$ [MVW99] can be summarised in one sentence: The lexicographically first string in Π_{hard} can be constructed in $\Delta_3\mathsf{P} := \mathsf{P}^{\Sigma_2\mathsf{P}}$ (which is necessarily single-valued).

Reduction to Avoid. It was observed in [KKMP21, Kor21] that explicit construction of elements from Π_{hard} is a special case of range avoidance: Let $\mathsf{TT}\colon \{0,1\}^{N-1} \to \{0,1\}^N$ (here $N = 2^n$) be a circuit that maps the description of a $2^n/n$-size circuit into its 2^n-length truth table (by [FM05], this circuit can be encoded by $N-1$ bits). Hence, a single-valued algorithm solving Avoid for TT is equivalent to a single-valued construction for Π_{hard}. This explains how our new range avoidance algorithms imply our new circuit lower bounds (as mentioned in Section 1.1.2).

1.3 Proof overview

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.

1.3.1 Korten's reduction

We start by reviewing Korten's reduction [Kor21], which reduces Avoid on a circuit $C : \{0,1\}^n \to \{0,1\}^{2n}$ to Avoid on some other circuit with much longer stretch.[7]

Given any circuit $C : \{0,1\}^n \to \{0,1\}^{2n}$ and parameter $T = n \cdot 2^k$, Korten builds another circuit $\mathsf{GGM}_T[C] : \{0,1\}^n \to \{0,1\}^T$ by applying the circuit C in a perfect binary tree:

1. Build a perfect binary tree of height k where for each $0 \leq i \leq k$ and $0 \leq j < 2^i$, (i,j) denotes the j'th vertex on the i'th level. For each $0 \leq i < k$ and $0 \leq j < 2^i$, the left child and right child of node (i,j) are nodes $(i+1, 2j)$ and $(i+1, 2j+1)$ respectively.
2. Assign the root vertex $(0,0)$ with value $v_{0,0} = x$. For each vertex (i,j) on the tree, evaluate $y = C(v_{i,j})$ and assign its left child $v_{i+1,2j}$ with the first n bits of y and its right child $v_{i+1,2j+1}$ with the last n bits of y.
3. The output of $\mathsf{GGM}_T[C](x)$ is simply the concatenation of the values of the 2^k leaves.

(The structure of this tree resembles the construction of pseudorandom functions by Goldreich, Goldwasser, and Micali [GGM86], hence the name $\mathsf{GGM}_T[C]$.)

[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) $\mathsf{BPE} \not\subset$ 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]Here we assume that C_n stretches n bits to $2n$ bits instead of $n+1$ bits, as there is another reduction in [Kor21] that reduces the range avoidance problem with stretch $n+1$ to the range avoidance problem with stretch $2n$. We will only use this reduction (from stretch $n+1$ to stretch $2n$) as a black box, although our techniques depend heavily on the other reduction (from stretch $2n$ to an arbitrary stretch).

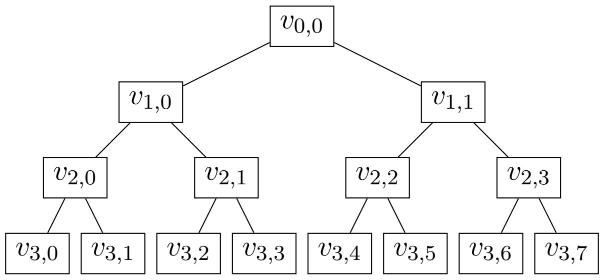

Figure 1: An illustration of a GGM tree of height 3.

Notice that on any fixed input $x \in \{0,1\}^n$, every vertex on the GGM tree contains an n-bit value. Hence, we call it a *fully-assigned* GGM tree. It is not hard to see that one can efficiently (in time linear in the height of the tree, $O(k)$) evaluate the assigned value at any vertex by traversing the tree and applying the circuit C at most k times. In other words, the outputs of a *fully-assigned* GGM tree, as truth tables, always have small circuit complexity.

Korten's reduction asserts that given any $f \in \{0,1\}^T \setminus \text{Range}(\mathsf{GGM}_T[C])$, there is a deterministic algorithm that given an NP oracle, finds a non-output of C and runs in time $\text{poly}(T,n)$. The algorithm is simple:

1. Set the assigned values of the leaves to be f.
2. Next, traverse the tree in a simple bottom up manner. That is, traverse the 2^{k-1} vertices on the $(k-1)$'th level one by one (say, from right to left), then proceed to the $(k-2)$'th level and so on, until reaching root.
3. For each interval vertex u traversed, assign v_u with the lexicographically first[8] n-bit string x such that $C(x)$ correctly evaluates to the assigned values of u's children. (Note that this step uses the NP oracle.)
4. Whenever such a string v_u does not exist, we successfully find a non-output of C (i.e., the assigned values of u's children). The algorithm now returns the non-output of C found and assigns $\perp$ to all remaining vertices.

