Electronic Colloquium on Computational Complexity, Report No. 113 (2024)

On Oracles and Algorithmic Methods for Proving Lower Bounds

Nikhil Vyas * Ryan Williams †

July 4, 2024

Abstract

This paper studies the interaction of oracles with algorithmic approaches to proving circuit complexity lower bounds, establishing new results on two different kinds of questions.

1. We revisit some prominent open questions in circuit lower bounds, and provide a clean way of viewing them as circuit *upper bound* questions. Let Missing-String be the (total) search problem of producing a string that does not appear in a given list L containing M bit-strings of length N, where $M < 2^n$. We show in a generic way how algorithms and uniform circuits (from restricted classes) for Missing-String imply complexity *lower* bounds (and in some cases, the converse holds as well).

• We give a *local* algorithm for Missing-String, which can compute any desired output bit making very few probes into the input, when the number of strings M is small enough. We apply this to prove a new nearly-optimal (up to oracles) time hierarchy theorem with advice.

• We show that the problem of constructing restricted uniform circuits for Missing-String is essentially equivalent to constructing functions without small *non-uniform* circuits, in a relativizing way. For example, we prove that small uniform depth-3 circuits for Missing-String would imply exponential circuit *lower bounds* for $\Sigma_2\mathsf{EXP}$, and depth-3 lower bounds for Missing-String would imply non-trivial circuits (relative to an oracle) for $\Sigma_2\mathsf{EXP}$ problems. Both conclusions are longstanding open problems in circuit complexity.

2. We construct an oracle relative to which SAT can be solved in "half-exponential" time, yet exponential time (EXP) has polynomial-size circuits. Improving EXP to NEXP would give an oracle relative to which $\Sigma_2\mathsf{E}$ has "half-exponential" size circuits, which is open. (Recall it is known that $\Sigma_2\mathsf{E}$ is not in "sub-half-exponential" size, and the proof relativizes.) Moreover, the running time of the SAT algorithm cannot be improved: relative to all oracles, if SAT is in "sub-half-exponential" time then EXP does not have polynomial-size circuits.

The conference version of this paper had an incorrect theorem (Theorem 1.10), pointed out to us by Jiatu Li. We have attached an erratum (coauthored with Jiatu Li) to the end of this paper, which includes a proof of the negation of Theorem 1.10.

1 Introduction

Which complexity classes have small-size non-uniform circuit families, and which do not? While this question dates back to the 1970s [22, 13, 21], our known answers are astoundingly incomplete. We believe that NP does not have polynomial-size circuit families, but our best-known results have to replace the class NP with considerably larger complexity classes. Kannan [13] proved that $\Sigma_2\mathsf{EXP}$

*SEAS, Harvard University, USA. Work done while the author was a PhD student at MIT, supported by NSF CCF-1909429 and NSF CCF-2127597.

†MIT CSAIL and EECS, USA. Supported by NSF CCF-1909429 and NSF CCF-2127597.

ISSN 1433-8092

(the exponential-time version of $\Sigma_2\mathsf{P}$) does not have polynomial-size circuits, and his proof extends to "sub-half-exponential" size circuits, i.e., for size functions $H(n)$ where $H(H(n)^c)^c < o(2^n/n)$ for a fixed constant $c \geq 1$ [18]. Buhrman, Fortnow, and Theirauf [7] proved that the exponential-time version of Merlin-Arthur (MAEXP) does not have polynomial-size circuits. However, it remains open whether NEXP (nondeterministic exponential time) has polynomial-size circuits, in spite of research attempting to attack the problem via faster circuit-analysis algorithms [11, 12, 26]. Perhaps even more embarrassingly, there could be *infinitely many input lengths* for which even $\Sigma_2\mathsf{EXP}$ and MAEXP problems have polynomial-size circuits on just those input lengths. Furthermore, there is no known barrier to establishing stronger lower bounds for $\Sigma_2\mathsf{EXP}$—not even a relativization barrier. For example, according to current knowledge, there could be a *relativizing* proof (one that holds with respect to all oracles) that $\Sigma_2\mathsf{EXP}$ requires *maximum* ($\Theta(2^n/n)$) circuit complexity, *or* there could be an oracle under which $\Sigma_2\mathsf{EXP}$ has at most 2^{n^ε} circuit complexity, for all $\varepsilon > 0$.[1] (Note that we could not have both.)

We propose a clean way of studying such lower bound questions, as circuit *upper bound* questions. We begin by defining a simple search problem over binary strings.

> Missing-String: Let N, M be such that $M < 2^N$. Given a list L of M strings which are N-bits long, output an N-bit string y that is not in L.

We call such a string y a *missing string* for L. Missing-String is a natural and classic problem arising (for example) whenever one wishes to assign a "name" to a new resource in a network.[2] Note that Missing-String is a *total* search problem; since $M < 2^N$, every instance has a solution.[3] Building on long-known connections showing how circuit lower bounds lead to oracles separating complexity classes [10], we show that the above circuit lower bound questions (and more) can be viewed as questions about how efficiently Missing-String can be solved, giving *equivalences* between algorithms for Missing-String and circuit lower bounds. In particular, the depth-3 circuit complexity of Missing-String is innately tied to the question of whether $\Sigma_2\mathsf{EXP}$ has functions with exponential circuit complexity.

1.1 Local Algorithms for Missing-String and New Time Hierarchies

First, we study how well Missing-String can be solved by algorithms which only probe a few bits of the list in order to determine an output bit, and use our results to derive new time hierarchy theorems with advice that are very close to tight (relative to oracles).

As a starting point, observe that when $M \leq N$ (the number of strings is at most the length of the strings) Cantor's diagonal argument shows that one can make only 1 probe to determine each output bit of a missing string. We consider natural generalizations of this fundamental algorithm. Let $K \geq 1$ be a small parameter. How many probes are needed to compute a missing string, when

[1] Using the oracle that makes $\mathsf{P} = \mathsf{NP}$, there is an oracle relative to which $\Sigma_2\mathsf{EXP}$ requires maximum circuit complexity. However, we don't know any oracles relative to which $\Sigma_2\mathsf{EXP}$ has smaller circuit complexity.

[2] In the literature, the representation is often a list of integers, and the problem is called the "missing integer" problem. There is a well-known linear time algorithm (see for example Chapter 1 of [6]) which is apparently a common tech interview problem.

[3] One may view Missing-String as an "exponential-sized" version of the Empty problem, studied in [16, 17, 19]: given a circuit $C : \{0,1\}^m \to \{0,1\}^N$ where $N > m$, find an N-bit string y such that no x has $C(x) = y$. In Empty, one is given a (small) circuit with m outputs whose *truth table* is the list of strings, and the task is to find a missing string in the list.

$M \leq KN$? In Section 3, we give an algorithm for MISSING-STRING that is *local* in that each output bit of the missing string can be computed separately in about $O(K \log^2 K)$ time, making only $O(K \log K)$ probes into the list L, provided that $M \leq KN$. (It is easy to show that $\Omega(K)$ probes are required; see Appendix A.) It is an interesting open problem to close the gap between our $O(K \log K)$ upper bound and the $\Omega(K)$ lower bound.

Applying our MISSING-STRING algorithm directly, we prove a new relativizing time hierarchy theorem against *super-linear* advice. The most generic form of our hierarchy is the following.

Theorem 1.1. *For all (time-constructible) $\alpha(n) \geq 0$, $t(n) \geq n^2$, unbounded $\beta(n)$, and every oracle A,*

$$\mathsf{TIME}^A[t(n) \log t(n) \cdot (\alpha(n) + \beta(n)) \cdot 2^{\alpha(n)+\beta(n)}] \not\subset \text{i.o.-}\mathsf{TIME}^A[t(n)]/(n + \alpha(n)).$$

That is, there are functions in time $O(t(n) \log t(n) \cdot (\alpha(n)+\beta(n)) \cdot 2^{\alpha(n)+\beta(n)})$ that cannot be simulated in time $t(n)$ even with $n + \alpha(n)$ non-uniform advice, for all but finitely many input lengths n.