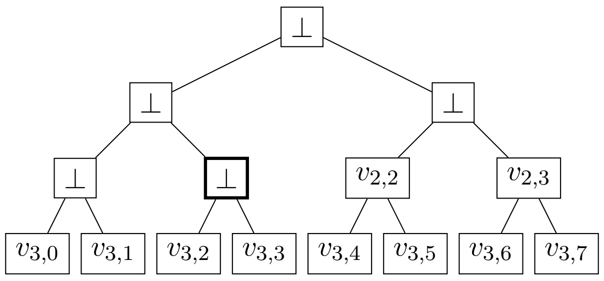

Figure 2: An illustration of the *partially-assigned* GGM tree from running Korten's algorithm.

[8]The original reduction from [Kor21] did not specify the lexicographically first requirement. This requirement is important for us in order to obtain a *single-valued* algorithm, and it comes for free given the NP oracle.

1.3.2 Computational history of the reduction

The computational history of Korten's reduction on a fixed input (C, f) consists of a *partially-assigned* GGM tree (i.e., some of the vertices are assigned $\perp$) and the vertex u where the reduction finds a non-output and halts. We are interested in the computational history because it has a few nice properties:

1. **Contains a Canonical Solution:** Notice that the execution of Korten's algorithm is fully deterministic. Hence, it produces the same non-output of C given the same f, and this non-output can be easily retrieved from the computational history.
2. **Locally Verifiable:** Every step of execution is very simple, making them locally verifiable. In other words, to verify any particular step of the execution, we only need to look at a constant number of assigned values on the *partially-assigned* GGM tree.

Moreover, by choosing $T = 2n \cdot 2^{2n}$, we know an f that is trivially not in the image of $\mathsf{GGM}_T[C]$: the concatenation of all $2n$-length strings.

1.3.3 A short description of the history

The downside of choosing $T = 2n \cdot 2^{2n}$ is that the size of the computational history is now exponential in n. The local verifiability of the history allows us to use a universal quantifier $(\forall)$ and an $O(\log T)$-bit variable to verify all $O(T)$ steps of the algorithm, but ultimately we need a short ($\mathrm{poly}(n)$) description of the computational history if we want to, for example build a $\mathsf{F\Sigma_2P}$ algorithm.

Here, the key observation is that, by changing the traversal order in Korten's reduction, the resulting computational history (in particular, the *partially-assigned* GGM tree) also has small circuit complexity! More specifically, if we change the traversal order to a post-order traversal (i.e., traverse the left subtree, then the right subtree, and finally the root), the resulting *partially-assigned* GGM tree can be decomposed into $O(n)$ smaller *fully-assigned* GGM trees. (See Figure 3 for an illustration where the roots of the *fully-assigned* GGM trees are drawn in circles.)

As such, we obtain a short description of the computational history: simply store the roots of all these $O(n)$ *fully-assigned* GGM tree.

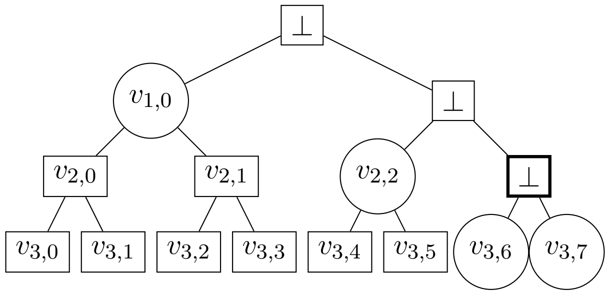

Figure 3: An illustration of the *partially-assigned* GGM tree from running modified Korten's algorithm.

1.3.4 Generalising to $\mathsf{FS_2P}$

The ideas above allow us to build a single-valued $\mathsf{F\Sigma_2P}$ algorithm for Avoid. In order to generalise it to a single-valued $\mathsf{FS_2P}$ algorithm, we need a selector algorithm that given two witnesses, picks the better one (in our case, the "more correct" description of the computational history). Now that we have a small description, this turns out to be an easy task. In particular, we can identify a single vertex with different assigned values in the two histories, and traverse down the tree until we hit the leaves, where we know the correct assigned value (i.e., the concatenation of all $2n$-length strings).

2 Preliminaries

We assume basic familiarity with computational complexity theory, such as complexity classes in the polynomial hierarchy (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 $T : \mathbb{N} \to \mathbb{N}$. 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 $\mathsf{S}_2\mathsf{P} \subseteq \Sigma_2\mathsf{P} \cap \Pi_2\mathsf{P}$.

2.1 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 $\mathsf{FS}_2\mathsf{P}$ 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.

A *search problem* Π maps every input $x \in \{0,1\}^*$ into a solution set $\Pi_x \subseteq \{0,1\}^*$. We say that a single-valued $\mathsf{F}\Sigma_2\mathsf{P}$ ($\mathsf{FS}_2\mathsf{P}$) 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 $1^{|x|}$.[9]

2.1.1 Single-valued $\mathsf{FS}_2\mathsf{P}$ and $\mathsf{F}\Sigma_2\mathsf{P}$ algorithms with $\mathsf{FP}^{\mathsf{NP}}$ post-processing

We also need the fact that single-valued $\mathsf{FS}_2\mathsf{P}$ or $\mathsf{F}\Sigma_2\mathsf{P}$ algorithms with $\mathsf{FP}^{\mathsf{NP}}$ post-processing can still be implemented by single-valued $\mathsf{FS}_2\mathsf{P}$ or $\mathsf{F}\Sigma_2\mathsf{P}$ algorithms, respectively. More specifically, we have:

Theorem 2.3. *Let $A(x)$ be a single-valued $\mathsf{FS}_2\mathsf{P}$ (resp. $\mathsf{F}\Sigma_2\mathsf{P}$) algorithm and $B(x, y)$ be an $\mathsf{FP}^{\mathsf{NP}}$ algorithm, both with fixed output length. The function $f(x) := B(x, A(x))$ also admits an $\mathsf{FS}_2\mathsf{P}$ (resp. $\mathsf{F}\Sigma_2\mathsf{P}$) algorithm.*

[9]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 $1^{|z|}$.

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 $a_1, \dots, a_n \in \{0,1\}^n$ 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 $\boldsymbol{\pi}_1$ and $\boldsymbol{\pi}_2$, where $\boldsymbol{\pi}_1$ consists of ω_1, acting as the second argument to V_A, and ℓ assignments $z_1^1, z_2^1, \dots, z_\ell^1 \in \{0,1\}^m$. Similarly, $\boldsymbol{\pi}_2$ consists of ω_2 and $z_1^2, z_2^2, \dots, z_\ell^2 \in \{0,1\}^m$.

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 $\boldsymbol{\pi}_\mu$ 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:

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 $Z^\mu = z_1^\mu \circ z_2^\mu \circ \dots \circ z_\ell^\mu$ 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 $\boldsymbol{\pi}_1$ or $\boldsymbol{\pi}_2$ be the corresponding proof for V_A concatenated with all lexicographically largest assignments for the $\{\phi_i\}_{i \in [\ell]}$. □

3 Modified 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. In fact, we will use a modified version of this reduction, which we introduce in this section.

Notation. Let s be an n-bit string. We use 0-indexing where s_0 denotes the first bit of s and s_{n-1} denotes the last bit of s. Let $i < j$, we use $s_{[i,j)}$ to denote the substring of s from the ith bit to the $(j-1)$th bit. We use $x \circ y$ to denote the concatenation of two strings x and y.

We consider perfect binary trees of height h. We identify each vertex in this tree with a tuple (i, j) where $i \in [0, h]$ and $j \in [0, 2^i - 1]$, indicating that the vertex is the j-th vertex on level i. Note that the two children of (i, j) are $(i+1, 2j)$ and $(i+1, 2j+1)$.

3.1 The GGM tree

Recall that the GGM tree construction from [GGM86] (vaguely speaking) increases the stretch of a circuit $C : \{0,1\}^n \to \{0,1\}^{2n}$ to arbitrarily long by applying C in a perfect binary tree manner.

Definition 3.1 (The GGM tree construction [GGM86])**.** Let $C : \{0,1\}^n \to \{0,1\}^{2n}$ 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 $\mathsf{GGM}_T[C] : \{0,1\}^n \to \{0,1\}^T$ 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 $0 \le j < 2^i$, denote $v_{i,j} \in \{0,1\}^n$ the value assigned to the vertex (i,j) (i.e., 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 $0 \le i < k, 0 \le j < 2^i$, 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 $C(v_{i,j})$).

Finally, we concatenate all values of the leaves and take the first T bits as the output:

$$\mathsf{GGM}_T[C](x) := (v_{k,0} \circ v_{k,1} \circ \cdots \circ v_{k,2^k-1})_{[0,T)} \,.$$

For our purposes, T is always set to $2n \cdot 2^{2n} = n \cdot 2^{2n+1}$. In other words, the GGM tree will always have height $h := 2n + 1$.

It is known that any output of the GGM tree is a truth table with small circuit complexity [Kor21]. Actually, given x and the label (i,j) of a node in the GGM tree, there is an efficient algorithm for computing the value assigned to this node.

Lemma 3.2. *Let* $\mathsf{GGMEval}(C,T,x,(i,j))$ *denote the n-bit assigned value $v_{i,j}$ in the evaluation of the GGM tree* $\mathsf{GGM}_T[C](x)$. *There is an algorithm running in $\widetilde{O}(|C| \cdot \log T)$ time that, given $C, T, x, (i,j)$, outputs* $\mathsf{GGMEval}(C,T,x,(i,j))$.

Proof sketch. To compute $v_{i,j}$, it suffices to traverse the GGM tree from the root to the vertex (i,j), applying the circuit C in each step. The running time is clearly bounded by the $\widetilde{O}(|C| \cdot \log T)$ since the GGM tree has height $O(\log T)$. □

3.2 Modifying Korten's reduction

It is shown in [Kor21] that the range avoidance problem for C reduces to the range avoidance problem for $\mathsf{GGM}_T[C]$. That is, given a hard truth table $f \notin \mathrm{Range}(\mathsf{GGM}_T[C])$ and an NP oracle, one can find a non-output for C in $\mathrm{poly}(T,n)$ time.