The above theorem holds for "reasonable" models of computation such as multitape Turing machines, random access Turing machines, random access machines, and so on. Previously, it was known that $\mathsf{TIME}(t(n) \log^2 t(n)) \not\subset \text{i.o.-}\mathsf{TIME}(t(n))/(n - \omega(1))$ for appropriate time bounds $t(n)$ [23, 9]. The basic idea in such proofs is to use the input itself as potential advice in the hard function; however, this strategy can only possibly diagonalize against advice of length *less than* n, whereas the lower bound in Theorem 1.1 holds for machines with advice longer than n. It was also well-known that by running for at least $\Omega(2^{\alpha(n)})$ time, one could enumerate all $\alpha(n)$-bit advice strings, and diagonalize against $\alpha(n)$-bit advice machines (see e.g.,Theorem 3 of [11]). Our Theorem 1.1 adds a factor of n to the advice length of such time hierarchies. In contrast, Theorem 1.1 is interesting in that it is not proved by *directly* diagonalizing over algorithms with $n + \alpha(n)$ advice: on any given input, the hard function in Theorem 1.1 has to simulate multiple algorithms on multiple advice strings and inputs, in order to determine its output.

Theorem 1.1 has several interesting corollaries, illustrating tradeoffs between running time and advice. As time hierarchies are so fundamental to complexity lower bounds, we believe these new hierarchies may be useful in future separations.

Corollary 1.2. *For all $c \geq 1$, $\mathsf{TIME}[2^n \cdot n^{c+3}] \not\subset \text{i.o.-}\mathsf{TIME}[2^n]/(n + c \log n)$.*

Corollary 1.3. *For all $\varepsilon > 0$, $\mathsf{TIME}[n^3 \cdot 2^{2\varepsilon n}] \not\subset \text{i.o.-}\mathsf{TIME}[2^{\varepsilon n}]/((1 + \varepsilon)n)$.*

Corollary 1.4. *For every c, $\mathsf{TIME}[n^{2c} \log^3 n] \not\subset \text{i.o.-}\mathsf{TIME}[n^c]/(n + c \log n)$.*

For example, Corollary 1.3 says there are problems solvable in about $2^{2\varepsilon n}$ time which cannot be solved in $2^{\varepsilon n}$ time with $(1 + \varepsilon)n$ advice: we can compute a hard function in significantly less time than the number of possible advice strings ($2^{n+\varepsilon n}$) that we must diagonalize against.

The lower bound of Theorem 1.1 is essentially tight, in that there are oracles relative to which the containment actually holds if the running time of the hard function is slightly decreased [27] (see also Appendix A).

Good Circuits for Missing-String Are Equivalent to Circuit Lower Bounds. Next, we show that the problem of constructing good uniform circuits for MISSING-STRING is *equivalent* to constructing relativizing circuit lower bounds for uniform complexity classes. First, we give an equivalence between almost-everywhere circuit lower bounds and circuits for MISSING-STRING.

To keep the presentation clean, we start with stating our results for $\Sigma_k\mathsf{E} = \Sigma_k\mathsf{TIME}[2^{O(n)}]$, but our connection holds for *any* exponential-time complexity class, with appropriate modifications for Missing-String circuits (See Theorem 1.7). Recall that $\mathsf{SIZE}^A(s(n))$ is the class of functions with an A-oracle circuit family of size $O(s(n))$, and i.o.-$\mathsf{SIZE}^A(s(n))$ is the class of functions having A-oracle circuits of size $O(s(n))$ on *infinitely many* input lengths n.

Theorem 1.5 (Equivalence of Missing-String Lower Bounds and Circuit Upper Bounds)**.** *For all k, the following are equivalent:*

- *For every oracle A, $\Sigma_k\mathsf{E}^A \not\subset$ i.o.-$\mathsf{SIZE}^A(\tilde{O}(s(O(n))))$.*
- *For all n, Missing-String has a poly(N)-time uniform depth-$(k+1)$ circuit family of $2^{poly(N)}$ size and poly(N) bottom fan-in, on all lists of length $M := 2^{\tilde{O}(s(O(\log N)))}$ with $N = 2^n$.*[4]

According to current knowledge, Theorem 1.5 only refers to an open problem in the case of $k = 2$. The other values of k are already settled, and Theorem 1.5 implies interesting results about the Missing-String problem in those cases:

- For $k = 1$, it is known ([27]) that there is an oracle A such that $\mathsf{NE}^A = \mathsf{NTIME}^A[2^{O(n)}] \subset \mathsf{SIZE}^A(O(n))$. Theorem 1.6 implies that Missing-String *cannot* be solved by depth-2 circuits of quasi-polynomial size, even for $M \leq 2^{\log^{1.1}(N)}$.
- For $k \geq 3$, recall it is known ([13, 18]) that for all oracles A, $\Sigma_3\mathsf{E}^A$ has *maximum* A-oracle circuit complexity almost everywhere (that is, $\Sigma_3\mathsf{E}^A \not\subset$ i.o.-$\mathsf{SIZE}(H(n) - 1)$, where $H(n) = (1 \pm o(1))2^n/n$ denotes the maximum circuit complexity of n-bit Boolean functions). This directly corresponds to the existence of uniform depth-4 circuits solving Missing-String for all $M < 2^N$.

Recall it is open whether $\Sigma_2\mathsf{E} \subset$ i.o.-$\mathsf{SIZE}(\tilde{O}(n))$ (even relative to some oracle). Theorem 1.5 shows that the problem of proving "almost-everywhere" (relativizing) lower bounds for $\Sigma_2\mathsf{E}$ is equivalent to constructing uniform depth-3 circuits for Missing-String over lists of length $M = 2^{\tilde{O}(\log N)}$.

Similarly to Theorem 1.5, we can show that constructing uniform circuits for Missing-String that succeed on only infinitely many bit-string lengths $N = 2^n$ would still imply circuit lower bounds.

Theorem 1.6 (Circuits for Missing-String Imply Circuit Lower Bounds)**.** *Let $k \geq 1$. If for infinitely many n, Missing-String has a poly$(\log N)$-time uniform depth-$(k+1)$ circuit family of $2^{poly(N)}$ size and poly(N) bottom fan-in, on all lists of length $M := 2^{\tilde{O}(s(\log N))}$ with strings of length $N = 2^n$, then $\Sigma_k\mathsf{E}^A \not\subset \mathsf{SIZE}^A(s(n))$ for all oracles A.*

As mentioned above, the proofs of Theorems 1.5 and 1.6 easily generalize to other exponential-time complexity classes (and correspondingly, other circuit types for Missing-String). Let us briefly describe what the generalization looks like. For $t \geq 0$, let $Y_1, \ldots, Y_{2^t}$ be the list of all t-bit strings in lexicographical order. For an arbitrary $G : \{0,1\}^\star \to \{0,1\}$, we define the complexity

[4] Our notion of uniformity is the typical "direct connect" uniformity; see the Preliminaries (Section 2) for more details.

class G-E, where a decision problem f is in $G\mathsf{E}$ if and only if there is a constant $c > 0$ and Turing machine $M(x, y)$ running in 2^{cn} time when $|x| = n$ and $|y| = 2^{cn}$, such that for all x,

$$f(x) = 1 \iff G(M(x, Y_1), \ldots, M(x, Y_{2^{2^{cn}}})) = 1.$$

That is, the value of $f(x)$ is determined by computing G on a doubly-exponential length input, and this input is computed by evaluating $M(x, \cdot)$ on all possible strings of length 2^{cn}. It is easy to see that OR-$\mathsf{E} = \mathsf{NE}$, AND-$\mathsf{E} = \mathsf{co\text{-}NE}$, $\Sigma_2\mathsf{E}$ is equivalent to a case where G is an OR of AND, and so on.

Theorem 1.7 (Generic Equivalence)**.** *The following are equivalent:*

- *For every oracle A, G-$\mathsf{E}^A \not\subset$ i.o.-$\mathsf{SIZE}^A(\tilde{O}(s(O(n))))$.*
- *For all n,* Missing-String *has a poly(N)-time uniform circuit family of $2^{poly(N)}$ size on all lists of length $M := 2^{\tilde{O}(s(O(\log N)))}$ with $N = 2^n$, where the output gate of each circuit is a copy of G, and the inputs to G are decision trees of depth poly(N).*

Non-Relativizing Circuit Lower Bounds as Circuits for Missing-String on "Easy" Inputs. Although no relativization barriers are known against circuit lower bounds for $\Sigma_2\mathsf{EXP}$, it would be natural for a complexity theorist to be unsettled by the relativizing nature of the previous theorems. Our perspective also yields insight into the precise difference between non-relativizing circuit lower bounds (e.g. $\mathsf{NEXP} \not\subset \mathsf{SIZE}(n^{O(1)})$, $\mathsf{MAEXP} \not\subset \mathsf{SIZE}(n^{O(1)})$ [7]) and relativizing ones, by viewing these questions in terms of low-depth circuits for Missing-String.

Let L be an ordered list of $M = 2^t$ bit strings, each of length $N = 2^n$. We say that L is $S(n)$-*compressible* if there is a circuit C of at most $S(n)$ size, with $t + n$ inputs, such that the 2^{t+n}-bit truth table of C corresponds exactly to L. In other words, L is $S(n)$-compressible if there is an $S(n)$-size circuit encoding it. We observe that constructing low-depth circuits for Missing-String on $S(n)$-compressible inputs corresponds directly to (non-relativizing) circuit lower bounds. We illustrate the idea with the case of $\mathsf{NE} := \mathsf{NTIME}[2^{O(n)}]$.

Theorem 1.8. *Let $S(n) \geq n$. $\mathsf{NE} \not\subset \mathsf{SIZE}(S(n))$ if and only if there is a poly(N)-time uniform depth-two circuit family of size $2^{poly(N)}$ for* Missing-String *that succeeds for infinitely many $N = 2^n$, when $M = 2^{O(S(n)\log S(n))}$ and the list L is $S(n)$-compressible.*

As noted above, since there is an oracle relative to which $\mathsf{NE} \subset \mathsf{SIZE}(O(n))$, Theorem 1.6 shows that Missing-String on $M = 2^{\text{poly}(\log N)}$ strings cannot be solved *on general inputs* with depth-two circuits of $2^{\text{poly}(N)}$ size. Theorem 1.8 says that, if we believe $\mathsf{NE} \not\subset \mathsf{SIZE}(n^{O(1)})$ is true (and we do!), then we should believe that Missing-String *can* be solved with small *uniform* depth-two circuits on $n^{O(1)}$-compressible inputs. The fact that compressible inputs make the problem easier is counterintuitive; many complexity theorists believe that separations like $\mathsf{EXP} \neq \mathsf{NEXP}$ are just as likely as separations like $\mathsf{P} \neq \mathsf{NP}$, but the EXP vs NEXP problem is exactly the P versus NP problem restricted to highly compressible inputs. In our setting, the difference between non-relativizing circuit lower bounds versus relativizing ones is precisely the difference between solving Missing-String on highly-compressible inputs versus general, arbitrary inputs (oracles).

1.2 A Non-Relativizing Aspect of the Algorithmic Method

The second result in this paper concerns proving circuit complexity lower bounds via the design of faster SAT algorithms. Are there oracles relative to which SAT can be solved more efficiently, yet no circuit lower bounds follow? It is known (e.g., [26], Proposition 2) that if SAT is in $O(H(n))$ time then $\mathsf{EXP} \not\subset \mathsf{P}/\text{poly}$, for any "sub-half-exponential" $H(n)$ satisfying $H(H(n^k)^2) \leq 2^n$ for all k, and that proof relativizes.[5] Could one conclude EXP lower bounds from a SAT algorithm that runs just slightly slower? We give an oracle relative to which this is not true:

Theorem 1.9. *There is an oracle A and a function H such that $\mathsf{TIME}^A[2^n] \subset \mathsf{SIZE}^A[O(n)]$, SAT^A can be solved in poly$(H(poly(n)))$ time with an A-oracle, $H(H(n)) \leq 2^n$ and H is monotone increasing.*

Could a slower SAT algorithm imply NEXP circuit lower bounds instead? If we could replace EXP with NEXP in Theorem 1.9, that would yield an oracle relative to which $\Sigma_2\mathsf{E}$ has "half-exponential" size circuits, which is open. (Recall it is known that $\Sigma_2\mathsf{E}$ is not in "sub-half-exponential" size, and the proof relativizes.) That is, our oracle stops just short of showing that non-relativizing methods are necessary for proving stronger $\Sigma_2\mathsf{E}$ circuit lower bounds. While half-exponential functions have been known to pop up naturally in circuit complexity for over 20 years [18], Theorem 1.9 is the first oracle result (to our knowledge) that "explains" the half-exponential barrier. We leave it as interesting open problems to show an algebrization barrier for similar settings.

The previous version of the paper stated the following *incorrect* theorem. We have attached the proof of negation of Theorem 1.10 to the end of this paper.

Theorem 1.10. *There is an oracle A such that CAPP (for A-oracle circuits) is in P^A but $\mathsf{EXP}^{\mathsf{NP}^A}$ has polynomial-size A-oracle circuits.*

2 Preliminaries

We assume familiarity with complexity theory, especially notions such as the polynomial hierarchy and non-uniform circuit families [4]. Here, we recall some particularly important notions for this work.

We use $\mathbb{h} : \mathbb{N} \to \mathbb{N}$ to denote a "half-exponential" function, i.e., for all n, $\mathbb{h}(n)$ is a strictly monotone increasing function satisfying $\mathbb{h}(\mathbb{h}(n)) = 2^n$.[6] It is widely believed (and sometimes claimed) that there exist $\mathbb{h}(n)$ which are time constructible [18, 2, 1].[7] For a function $f : \{0,1\}^\star \to \{0,1\}$, we let $f_n : \{0,1\}^n \to \{0,1\}$ denote the *n-th slice of f*, i.e., the function f restricted to n-bit inputs.

We say that a function $s : \mathbb{N} \to \mathbb{N}$ is *nice* if it satisfies $s(n + \log(n)) \leq \tilde{O}(s(n)) = s(n) \cdot \text{poly}(\log s(n))$. We note that most commonly studied functions such as $cn, n^k, n^{\log^k(n)}, 2^{n^\varepsilon}, 2^{\varepsilon n}$ are all nice. Unless otherwise specified, we will assume all functions s are nice.

[5] See Appendix B for a proof outline.

[6] Note that such a function must be one-to-one, and therefore strictly increasing: if $h(n) = h(m) = z$ for some $n \neq m$, then $h(z) = h(h(n) = 2^n$ and $h(z) = h(h(m)) = 2^m$, a contradiction.

[7] We have not yet found a rigorous proof in the literature, but we have no reason to doubt its existence, given the vast literature on computing half-exponential functions.

Uniform Circuits. In Theorems 1.6 and 1.5, we consider poly(N)-time constant-depth *uniform* circuits of $2^{\text{poly}(N)}$ size for the MISSING-STRING problem, and relate their existence directly to functions in $\Sigma_k\mathsf{E}$ that have high circuit complexity. Here we give a few more details about the kind of uniformity we require. These circuit families are "direct connect" uniform (in the sense of Ruzzo [20, 14]), in that local gate information about the n-th circuit C_n of size $2^{\text{poly}(N)}$ can be computed in poly(N) time. In particular, there is a language in P of strings of the form $\langle N, g, y\rangle$ where g and y are bit strings labelling gates in C_n (taking poly(N) bits to describe) and the output of y is an input to g in C_n, along with strings of the form $\langle N, g, t\rangle$ where t is the type of gate g (taking poly(N) bits to describe). This set of triples is often called the *connection language* of the underlying circuit. We stipulate that for circuits with 2^n outputs, each output gate has a label of the form $x01^t$ where $x \in \{0,1\}^n$ is the label of the x-th output. This makes it convenient to construct the label of any particular output gate.

Characterizations of NP and $\Sigma_k\mathsf{P}$ in general in terms of exponential-size "direct-connect" uniform circuits of constant depth are well-known [24, 25]. We will use relativized versions of these results. In our results regarding $\Sigma_k\mathsf{E}$ and circuit lower bounds, the inputs to our uniform circuits for MISSING-STRING will already be of length $N = 2^n$ (where n is the input length to the $\Sigma_k\mathsf{E}$ machine). In particular, we only need the following relativized equivalence between uniform circuits and the exponential hierarchy, which follows readily from the literature on circuits representing PH [10, 3].

Theorem 2.1. *For all oracles A, the following classes are equivalent:*

- $\Sigma_k\mathsf{E}^A = \Sigma_k\mathsf{TIME}^A[2^{O(n)}]$.
- *The class of decision problems computed by poly(2^n)-time uniform depth-($k+1$) circuits with output gate OR, size $2^{\text{poly}(2^n)}$, and bottom fan-in poly(2^n). The n-th circuit takes as input an n-bit string x as well as all values of the oracle A up to length poly(2^n).*
- *The class of decision problems computed by poly(2^n)-time uniform depth-($k+1$) circuits with output gate OR, size $2^{\text{poly}(2^n)}$, bottom fan-in poly(2^n), and 2^n output gates. The n-th circuit takes as input all values of the oracle A up to length poly(2^n), and outputs the truth table of the function on all n-bit inputs.*

Relativized Circuit Complexity. We recall notions of relativized circuit complexity, as originally defined by Wilson [27]. In a nutshell, relativized circuit complexity studies Boolean logic circuits with the usual AND, OR, NOT gates, along with extra oracle gates. In particular, for $A : \{0,1\}^\star \to \{0,1\}$, an A-oracle circuit C is a Boolean circuit over the basis of ORs and ANDs of fan-in two, NOTs, and gates computing $A_k : \{0,1\}^k \to \{0,1\}$ where A_k is the k-th slice of A. The *size* of such a circuit C is defined to be the sum of all fan-ins over all gates (in other words, we are counting the number of wires). Note that each copy of A_k contributes k to the circuit size. We define $\mathsf{SIZE}^A[(S(n))]$ to be the class of decision problems $f : \{0,1\}^\star \to \{0,1\}$ such that for every n, there is an A-oracle circuit C_n of $O(S(n))$ size that computes the n-th slice of f.