In what follows, we review this reduction and apply the following modification to it: instead of traversing the perfect binary tree in a simple bottom-up manner, we perform a *post-order traversal* (i.e., traverse the left subtree, then the right subtree and finally the root). Along the way we also define the *computational history* of "solving range avoidance of C from $\mathsf{GGM}_T[C]$" and show that this computational history always has a short description, which will be crucial to our proof. The modified reduction is depicted in Algorithm 1.

Fact 3.3. *In a post-order traversal, any root vertex of a subtree is traversed after all other vertices in the subtree. Any vertex in the right subtree is traversed after all vertices in the left subtree.*

For ease of presentation, we define the total order $<_P$ for all vertices on a perfect binary tree to be the post-order traversal order. In other words, $u_1 <_P u_2$ if and only if u_1 is traversed before u_2.

Fact 3.4. *Given two vertices $u_1 \neq u_2$ in a perfect binary tree, there is an algorithm that decides whether $u_1 <_P u_2$ or $u_2 <_P u_1$ and runs in time linear in the height of the tree.*

Proof sketch. The algorithm simply finds the least common ancestor u_a of u_1 and u_2. By definition of the least common ancestor, u_1 and u_2 cannot live in the same proper subtree of u_a.

The vertex living in the left subtree of u_a will be traversed first. If none of them lives in the left subtree of u_a, then one of them must be u_a itself. In this case, the vertex living in the right subtree will be traversed first. □

It is easy to see that this is indeed a reduction from the range avoidance problem for C to the range avoidance problem for $\mathsf{GGM}_T[C]$:

Algorithm 1 $\mathsf{Korten}'(C, f)$: Modified Korten's reduction

```
Input: $C : \{0,1\}^n \to \{0,1\}^{2n}$ 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: A perfect binary tree of height $k$ that contains the computational history.
1 for $j \leftarrow 0$ to $2^k - 1$ do
2     $v_{k,j} \leftarrow f_{[jn,(j+1)n)}$ ;                                  // set $f$ to the leaves
3 for vertices $(i, j)$ in the Post-Order Traversal do
4     Set $v_{i,j}$ be the lexicographically smallest string such that $C(v_{i,j}) = v_{i+1,2j} \circ v_{i+1,2j+1}$ ; // this step
        requires a NP oracle
5     if $v_{i,j}$ does not exist then
6         Set all remaining vertices $\perp$
7         return $v_{i+1,2j} \circ v_{i+1,2j+1}$
8 return $\perp$
```

Lemma 3.5. *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 1) returns 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 $f \in \{0,1\}^T \setminus \mathrm{Range}(\mathsf{GGM}_T[C])$. □

Finally, since every output of $\mathsf{GGM}_T[C]$ has small circuit complexity (as shown in Lemma 3.2), Algorithm 1 is also a reduction from the range avoidance problem (for C) to the problem of finding a hard truth table.

3.3 Computational history of $\mathsf{Korten}'(C, f)$

The computational history of $\mathsf{Korten}'(C, f)$ is essentially a *partially-assigned* perfect binary tree of height $2n + 1$ where each vertex (i, j) stores an n-bit string $v_{i,j}$ or $\perp$. To emphasise the tree structure of this computational history, let us call it $\mathsf{Histree}(C, f)$.

Definition 3.6 (The computational history of $\mathsf{Korten}'(C, f)$)**.** Let $n, T \in \mathbb{N}$ be such that $T = n \cdot 2^{2n+1}$. Let $C : \{0,1\}^n \to \{0,1\}^{2n}$ 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 $\mathsf{Histree}(C, f)$, is the partially-assigned perfect binary tree obtained by executing $\mathsf{Korten}'(C, f)$.

Let $h := \mathsf{Histree}(C, f)$. For any vertex u in the perfect binary tree, we use $h(u)$ to denote the value v_u stored at u. We use $c_L(u)$ and $c_R(u)$ to denote the left child and right child of u.

Definition 3.7 (Proper left children)**.** Given a set of vertices S from a binary tree, we define the set of proper left children of S to be:

$$\{u : \exists w \in S, c_L(w) = u, u \notin S\}.$$

The following lemma shows that h has a succinct description.

Lemma 3.8. *Let $n, T \in \mathbb{N}$ be such that $T = n \cdot 2^{2n+1}$. Let $C : \{0,1\}^n \to \{0,1\}^{2n}$ be a circuit, and $f \in \{0,1\}^T$. Let $h := \mathsf{Histree}(C, f)$, then h admits a unique description D_h such that:*

- $|D_h| \leq O(n) \cdot \log T$.
- *There is an algorithm* Eval *that takes as inputs D_h and any vertex (i,j), outputs $h(i,j)$ in time* $\mathrm{poly}(n) \cdot \log T$.

Proof. We start by describing D_h.

Let $u^* = (i^*, j^*)$ be the vertex where Algorithm 1 finds a solution and terminates. Let $S = \{u_0 = (0,0), u_1, u_2, \ldots, u^*\}$ be the set of vertices on the unique path starting from the root $(0,0)$ to u^*. Note that all vertices in S are assigned $\perp$.

D_h is defined to contain all *proper left children* of S and the right child of u^*, as well as all the stored values in these vertices. We further note that all these vertices have non $\perp$ stored values. In particular, consider any proper left children vertex u_L, it lives in the left subtree of its parent while u^* lives in the right subtree. So u_L must have already been traversed when the algorithm terminates.

Note that D_h contains both children of u^* and hence the output of $\mathsf{Korten}'(C, f)$. It is clear from how we construct D_h that $|D_h| \leq O(n) \cdot \log T$ since any path on the tree contains $O(\log T)$ vertices and every vertex stored in D_h carries $O(n)$ bits of information. Also, D_h is uniquely defined for any fixed h.

Next, we show how to efficiently evaluate $h(i,j)$ given any vertex (i,j) in the perfect binary tree. Notice that any subtree in h is also a (smaller) GGM tree. From Lemma 3.2, if we know the stored value of any ancestor of (i,j), we can efficiently (in time $\mathrm{poly}(n) \cdot \log T$) evaluate $h(i,j)$.

Therefore, it suffices to show that for any (i,j), one of its (non $\perp$) ancestors is stored in D_h:

1. If u^* is an ancestor of (i,j), then one of u^*'s children is an ancestor of (i,j) and we know both children are stored in D_h.
2. If (i,j) is an ancestor of u^*, then $h(i,j) = \perp$.
3. Otherwise, let u_a be the least common ancestor of u^* and (i,j), and we know that $u_a \in S$. If (i,j) falls in the left subtree of u_a, then the left child of u_a is an ancestor of (i,j) which is stored in D_h. If (i,j) falls in the right subtree of u_a then we argue that $h(i,j) = \perp$. This is because the algorithm stopped at u^* living in the left subtree of u_a, and would not have traversed (i,j).

This concludes the proof. □

The final ingredient we need is that D_h admits a Π_1 verifier on whether its corresponding computational history h is the correct one.

Lemma 3.9 (Π_1 verification of the history)**.** *Let $h := \mathsf{Histree}(C, f)$. There is an oracle algorithm V with input parameters T, n such that the following holds:*

1. *V takes $\tilde{f} \in \{0,1\}^T$ as an oracle and C, $\widetilde{D_h}$ and $w \in \{0,1\}^{2\log T + n}$ as inputs. It runs in* $\mathrm{poly}(n)$ *time.*
2. *D_h defined on $h := \mathsf{Histree}(C, f)$ is the unique string satisfying the following:*

$$\forall w \in \{0,1\}^{2\log T + n},\ V^f(C, D_h, w) = 1.$$

Proof. First, the oracle algorithm V parses $\widetilde{D_h}$ and checks whether it is indeed generated from a valid post-order traversal on the perfect binary tree. In particular, V reads the terminating vertex u^* based on the two children of u^* stored in $\widetilde{D_h}$, finds the path from $(0,0)$ to u^*, and checks that all proper left children of vertices on the path are included in $\widetilde{D_h}$ (of course the right child of u^* should also be included). V also checks that the stored values of these vertices are not $\perp$.

If this part of verification succeeds, we know $\widetilde{D_h}$ corresponds to some *partially-assigned* perfect binary tree $\tilde{h}$ and it remains to check that $\tilde{h}$ is the computational history $\mathsf{Histree}(C, f)$. One should think of the verifier making at most $2^{|w|}$ checks and accepts only if all $2^{|w|}$ check passes.

The verifier V makes the following checks. Note that whenever we need some value $v_{i,j}$, we can compute it by calling $\mathsf{Eval}(\widetilde{D_h}, (i, j))$. Recall that the total order $<_P$ is defined according to the post-order traversal sequence.

1. The values written on the leaves are indeed f. Hence, for every $j \in [0, 2^{2n+1} - 1]$, V checks that $v_{2n+1,j}$ is consistent with the corresponding string in f.
2. For every $(i, j) <_P u^*$, $C(v_{i,j}) = v_{i+1,2j} \circ v_{i+1,2j+1}$. (The values of each $v_{i,j}$ is consistent with its children.)
3. For every $(i, j) <_P u^*$, for every $x \in \{0, 1\}^n$ that is lexicographically smaller than $v_{i,j}$, $C(x) \neq v_{i+1,2j} \circ v_{i+1,2j+1}$. (The values of each $v_{i,j}$ is the lexicographically smallest possible one.)
4. Let $u^* = (i^*, j^*)$, then for every $x \in \{0, 1\}^n$, $C(x) \neq v_{i^*+1,2j^*} \circ v_{i^*+1,2j^*+1}$. (The two children of u^* form a non-output of C.)
5. For every (i, j) where $u^* \leq_P (i, j)$, $v_{i,j} = \perp$.

Each of the above checks is local (requires the assigned values of at most 3 vertices) and efficient (runs in time $\mathrm{poly}(n, \log T)$). There are in total $O(T)$ vertices and therefore $O(T)$ tests, which can be implemented with a universal ($\forall$) quantification over at most $2 \log T + n$ bits.