In the following, we consider a binary encoding of A-oracle circuits which is paddable: if a circuit C has an encoding of length ℓ, then for all $\ell' > \ell$, C has an encoding of length ℓ'. In our encoding of an A-oracle circuit, we do not give a full description of A_k if it appears in the circuit; rather, we assume that the oracle A is known implicitly to the encoder/decoder. We can define a relativized version of the P-complete Circuit Evaluation problem:

Definition 2.2. *Let $A : \{0,1\}^\star \to \{0,1\}$ be an oracle. In the CIRC-EVAL^A problem, we are given an input $1^n 0 z x$, and the task is to return the evaluation of $x \in \{0,1\}^n$ on the A-oracle circuit encoded by z.*

Observe that by standard arguments, CIRC-EVAL$^A \in \mathsf{TIME}^A[\tilde{O}(n)]$, i.e., A-oracle circuits can be evaluated in quasi-linear time with an A-oracle.

2.1 A Normal Form for Relativized Circuits

The results of this subsection may be known, but we have not found a reference. We show that relativized circuit complexity can be radically simplified for the problems we wish to study. For the oracles A in the literature placing a complexity class $\mathcal{C}^A$ in $\mathsf{SIZE}^A[s(n)]$ for small $s(n)$ (e.g., [27, 8]), the actual circuits used in such simulations is always extremely simple: they consist of **one** A-oracle gate with some of its $O(s(n))$ inputs hard-coded with zeroes and ones.

It turns out that considering such simple circuits is *without loss of generality*. We will show that for the purposes of simulating complexity classes with relativized circuits, we may freely assume that the A-oracle circuits we consider consist of **one A-oracle gate**. In the following, let $A : \{0,1\}^\star \to \{0,1\}$ be an oracle.

Definition 2.3. *For nice $s(n) > n$, $A[s(n)]$ is the class of decision problems $f : \{0,1\}^\star \to \{0,1\}$ such that there is an infinite sequence of strings $\{z_n\}$ such that for all n, $|z_n| \leq s(n)$, and for all $x \in \{0,1\}^n$, $f(x) = 1 \iff A(z_n x) = 1$.*

In other words, $A[s(n)]$ is the class of problems solvable by circuits consisting of **exactly one** gate, an A-oracle gate, along with an "advice" string z_n plugged into that oracle gate. Observe that from our definitions we have $A[s(n)] \subseteq \mathsf{SIZE}^A[O(s(n))]$, as the "fan-in" of the single A gate is $n + s(n)$.

The following basic lemma shows that for most interesting complexity classes $\mathcal{C}$, if there is an oracle A relative to which $\mathcal{C}$ has $O(s(n))$-size A-oracle circuits, then there is another oracle B relative to which $\mathcal{C}$ has circuits comprised of exactly one B-oracle gate of $O(s(n)\log s(n))$ fan-in. This "normal form" greatly simplifies the problem of finding an oracle relative to which a complexity class $\mathcal{C}$ has small circuits, and will be very useful in our results relating circuits for Missing-String to circuit complexity lower bounds (Theorems 1.6 and 1.5).

Lemma 2.4. *Let $\mathcal{C}$ be a complexity class (defined by machines) such that for all A, $\mathcal{C}^{\mathsf{TIME}^A[\tilde{O}(n)]} \subset \mathcal{C}^A$.*

- *There is an oracle A such that $\mathcal{C}^A \subset \mathsf{SIZE}^A[s(n) \cdot poly(\log s(n))]$ if and only if there is an oracle B such that $\mathcal{C}^B \subset B[s(n) \cdot poly(\log s(n))]$.*
- *Similarly, there is an oracle A such that $\mathcal{C}^A \subset$ i.o.-$\mathsf{SIZE}^A[s(n) \cdot poly(\log s(n))]$ if and only if there is an oracle B such that $\mathcal{C}^B \subset$ i.o.-$B[s(n) \cdot poly(\log s(n))]$.*

Proof. For both statements, one direction of the equivalence is trivial. In the following, we prove that an oracle A satisfying $\mathcal{C}^A \subseteq \mathsf{SIZE}^A[O(s(n))]$ implies an oracle B satisfying $\mathcal{C}^B \subseteq B[O(s(n)\log s(n))]$.

The idea is to simply define B to be CIRC-EVAL^A. Let M be an arbitrary machine computing a function in $\mathcal{C}$. Since CIRC-EVAL$^A \in \mathsf{TIME}^A[\tilde{O}(n)]$, using the assumption that $\mathcal{C}^{\mathsf{TIME}^A[\tilde{O}(n)]} \subseteq \mathcal{C}^A$

we may infer that $M^B = M^{\text{CIRC-EVAL}^A}$ computes some function $f' \in \mathcal{C}^A$. Let N^A be a $\mathcal{C}^A$-machine computing f', so N^A is equivalent to M^B. Applying the assumption $\mathcal{C}^A \subset \mathsf{SIZE}^A[s(n)]$, there is an A-oracle circuit family $\{D_n^A\}$ of size $O(s(n))$ simulating N^A. Let z_n be a description of D_n^A, such that $|z_n| \leq O(s(n) \log s(n))$. We have $D_n^A(x) = M^B(x)$ on all n-bit inputs x, and $D_n^A(x) = B(1^n 0 z_n x)$, since $B = \text{CIRC-EVAL}^A$. Therefore the function computed by M^B is in $B[O(s(n) \log s(n))]$. As this containment holds for every M, we conclude that $\mathcal{C}^B \subseteq B[O(s(n))]$. An analogous argument proves the same result for infinitely-often inclusions. □

2.2 Efficient Methods for Finding a Missing String

Multiple linear-time algorithms are known for Missing-String; see Chapter 1 of [6] for one of them. In fact, Missing-String can be solved very efficiently using a "voting" strategy. The strategy is well-known in computational learning theory; the first reference we are aware of that uses the strategy to find a missing string is Lemma 3 of [13] (a paper on circuit lower bounds), although as a method for mistake-bounded learning (a.k.a. the "Halving Algorithm") it was apparently first observed by Barzdins and Freivalds [5].

Theorem 2.5. *The* Missing String *problem on M strings (each of length N) can be solved by an algorithm using $O(M+N)$ time, or an algorithm using simultaneously $O(M \log M + N)$ time and $O(\log(M) + \log(N))$ space, assuming constant-time random access to the input. In particular, to compute any bit of the missing string, both algorithms only have to probe at most $2M$ distinct bits of the input.*

Proof. We generalize the proof of Lemma 3 in [13]. Compute the bit b_1 that occurs *least* among the first bits of all strings, taking $b_1 = 0$ if both bits occur equally often. Thus the number of strings with first bit equal to b_1 is at most $M/2$. Among those strings with their first bit equal to b_1, compute the bit b_2 that occurs least in the second bit of those strings. Repeating this process for at most $t \leq \lceil \log_2(M) \rceil$ times on the i-th bit, for $i = 1, ..., t$, we obtain a prefix $b_1 \cdots b_t$ that is not a prefix of any string in the list. Filling the rest of the string with zeroes (which takes at most $O(N)$ time), we obtain a missing string.

Let us first analyze the number of probes needed. To compute b_1, we have to make M probes. For $i > 1$, to compute b_i given the bits $b_1, \ldots, b_{i-1}$, the aforementioned procedure only has to make at most $M/2^{i-1}$ probes into the list. This is because in the previous iteration, we probed the $(i-1)$th bit of $t_{i-1} \leq M/2^{i-2}$ strings, and to compute b_i we only have to probe those i-th bits for which the $(i-1)$th bit of those t_{i-1} strings equals the least occurring bit b_{i-1}. Hence the total number of probes to compute b_t (and any other bits) is at most

$$\sum_{i=0}^{t-1} M/2^i \leq 2M.$$

We can use an $O(\log M)$-bit counter to determine each bit b_i. Recalling that incrementing a counter takes only constant amortized time, the above procedure can be implemented in $O(M)$ time by maintaining a data structure that is initially filled with all M strings, such that after the i-th iteration we can remove the strings whose i-bit prefix disagrees with $b_1 \cdots b_i$ in $O(|S_i|)$ time, where $|S_i|$ is the cardinality of the list in the i-th iteration. (A modified linked list suffices.)