Clearly the correct history h (and therefore its unique description D_h) passes all these checks. Also these checks uniquely determine h as they are essentially enforcing every step of the execution of $\mathsf{Korten}'(C, f)$. □

4 Circuit lower bounds for $\mathsf{S_2E}$

4.1 Single-valued $\mathsf{F\Sigma_2P}$ algorithms for Avoid

We start by showing a simple single-valued $\mathsf{F\Sigma_2P}$ algorithm for Avoid.

Theorem 4.1. *There is a single-valued* $\mathsf{F\Sigma_2P}$ *algorithm A that given any circuit $C : \{0, 1\}^n \to \{0, 1\}^{2n}$ as input, $A(C)$ outputs y_C such that $y_C \notin \mathrm{Range}(C)$.*

Proof. On input a circuit $C : \{0, 1\}^n \to \{0, 1\}^{2n}$, let $T = 2n2^{2n}$ and $f \in \{0, 1\}^T$ be the concatenation of all $2n$-bit strings. Let $h = \mathsf{Histree}(C, f)$. The algorithm $V_A(C, \pi_1, \pi_2)$ is defined as follows: it parses π_1 as D_h and π_2 as w, simulates the verifier $V^f(C, D_h, w)$ in Lemma 3.9. It outputs the non-output of C stored in D_h iff $V^f(C, D_h, w) = 1$. Otherwise it outputs $\perp$.

Note that every position of f can be easily computed since it is just enumerating all $2n$-length strings. Hence the algorithm runs in ($\mathsf{F\Sigma_2}$-)polynomial time. The algorithm is single-valued because it can only output $\mathsf{Korten}'(C, f)$ which is a fixed string. □

4.2 Single-valued $\mathsf{FS_2P}$ algorithms for Avoid

In order to generalise the $\mathsf{F\Sigma_2P}$ algorithm above to a $\mathsf{FS_2P}$ algorithm, we need a 'selector' that chooses the correct D_h when two candidates are given. We formalise such selector in the following lemma.

Lemma 4.2. *Let $n, T \in \mathbb{N}$ be such that $T = 2n2^{2n}$. Let $C : \{0, 1\}^n \to \{0, 1\}^{2n}$ be a circuit, and $f \in \{0, 1\}^T$. Let $h := \mathsf{Histree}(C, f)$ and D_h be the succinct description of h defined in Lemma 3.8. Given f as an oracle and two strings π_1, π_2 as additional inputs, with the promise that $\pi_b = D_h$ for at least one $b \in \{0, 1\}$, there is a deterministic algorithm S running in $\mathrm{poly}(n) \cdot \log T$ time such that $S^f(C, \pi_1, \pi_2) = \pi_b$.*

Proof. S starts by parsing π_1, π_2 as D_h. If any of them fail to parse (i.e., the vertices are not derived from a post-order traversal), S simply discards it and output the other one.

Let h_1 and h_2 be the corresponding perfect binary tree generated from π_1 and π_2. Let (i_1^*, j_1^*) be the termination vertex in h_1 and (i_2^*, j_2^*) be the termination vertex in h_2. Then, S will efficiently find a single vertex that contains different stored values in h_1 and h_2. In particular, we consider two cases:

1. $(i_1^*, j_1^*) \neq (i_2^*, j_2^*)$: Without loss of generality, assume $(i_1^*, j_1^*) <_P (i_2^*, j_2^*)$. Then we know that $h_2(i_1^*, j_1^*) \neq \perp$.

2. $(i_1^*, j_1^*) = (i_2^*, j_2^*)$: Then they store the same set of vertices in π_1 and π_2, and one of the vertex must have a different stored value.

Now given a vertex (say u) where $h_1(u) \neq h_2(u)$, S proceeds as follows:

1. If u is a leaf vertex, then S checks it against f and decides which one is the correct D_h.

2. Otherwise, S checks if $C(h_1(u)) = C(h_2(u))$. If they are indeed pre-images of the same value, S picks the lexicographically smaller one.

   If $C(h_1(u)) \neq C(h_2(u))$ or if $h_2(u) \neq h_1(u) = \perp$, then at least one of u's children (say u') should have a different stored value. S then repeats the whole procedure on u'.

It is clear from the description that S terminates in $O(\log T)$ recursive steps, and the overall running time is $\mathrm{poly}(n) \cdot \log T$. ☐

Theorem 4.3. *There is a single-valued* $\mathsf{FS_2P}$ *algorithm A that given any circuit $C : \{0,1\}^n \to \{0,1\}^{2n}$ as input, $A(C)$ outputs y_C such that $y_C \notin \mathrm{Range}(C)$.*

Proof. On input a circuit $C : \{0,1\}^n \to \{0,1\}^{2n}$, let $T = 2n2^{2n}$ and $f \in \{0,1\}^T$ be the concatenation of all $2n$-bit strings. Let $h = \mathsf{Histree}(C, f)$. The algorithm $V_A(C, \pi_1, \pi_2)$ applies the selector algorithm $S^f(C, \pi_1, \pi_2)$ from Lemma 4.2 to obtain $\pi_b = D_h$, and then returns the non-output of C stored in D_h.

Again, every position of f can be easily computed since it is just enumerating all $2n$-length strings, hence the algorithm runs in ($\mathsf{FS_2}$-)polynomial time. The algorithm is single-valued because it can only output $\mathsf{Korten}'(C, f)$ which is a fixed string. ☐

The algorithm in Theorem 4.3 only works when the input circuit has stretch $n \mapsto 2n$. Fortunately, Korten [Kor21] showed a $\mathsf{P^{NP}}$ reduction for the range avoidance problem from stretch $n \mapsto (n+1)$ to stretch $n \mapsto 2n$:

Lemma 4.4 ([Kor21, Lemma 3]). *Let $n \in \mathbb{N}$. There is a polynomial time algorithm A and an* $\mathsf{FP^{NP}}$ *algorithm B such that the following holds:*

1. *Given a circuit $C : \{0,1\}^n \to \{0,1\}^{n+1}$, $A(C)$ outputs a circuit $D : \{0,1\}^n \to \{0,1\}^{2n}$.*

2. *Given any $y \in \{0,1\}^{2n} \setminus \mathrm{Range}(D)$, $B(C, y)$ outputs a string $z \in \{0,1\}^{n+1} \setminus \mathrm{Range}(C)$.*

Corollary 4.5. *There is a single-valued* $\mathsf{FS_2P}$ *algorithm A that given any polynomial-sized circuit $C : \{0,1\}^n \to \{0,1\}^{n+1}$ as input, $A(C)$ outputs y_C such that $y_C \notin \mathrm{Range}(C)$.*

Proof Sketch. It follows from Lemma 4.4 and Theorem 4.3 that there is a single-valued $\mathsf{FS_2P}$ algorithm with $\mathsf{FP^{NP}}$ post-processing that solves the range avoidance problem (for stretch $n \mapsto n+1$). By Theorem 2.3, such an algorithm is (still, simply) a single-valued $\mathsf{FS_2P}$ algorithm. ☐

4.3 Circuit lower bounds

Finally, we prove our near-maximum circuit lower bound for $\mathsf{S_2E}$. It suffices to apply our range avoidance algorithm to the *truth table generator* $\mathsf{TT}_{n,s}$. Frandsen and Miltersen [FM05] showed 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 here (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 $L_{n,s} < 2^n$.

Theorem 4.6. $\mathsf{S_2E} \not\subset$ i.o.-$\mathsf{SIZE}[2^n/n]$.

Proof Sketch. Let A be the single-valued algorithm from Corollary 4.5 and set $s := 2^n/n$. Define the language $\mathcal{L}$ such that the truth table of the characteristic function of $\mathcal{L} \cap \{0,1\}^n$ is $A(\mathsf{TT}_{n,s})$. By our choice of s, $L_{n,s} = (s+1)(7+\log(n+s)) < 2^n$ and hence $\mathsf{TT}_{n,s}$ is a valid Avoid instance.

Since every s-size n-input circuit C can be encoded into a stack program of description size $L_{n,s}$ bits [FM05], and $A(\mathsf{TT}_{n,s}) \notin \mathrm{Range}(\mathsf{TT}_{n,s})$, we have that $\mathcal{L} \notin$ i.o.-$\mathsf{SIZE}[s(n)]$. On the other hand, since the truth table of $\mathcal{L}$ can be computed by a single-valued $\mathsf{FS_2P}$ algorithm A, we have $\mathcal{L} \in \mathsf{S_2E}$. □

Remark 4.7. Finally, it is easy to see that all our results above relativise.

Acknowledgments

Part of the work was done when L. Chen, S. Hirahara, and H. Ren 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. Zeyong Li is supported by the NUS-NCS Joint Laboratory for Cyber Security, Singapore. Hanlin Ren received support from DIMACS through grant number CCF-1836666 from the National Science Foundation. We thank Eldon Chung, Karthik Gajulapalli, Alexander Golovnev, Oliver Korten, Zhenjian Lu, Igor C. Oliveira, Rahul Santhanam, Sidhant Saraogi, Noah Stephens-Davidowitz, Roei Tell, and Ryan Williams for helpful discussions. We also want to thank Divesh Aggarwal, Alexander Golovnev, Jiatu Li, Igor C. Oliveira, Sidhant Saraogi, and Roei Tell for comments on earlier versions 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. 2

[AB09] Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, 2009. doi:10.1017/CBO9780511804090. 8

[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. 2, 18

[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. 18

[BFT98] Harry Buhrman, Lance Fortnow, and Thomas Thierauf. Nonrelativizing separations. In *CCC*, pages 8–12, 1998. doi:10.1109/CCC.1998.694585. 2

[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

[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. 2, 3, 8, 18

[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. 8

[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. 2

[CHR24] Lijie Chen, Shuichi Hirahara, and Hanlin Ren. Symmetric exponential time requires near-maximum circuit size. In *STOC*, pages 1990–1999. ACM, 2024. doi:10.1145/3618260.3649624. 1

[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. 18

[CT21] 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. 4

[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. 3

[Erd59] Paul Erdős. Graph theory and probability. *Canadian Journal of Mathematics*, 11:34–38, 1959. doi:10.4153/CJM-1959-003-9. 3