To get a space-efficient algorithm, observe that we only need $O(\log M)$ extra space to store a counter and to store the bits $b_1, \ldots, b_i$. However, in the i-th iteration, to compute b_i given

$b_1, \ldots, b_{i-1}$, we will apparently need to re-compute all $\sum_{j=0}^{i-1} M/2^j = 2M(1-1/2^i)$ bits probed so far, in order to count the "minority bit" over those strings which have not yet been eliminated; this makes the running time $O(M \log M)$. Note that any extra zeroes printed at the end of the missing string (to make it have length N) do not count as part of the space usage; they are part of the output, and they can be handled with an $O(\log N)$-bit counter that counts up to $N - t$. □

Theorem 2.5 will be useful in our Missing-String algorithms which use a smaller number of probes.

3 Local Algorithms for Missing-String and New Non-Uniform Hierarchies

In this section, we consider algorithms computing Missing-String in a local fashion, probing very few bits to determine any particular output bit of the missing string. As a consequence, we will conclude a new non-uniform time hierarchy theorem.

Recall that in Missing-String, we are given a list L of $M < 2^n$ strings each of N bits, and wish to output an N-bit string (a *missing string*) that is not in L. We say an algorithm A for Missing-String is *b-probe* if for all $i = 1, \ldots, N$, A can determine the i-th bit of its missing string by only probing b bits in the input list L.

In order to have a small value for b, it is required that the list length M is fairly small. As an example, consider the special case where $M \leq N$: the number of strings in L is at most the string length. The classical diagonal argument immediately gives a 1-probe algorithm for Missing-String: the i-th bit of our missing string will flip the i-th bit of the i-th string in L. For integer $k \geq 1$, when $M \geq k \cdot N$, it is easy to prove that there is no k-probe Missing-String algorithm at all (we give a proof in Appendix A for completeness). We provide an algorithm that nearly matches this simple lower bound:

Theorem 3.1 (Local Algorithm for Missing-String)**.** *For all positive t, set $k := \lfloor (2^t - 1)/(2t) \rfloor$. For all positive integers N and $M \geq 2^t$ such that $M \leq kN$, there is an algorithm for* Missing-String *on all M-size lists of N-bit strings which uses only $O((\log N)^2 + k \log^2 k)$ time and $2(2^t - 1) \leq O(k \log k)$ probes to compute any desired output bit.*

Proof. The idea is to reduce the case of $M \leq kN$ to the case where we have $2^t - 1$ strings of length t for small t, by breaking the N-bit strings into substrings.

Let L be a list of $M \leq kN$ strings of length N, and let $i \in \{1, \ldots, n\}$ be the index of the desired output bit. For simplicity let us assume M is a multiple of $2^t - 1$ (but the result does not require it). Our algorithm A conceptually partitions the list of M strings into $\ell := M/(2^t - 1)$ sublists $L_1, \ldots, L_\ell$ of at most $2^t - 1$ strings each.[8] (We say "conceptually", because the algorithm does not actually have to partition the list in order to determine the i-th output bit.) We say that the sublist L_j is *responsible* for the i-th output bit if and only if

$$1 + t \cdot (j-1) \leq i \leq t \cdot j.$$

[8] If M is not a multiple of $2^t - 1$, then we need to set $\ell := \lceil M/(2^t - 1) \rceil$. To accommodate that increase in the calculations, we may set $k := \lfloor (2^t - 1)/(2t) \rfloor$.

For example, the sublist L_1 is responsible for output bits $1, \ldots, t$, sublist L_2 is responsible for output bits $t+1, \ldots, 2t$, and so on. From our assumption that $M \le kN$, it follows that $t \cdot M/(2^t - 1) = t \cdot \ell \le N$, so it may be that not all output bits are "covered" by some sublist.

Given i, our MISSING-STRING algorithm A first determines the $j \in \{1, \ldots, \ell\}$ such that L_j is responsible for the i-th output bit, by simply dividing i by t, taking at most $O((\log N)^2)$ time. (If there is no such sublist, then A outputs 0 for the i-th output bit.) Next, the algorithm A considers an instance of MISSING STRING on a set of substrings, defined as follows: over all $q \le 2^t - 1$ strings in L_j, strings $y_1, \ldots, y_q \in \{0,1\}^t$ are defined by taking each string $x \in L_j$ and truncating x to the t-bit substring $x[1 + t \cdot (j-1)] \cdots x[t \cdot j]$. Then, A uses the algorithm of Theorem 2.5 to determine a missing string $z \in \{0,1\}^t$ that is not among $y_1, \ldots, y_q \in \{0,1\}^t$, by probing at most $2q \le 2(2^t - 1) \le O(k \log k)$ bits and (conservatively) using $O(t2^t) \le O(k \log^2 k)$ time.[9] Finally, A outputs the bit $z_{i^\star}$, where $i^\star = i - t(j-1)$.

We claim that the string output by A (over all $i = 1, \ldots, N$) avoids every string in L. For each $j = 1, \ldots, \ell$, over those t bit positions i that L_j is responsible for, we are constructing a substring z of length t which avoids all strings in L_j, in the corresponding t bit positions of the strings in L_j. Since $N \ge t \cdot \ell$, there are enough bit positions so that for all ℓ sublists L_j, we can construct a substring z of length t avoiding the strings in L_j, each time using a disjoint set of t bit positions from all other sublists. The string output by A is a concatenation of all such z, avoiding all sublists L_j and thus avoiding the entire list L. □

What is the minimum number of probes needed to compute bits of a MISSING-STRING, when $M \le kN$? We leave it as an interesting open problem to close the gap between the simple lower bound of k probes (Theorem A.1) and our upper bound of $O(k \log k)$ probes (Theorem 3.1).

Remark 1. *Instead of measuring the worst-case number of probes needed to compute one bit of the missing string, suppose we consider the problem of simply constructing a missing string with the minimum number of* **total** *bit probes. In this case, the algorithm of Theorem 2.5 is essentially optimal: it constructs a missing string over $M \le 2^N - 1$ strings of length N with at most $2M$* **total** *probes into the input, and it is easy to see that at least M total probes are required (each string must be probed in at least one bit position). Therefore, with respect to this measure of probe complexity, we know the optimal number of probes within a factor of 2.*

Remark 2. *For at least one interesting case, we can determine the exact number of probes required. Suppose we are given* **exactly** $2^t - 1$ *strings of length t, so that the missing string is unique. In this case, the i-th bit of the missing string can be determined by taking the XOR of the i-th bits of all strings in the list. (The missing string equals the bit-wise XOR of all $2^t - 1$ other strings.) Thus we can use only $2^t - 1$ probes for each output bit. In this scenario, there is an algorithm making $M = 2^t - 1$ probes per output bit, and the number of probes cannot be made smaller than M.*

3.1 Non-Uniform Time Hierarchies

While Theorem 3.1 is interesting in its own right, we are interested in complexity-theoretic applications. Let us first prove Theorem 1.1, which is the most general form we will consider.

[9] We add a log-factor to the running time here, just to ensure that the time bound of Theorem 2.5 also holds for models such as multitape Turing machines.

Reminder of Theorem 1.1. *For all (time-constructible) $\alpha(n) \geq 0$, unbounded $\beta(n)$, $t(n) \geq n^2$, and every oracle A,*

$$\mathsf{TIME}^A[t(n)\log t(n) \cdot (\alpha(n)+\beta(n)) \cdot 2^{\alpha(n)+\beta(n)}] \not\subset i.o.\text{-}\mathsf{TIME}^A[t(n)]/(n+\alpha(n)).$$

That is, there are functions in time $O(t(n)\log t(n) \cdot (\alpha(n)+\beta(n)) \cdot 2^{\alpha(n)+\beta(n)})$ that cannot be simulated in time $t(n)$ even with $n+\alpha(n)$ non-uniform advice, for all but finitely many input lengths n.

Proof. Let A be an arbitrary oracle. We will construct a function f that is computable in time $O(t(n)\log t(n) \cdot (\alpha(n)+\beta(n)) \cdot 2^{\alpha(n)+\beta(n)})$ with an A-oracle, which cannot be computed in $O(t(n))$ time with an A-oracle, even with $n+\alpha(n)$ bits of non-uniform advice, on all but finitely many input lengths n.

To compute f, we will (implicitly) define a very large instance of Missing-String, such that the list L contains the 2^n-bit truth tables of all machines we would like to diagonalize against, and we will define the function f_n (f restricted to n-bit inputs) to have our missing string as its truth table. Each bit of the truth table of f_n will be computable in the desired running time, by applying Theorem 3.1 appropriately.

Set $m := n + \alpha(n) + \beta(n)$. Associate all m-bit strings with pairs of the form (y, i), where y is a possible advice string of length $n + \alpha(n)$, and i is a $beta(n)$-bit string encoding a possible Turing machine/algorithm M_i running in $t(n)$ time with an A-oracle. (Since $\beta(n)$ is unbounded, eventually every machine is encoded.) Make a list L of $M = 2^m$ strings of length $N = 2^n$, where each string encodes the truth table obtained by running some M_i with some advice y on all n-bit inputs. Provided we can solve Missing-String for the list L, on every possible n, this will produce the truth table of a function that cannot be computed on *all but finitely many input lengths* n by all such algorithms with $n + \alpha(n)$ advice.

We can use our algorithm from Theorem 3.1 provided that $2^m \leq k2^n$, i.e., $k \geq 2^{\alpha(n)+\beta(n)}$. Setting k to be $2^{\alpha(n)+\beta(n)}$, we can make a number of probes which is at most

$$O(k \log k) \leq O((\alpha(n)+\beta(n)) \cdot 2^{\alpha(n)+\beta(n)}).$$

For our particular list L, every probe into L corresponds to simulating some algorithm M_i on some advice y of length $n + \alpha(n)$ and some input x' of length n, for $O(t(n)\log t(n))$ time. (We allow an extra $\log t(n)$ factor for a universal simulator.) Thus the running time required to compute all probes is

$$O(t(n)\log t(n) \cdot (\alpha(n)+\beta(n)) \cdot 2^{\alpha(n)+\beta(n)}).$$

Besides computing all the probes, the additional runtime cost of the local algorithm for Missing-String is negligible:

$$O((\log N)^2 + k\log^2 k) \leq O(n^2 + t(n)\log t(n) \cdot (\alpha(n)+\beta(n)) \cdot 2^{\alpha(n)+\beta(n)}).$$

This completes the proof. □

Theorem 1.1 has several interesting corollaries, illustrating tradeoffs between the running time and advice.

Corollary 3.2. *1. For all $c \geq 1$, $\mathsf{TIME}[2^n \cdot n^{c+3}] \not\subset \mathsf{TIME}[2^n]/(n + c\log n)$.*
2. For all $\varepsilon > 0$, $\mathsf{TIME}[n^3 \cdot 2^{2\varepsilon n}] \not\subset \mathsf{TIME}[2^{\varepsilon n}]/((1+\varepsilon)n)$.
3. For every c, $\mathsf{TIME}[n^{2c}\log^3 n] \not\subset \mathsf{TIME}[n^c]/(n + c\log n)$.

Proof. 1. Set $t(n) = 2^n$, $\beta(n) = \log n$, and $\alpha(n) = c \log n$ in Theorem 1.1.
2. Set $\alpha(n) = \varepsilon n$, $\beta(n) = \log n$, $t(n) = 2^{\varepsilon n}$.
3. Set $\alpha = c \log n$, $\beta(n) = \log \log n$, $t(n) = n^c$. □

In the above corollaries, note that in all cases the hard function actually does not have enough time to directly "try all possible advice strings" and run the algorithms it must diagonalize against. Furthermore, as mentioned earlier, the theorem is essentially tight for relativizing lower bounds, because there *is* an oracle B such that $\mathsf{TIME}^B(o(2^{cn})) \subset \mathsf{TIME}^B(O(n))/((c+1)n)$. (See Appendix A for a sketch of this result, in terms of Missing-String.)

4 Circuits for Missing-String And Circuit Lower Bounds

In this section, we prove relations and equivalences between the existence of uniform circuits for Missing-String and relativizing circuit lower bounds for functions in the exponential hierarchy. We will focus on the case of $\Sigma_2\mathsf{E}$ as it is the most relevant "frontier" class for exponential-size circuit lower bounds.

Definition 4.1. *We say that an oracle $A : \{0,1\}^\star \to \{0,1\}$ is* supported on m-bit strings *if $A(x) = 0$ for all x satisfying $|x| \neq m$. We will view an A supported on m-bit strings as a string of length 2^m.*

In the following, think of $f(n)$ as a very slow-growing function (e.g., $f(n) = \log(n)$).

Definition 4.2. *Fix an $n \in \mathbb{N}$. Let U be a $\Sigma_2\mathsf{E}$ Turing machine such that $|U| \leq f(n)$. Define the function $D_{U,n} : \{0,1\}^{2^{s(n)+n}} \to \{0,1\}^n$ such that for every oracle A supported on $(s(n)+n)$-bit strings,*

$$D_{U,n}(A) = \{U^A(x)\}_{x \in \{0,1\}^n}.$$

We define $\{D_{U,n}\}$ to be the corresponding family of functions.

Definition 4.3. *Let $\{G_n\}$ be a multi-output circuit family with $2^{s(n)+n}$ inputs and 2^n outputs. We say that $\{G_n\}$ is a* **good circuit family** *if it is a poly(2^n)-time uniform depth-3 family of the form*

$$OR_{2^{poly(2^n)}} \circ AND_{2^{poly(2^n)}} \circ OR_{poly(2^n)},$$

as defined in Section 2.

Lemma 4.4. *Let $\{G_n\}$ be a good circuit family. Then there exists a $\Sigma_2\mathsf{E}$ Turing machine U such that for all n and all inputs $A \in \{0,1\}^{2^{s(n)+n}}$, $G_n(A) = D_{U,n}(A)$. That is, $G_n(A)$ outputs the values of $U^A(x)$ over all x of length n.*

Proof. Follows from Theorem 2.1. □

Here we will prove Theorem 1.6 for the case of $k = 2$; the case of arbitrary k is straightforward.

Theorem 4.5 (Circuits for Missing-String Imply Circuit Lower Bounds, Theorem 1.6 for $\Sigma_2\mathsf{E}$)**.** *If for infinitely many n, Missing-String can be solved by a good circuit family, on all lists of length $M := 2^{\tilde{O}(s(\log N))}$ with strings of length $N = 2^n$, then $\Sigma_2\mathsf{E}^A \not\subset \mathsf{SIZE}^A(\tilde{O}(s(n)))$ for all oracles A.*

Similarly, if for all n, Missing-String can be solved by a good circuit family, on all lists of length $M := 2^{\tilde{O}(s(\log N))}$ with strings of length $N = 2^n$, then $\Sigma_2\mathsf{E}^A \not\subset$ i.o.-$\mathsf{SIZE}^A(\tilde{O}(s(n)))$ for all oracles A.

Proof. We will prove the first implication; the modification to the second implication is straightforward. By Lemma 4.4, the hypothesis implies that there exists a $\Sigma_2\mathsf{E}$ Turing machine U such that the good circuit family solving MISSING-STRING computes the same function as $\{D_{U,n}\}$. Let V be a $\Sigma_2\mathsf{E}$-machine simulating U, so that whenever U asks a query to the oracle of length $\log(M)+n$, V uses the actual oracle result, and for all queries of other lengths V pretends the query answer is 0. Let A be any oracle and let A_n be its restriction to $(\log(M)+n)$-bit strings; that is, $A_n(z) = A(z)$ if $|z| = (\log(M)+n)$, and $A_n(z) = 0$ otherwise. Observe that A_n restricted to strings of length $(\log(M)+n)$ can be viewed as an instance of the MISSING-STRING problem with a list of length M and each string of length $N = 2^n$. From these definitions, we infer that for all $x \in \{0,1\}^n$, $V^A(x) = U^{A_n}(x)$ and hence

$$\{V^A(x)\}_{x\in\{0,1\}^n} = \{U^{A_n}(x)\}_{x\in\{0,1\}^n} = D_{U,n}(A_n).$$

As $D_{U,n}$ can solve the MISSING-STRING problem with a list of length M and each string of length $N = 2^n$ for infinitely many n, we have that for infinitely many n, for all A, and for all $z \in \{0,1\}^{\log(M)}$,

$$\{V^A(x)\}_{x\in\{0,1\}^n} = D_{U,n}(A_n) \neq \{A_n(z,x)\}_{x\in\{0,1\}^n} = \{A(z,x)\}_{x\in\{0,1\}^n}.$$