[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, 5, 15

[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, 5

[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. 5, 9

[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. 3

[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. 3

[Gol08] Oded Goldreich. *Computational complexity: a conceptual perspective*. Cambridge University Press, 2008. doi:10.1017/CBO9780511804106. 8

[HLR23] Shuichi Hirahara, Zhenjian Lu, and Hanlin Ren. Bounded relativization. In *CCC*, volume 264 of *LIPIcs*, pages 6:1–6:45. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPIcs.CCC.2023.6. 2, 4

[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. 18

[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. 2

[Jeř04] Emil Jeřábek. Dual weak pigeonhole principle, Boolean complexity, and derandomization. *Ann. Pure Appl. Log.*, 129(1-3):1–37, 2004. doi:10.1016/j.apal.2003.12.003. 3

[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. 2

[KC00] Valentine Kabanets and Jin-Yi Cai. Circuit minimization problem. In *STOC*, pages 73–79, 2000. doi:10.1145/335305.335314. 2

[KKMP21] Robert Kleinberg, Oliver Korten, Daniel Mitropolsky, and Christos Papadimitriou. Total functions in the polynomial hierarchy. In *ITCS*, volume 185 of *LIPIcs*, pages 44:1–44:18, 2021. doi:10.4230/LIPIcs.ITCS.2021.44. 3, 5

[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, 18

[Kor21] Oliver Korten. The hardest explicit construction. In *FOCS*, pages 433–444. IEEE, 2021. doi:10.1109/FOCS52979.2021.00051. 3, 4, 5, 6, 9, 10, 14

[Kra01] Jan Krajíček. Tautologies from pseudo-random generators. *Bull. Symb. Log.*, 7(2):197–212, 2001. doi:10.2307/2687774. 3

[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. 9

[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. 3

[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. 2

[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. 18

[Li23] Xin Li. Two source extractors for asymptotically optimal entropy, and (many) more. In *FOCS*, pages 1271–1281. IEEE, 2023. doi:10.1109/FOCS57990.2023.00075. 3

[Li24] Zeyong Li. Symmetric exponential time requires near-maximum circuit size: Simplified, truly uniform. In *STOC*, pages 2000–2007. ACM, 2024. doi:10.1145/3618260.3649615. 1

[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. 2, 5

[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. 2

[RS98] Alexander Russell and Ravi Sundaram. Symmetric alternation captures BPP. *Comput. Complex.*, 7(2):152–162, 1998. doi:10.1007/s000370050007. 3, 8

[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. 3, 4

[San09] Rahul Santhanam. Circuit lower bounds for Merlin–Arthur classes. *SIAM J. Comput.*, 39(3):1038–1061, 2009. doi:10.1137/070702680. 2

[Sel94] Alan L. Selman. A taxonomy of complexity classes of functions. *J. Comput. Syst. Sci.*, 48(2):357–381, 1994. doi:10.1016/S0022-0000(05)80009-1. 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

[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. 3

[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. 2

[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. 2, 4

A Karp–Lipton collapses and the half-exponential barrier

In the following, we elaborate on the half-exponential barrier mentioned in the introduction. 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.

Let $\mathcal{C}$ be a "typical" uniform complexity class containing P, a *Karp–Lipton collapse* to $\mathcal{C}$ states that if a large class (say EXP) has polynomial-size circuits, then this class collapses to $\mathcal{C}$. For example, there is a Karp–Lipton collapse to $\mathcal{C} = \Sigma_2\mathsf{P}$:

Suppose $\mathsf{EXP} \subseteq \mathsf{P}/_{\mathrm{poly}}$, then $\mathsf{EXP} = \Sigma_2\mathsf{P}$. ([KL80], attributed to Albert Meyer)

Now, assuming that $\mathsf{EXP} \subseteq \mathsf{P}/_{\mathrm{poly}} \implies \mathsf{EXP} = \mathcal{C}$, the following win-win analysis implies that $\mathcal{C}$-EXP, the exponential-time version of $\mathcal{C}$, is not in $\mathsf{P}/_{\mathrm{poly}}$: (1) if $\mathsf{EXP} \not\subset \mathsf{P}/_{\mathrm{poly}}$, then of course $\mathcal{C}$-$\mathsf{EXP} \supseteq \mathsf{EXP}$ does not have polynomial-size circuits; (2) otherwise $\mathsf{EXP} \subseteq \mathsf{P}/_{\mathrm{poly}}$. We have $\mathsf{EXP} = \mathcal{C}$ and by padding $\mathsf{EEXP} = \mathcal{C}$-$\mathsf{EXP}$. Since EEXP contains a function of maximum circuit complexity by direct 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.[10]

The half-exponential barrier. The above argument is very successful at proving various super-polynomial lower bounds. However, a closer look shows that it is only capable of proving *sub-half-exponential* circuit lower bounds. Indeed, suppose we want to show that $\mathcal{C}$-EXP does not have circuits of size $f(n)$. We will have to perform the following win-win analysis:

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

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