In turn, this implies that for infinitely many n, for all A, and for $z \in \{0,1\}^{\log(M)}$,

$$\{V^A(x)\}_{x\in\{0,1\}^n} \neq \{A(z,x)\}_{x\in\{0,1\}^n}.$$

As this condition holds for infinitely many n, we conclude that for all A, V^A does not have A-oracle circuits consisting of one query of length $\log(M)$ to A, i.e., $V^A \notin A[\log(M)]$.[10] For every constant d, assigning $M := 2^{s(\log N)\log^d(s(\log N))}$ implies that for all A, $V^A \notin A[s(n)\log^d(s(n)]$. Hence, $V^A \notin A[\tilde{O}(s(n))]$. Since V^A is a $\Sigma_2\mathsf{E}^A$ machine, this implies that for all A, $\Sigma_2\mathsf{E}^A \not\subset A[\tilde{O}(s(n))]$. Finally, by Lemma 2.4, this implies that for all B, $\Sigma_2\mathsf{E}^B \not\subset \mathsf{SIZE}[\tilde{O}(s(n))]^B$. □

Now, we want to go in the opposite direction: we want to show that if MISSING-STRING *does not* have good uniform circuits, then $\Sigma_2\mathsf{E}^A \subset$ i.o.-$\mathsf{SIZE}^A[O(s(n))]$ for some oracle A. (Contrapositively, we will show that relativizing circuit lower bounds for $\Sigma_2\mathsf{E}$ imply good uniform circuits for MISSING-STRING.) This direction will take more work. To this end, we define a property P which will be sufficient to imply $\Sigma_2\mathsf{E}^A \subset$ i.o.-$\mathsf{SIZE}^A[O(s(n))]$. Property P asks for something stronger than a typical lower bound: a single input to the MISSING-STRING problem on which all uniform circuits (algorithms) fail. We will later see (Lemma 4.9) how MISSING-STRING lower bounds can be used to derive property P, by allowing a little loss in the parameters.

Definition 4.6. *Let $P(s, f, n)$ be the following property:*

There is an oracle Q supported on $(s+n)$-bit strings such that, for every $\Sigma_2\mathsf{TIME}[2^n]$ Turing machine of description length $|U| \leq f$, the function $D_{U,n}$ cannot solve the MISSING-STRING *problem on the input Q, with list of length $M = 2^s$ and strings of length $N = 2^n$.*

Roughly speaking, property P says "there is a bad input Q that breaks every machine U (of length at most f) trying to solve MISSING-STRING."

Lemma 4.7. *Let $f(n)$ be unbounded. If there are infinitely many n such that $P(s(n), f(n), n)$ holds, then there exists a constant μ and an oracle A such that $\Sigma_2\mathsf{TIME}[2^n]^A \subset$ i.o.-$\mathsf{SIZE}^A[\mu s(n)]$.*

[10] See Section 2 for a definition of the $A[s(n)]$ notation.

Proof. The oracle will be constructed in stages.

Stage 1: Start with the first n for which $P(s(n), f(n), n)$ holds. Then there exists an oracle B supported on $(s(n)+n)$ bit strings, so that for all $\Sigma_2\mathsf{TIME}[2^n]$ Turing machines U where $|U| \leq f(n)/2$, there exists a $z \in \{0,1\}^{s(n)}$ such that that $U^B(x) = B(z, x)$ for all x of length n. That is, U^B has a "trivial" B-oracle circuit of size $s(n)+n$ on inputs of length n. Set $A := B$. We will view A as a set defined by the inputs on which the oracle A is 1.

Stage i for $i > 1$: Let us now extend the previous statement for infinitely many n. Let m upper bound the length of the binary encoding of the set A. Let n_j refer to the integer n chosen in stage j, where $1 \leq j < i$.

In stage i, we choose n such that three properties hold.

1. $f(n) > 2 \cdot m$
2. $P(s(n), f(n), n)$ holds.
3. n is large enough to satisfy $2^{n_{i-1} \cdot f(n_{i-1})} < s(n) + n$.

We want to add a set B_n of strings of length $s(n)+n$ to our existing oracle A, yielding a new oracle $A := A \cup B_n$. We can argue by induction that all existing entries in the set A have length at most $2^{n_{i-1} f(n_{i-1})}$, hence by Property 3, $A \cap B_n = \emptyset$. Note that any machine of the form $U^{A \cup B_n}$ can be thought of as another machine $U_2^{B_n}$ where in U_2 we hard-code A, where $|U_2| \leq |U| + m$. By Property 1, if $|U| \leq f(n)/2$, then $|U_2| \leq f(n)$. As in stage 1, property $P(s(n), f(n), n)$ implies there exists an oracle B_n supported on $(s(n)+n)$ bit strings such that for all $\Sigma_2\mathsf{TIME}[2^n]$ machines U_2 satisfying $|U_2| \leq f(n)$, there is a string z of length $s(n)$ such that $U_2^{B_n}(x) = B_n(z, x)$ for all $x \in \{0,1\}^n$. That is, $U_2^{B_n}$ has a "trivial" B_n-oracle circuit on inputs of length n.

Therefore, for all $\Sigma_2\mathsf{TIME}[2^n]$ machines U satisfying $|U| \leq f(n)/2$, there is a string z' of length $s(n)$ such that for all n-bit x, $U^{A \cup B_n}(x) = U_2^{B_n}(x) = B_n(z', x)$. Note that since z' is of length $s(n)$, and A only contains strings of length less than $s(n)+n$, we have $B_n(z', x) = (A \cup B_n)(z', x)$. Therefore $U^{A \cup B_n}(x) = (A \cup B_n)(z', x)$ for all x. That is, $U^{A \cup B_n}$ also has a "trivial" $(A \cup B_n)$-oracle circuit.

Now, we set A to be $A \cup B_n$. By Property 3, we chose n to be large enough that all previous machines considered do not query strings of length $s(n)+n$. Hence for this new setting of A, we still have that for all $\Sigma_2\mathsf{TIME}[2^n]$ Turing machines U with $|U| \leq f(n_j)/2$, and for all $j < i$, there is an z_j of length $s(n_j)$ such that for all x of length n_j, $U^A(x) = A(z_j, x)$.

Let A denote the final oracle after all stages. For infinitely many n, for all $\Sigma_2\mathsf{TIME}[2^n]$ machines U with $|U| \leq f(n)/2$, we have a string z_n of length $s(n)$ such that for all x, $U^A(x) = A(z_n, x)$. As $f(n)$ is unbounded, this is sufficient to imply that $\Sigma_2\mathsf{TIME}[2^n]^A \subseteq$ i.o.-$A[s(n)]$.[11] Taking μ to be a constant satisfying $A[s(n)] \subseteq \mathsf{SIZE}^A[\mu s(n)]$, we conclude that $\Sigma_2\mathsf{TIME}[2^n]^A \subset$ i.o.-$\mathsf{SIZE}^A[\mu s(n)]$. ☐

We next define a property R, which intuitively corresponds to the existence of lower bounds for the Missing-String problem.

Definition 4.8. *Let $R(s, f, n)$ denote the following property:*

For all functions $D_{U,n}$ (where U is a $\Sigma_2\mathsf{TIME}[2^n]$ Turing machine, and the description length $|U| \leq f$), there exists an oracle Q supported on $(s+n)$ bit strings such that $D_{U,n}$ cannot solve the Missing-String *problem on lists of length $M = 2^s$ and strings of length $N = 2^n$ on input Q.*

[11] See Section 2 for a definition of the $A[s(n)]$ notation.

Roughly speaking, property R says "for all machines U, there is a bad input Q on which U does not solve MISSING-STRING", while property P said "there is a single bad input Q such that no machine U solves MISSING-STRING on Q." In Lemma 4.9 we will show how property R actually implies the stronger property P, with a little loss in the parameters.

Let $f(n) < s(n), n < s(n)$ and let $f(n)$ be poly(n)-time constructible (eventually we will choose $f(n) = \log n$). Define $n' := n + f(n)$ and define s' such that $s'(n') := s(n) - f(n)$. Note that $s(n) + n = s'(n') + n'$. We now show that property R implies property P, with a mild loss in the parameters.

Lemma 4.9. *There exists a constant c such that $R(s'(n), c, n') \implies P(s(n), f(n), n)$. Therefore, if there are infinitely many m such that $R(s'(m), c, m)$ holds, then there are infinitely many m such that $P(s(m), f(m), m)$ holds.*

Proof. We will prove the contrapositive. If $P(s(n), f(n), n)$ is false, then for every possible input Q of length $2^{n+f(n)}$ to the MISSING-STRING problem with $M = 2^{s(n)}$ and $N = 2^n$, at least one function of the form $D_{U,n}$ (where U is a $\Sigma_2\mathsf{TIME}[2^n]$ machine, $|U| \leq f$) correctly solves MISSING-STRING on Q.

We will now show that there is a *single* function which solves the MISSING-STRING problem over all lists of length $M = 2^{s(n)-f(n)}$ and strings of length $N = 2^{n+f(n)}$. Later we will argue that this function can be computed in $\Sigma_2\mathsf{TIME}[2^n]$.

Note that $M \cdot N = 2^{n+s(n)}$. We may think of each string T of length $2^{n+f(n)}$ in the list as being divided into $2^{f(n)}$ sub-strings, each of length 2^n. Creating a list of all these sub-strings gives us an instance I of the MISSING-STRING problem with list of length $2^{s(n)-f(n)} \cdot 2^{f(n)} = 2^{s(n)}$ and strings of length 2^n. Similarly we will think of the output of length $2^{n+f(n)}$ as being divided into $2^{f(n)}$ sub-outputs, each of length 2^n. Let U_z be the $\Sigma_2\mathsf{TIME}[2^n]$ machine with description z, where $|z| \leq f(n)$. To obtain the z^{th} sub-output, we evaluate the function $D_{U_z,n}$ on I, obtaining a string of length 2^n. Note that all the sub-functions are being run on the same instance of MISSING-STRING with list of length $M = 2^{s(n)}$ and strings of length $N = 2^n$, generated by treating the sub-strings as strings. Since $P(s(n), f(n), n)$ is false, some machine U_w solves MISSING-STRING correctly: there exists a string w of length at most $f(n)$ such that $D_{U_w,n}$ is correct on the instance I. This means that the w^{th} sub-output is distinct from all sub-strings in the input. Hence the entire output string of length $2^{n+f(n)}$ must be distinct from all strings in the input. Therefore this function correctly solves MISSING-STRING on lists of length $M = 2^{s(n)-f(n)}$ and strings of length $N = 2^{n+f(n)}$.

We observe that this function is of the form $D_{U',n'}$ where U' is a $\Sigma_2\mathsf{TIME}[2^n]$ machine and $|U'| = O(1)$. In fact, U' is the machine which on input zx (where $|z| = f(n)$, $|x| = n$, $|zx| = n + f(n) = n'$) simply returns the output of the $\Sigma_2\mathsf{TIME}[2^n]$ machine with description z on input x. It is now easy to verify that the constructed function is equivalent to $D_{U',n'}$. Note that U' runs in time $O(2^n) \leq O(2^{n'})$. As $f(n)$ is polynomial-time constructible, we get that $|U'| \leq O(1)$. Taking $c = |U'|$, we see that $R(s'(n), c, n')$ is false. □

Corollary 4.10. *Let $f(n) = \log(n)$ and let s' be a nice function. There exists constants c, d such that if there are infinitely many n' such that $R(s'(n), c, n')$ holds, then there are infinitely many n such that $P(s'(n)\log^d(s'(n)), f(n), n)$ holds.*

Proof. Choose c to be the constant from Lemma 4.9. Choose d to be a constant such that $s'(n + \log(n)) + \log(2n) < s'(n)\log^d(s'(n))$, existence of d is guaranteed because s' is a nice function (see Section 2 for a definition). Let $R(s'(n), c, n')$ be true. Define $n, s(n)$ to be the solutions of

$n + f(n) = n'$ and $s(n) = f(n) + s'(n')$. By Lemma 4.9 we have that $P(s(n), f(n), n)$ is true. Note that $s(n) = f(n)+s'(n') \leq f(n')+s'(n') = \log(n+\log(n))+s'(n+\log(n)) \leq \log(2n)+s'(n+\log(n)) \leq s'(n)\log^d(s'(n))$. Hence $P(s'(n)\log^d(s'(n)), f(n), n)$ holds. □

Theorem 4.11 (Equivalence of Missing-String Lower Bounds and Circuit Upper Bounds, Theorem 1.5 for $\Sigma_2\mathsf{E}$)**.** *For all nice functions $s(n)$, the following are equivalent:*

1. *For every oracle A, $\Sigma_2\mathsf{E}^A \not\subset$ i.o.-$\mathsf{SIZE}^A(\tilde{O}(s(O(n))))$.*
2. Missing-String *can be solved by a good circuit family on all lists of length M and strings of length $N = 2^n$ where M satisfies $M = 2^{\tilde{O}(s(O(\log N)))}$*

Proof. By Theorem 4.5 we have that $2 \implies 1$.

Let us now prove $\neg 2 \implies \neg 1$. By Lemma 4.4, for a good circuit family $\{G_n\}$ there exists a $\Sigma_2\mathsf{E}$ Turing machine U such that the output of G_n is the same as that of $\{D_{U,n}\}$. Hence by $\neg 2$ we have that for all circuit families $\{D_{U,n}\}$ such that U is a $\Sigma_2\mathsf{E}$ machine, there are infinitely many n such that $D_{U,n}$ cannot solve the Missing-String problem for some $M = 2^{\tilde{O}(s(O(n)))}$ and $N = 2^n$. This implies that for all constants c, there are infinitely many n such that property $R(\tilde{O}(s(O(n))), c, n)$ holds. Taking $f(n) = \log(n)$ and applying Corollary 4.10, it follows that there are infinitely many n such that $P(\tilde{O}(s(O(n))), f(n), n)$ holds. Applying Lemma 4.7, there exists an oracle B such that $\Sigma_2\mathsf{TIME}[2^n]^B \in$ i.o.-$\mathsf{SIZE}^B[\tilde{O}(s(O(n)))]^B$. Finally, by padding we conclude that $\Sigma_2\mathsf{E}^B \in$ i.o.-$\mathsf{SIZE}^B[\tilde{O}(s(O(n)))]$, which is equivalent to $\neg 1$. □

4.1 A Perspective on Non-Relativizing Lower Bounds

In this section, we observe that non-relativizing lower bounds are equivalent to uniform circuits for Missing-String on low-complexity inputs.

Reminder of Theorem 1.8. *Let $S(n) \geq n$. $\mathsf{NE} \not\subset \mathsf{SIZE}(S(n))$ if and only if there is a poly(N)-time uniform depth-two circuit family of size $2^{poly(N)}$ and poly(N) bottom fan-in for* Missing-String *that succeeds for infinitely many $N = 2^n$, when the list L is $S(n)$-compressible.*

The proof follows easily from the framework given earlier in this section.

Proof. (Sketch) First, assume $\mathsf{NE} \not\subset \mathsf{SIZE}(S(n))$, and let M be an NE machine without $S(n)$-size circuits for infinitely many n. Applying the standard translation between NP and DC-uniform circuits (see 2) we can define a poly(2^n)-uniform depth-two circuit family $\{C_n\}$ which takes no inputs, and whose 2^n output gates print the truth table of M on n-bit inputs. For infinitely many n, this family $\{C_n\}$ is printing a string which is not $S(n)$-compressible; therefore it cannot be on any $S(n)$-compressible list.

For the other direction, suppose there is a poly(N)-time uniform depth-two circuit family $\{C_n\}$ of size $2^{\text{poly}(N)}$ for Missing-String that succeeds for infinitely many $N = 2^n$, when the list L is $S(n)$-compressible. So there is an infinite set $\{n_i\}$ where for all i, the circuit C_{n_i}, given the list L_{n_i} of all possible 2^{n_i}-bit truth tables of circuit complexity at most $S(n_i)$, outputs an 2^{n_i}-bit truth table T which does not appear in L_{n_i}. This T therefore has circuit complexity greater than $S(n)$. Note that either there are infinitely many i such that the depth-two circuit C_{n_i} is an OR of AND (of poly(N) fan-in), or there are infinitely many i such that it is an AND of OR (of poly(N) fan-in). In the first case, we have an NE machine computing a function not in $\mathsf{SIZE}(S(n))$; in the second case, we have a coNE machine computing a function not in $\mathsf{SIZE}(S(n))$ (which implies an NE machine as well; the class $\mathsf{SIZE}(S(n))$ is closed under complement). In more detail, in the first case, our NE machine on an input x (of length n) will first check the gate type of the output gate of C_n. If the type is OR, then the NE machine evaluates C_n on the list L_n, obtaining a truth table T, and the machine outputs the appropriate bit of T, corresponding to x. (If the type is AND, the NE machine just reports 0.) $\square$

5 Oracles Breaking The Algorithmic Method for Circuit Lower Bounds

Reminder of Theorem 1.9. *There is an oracle A and a function H such that $\mathsf{TIME}^A[2^n] \subset \mathsf{SIZE}^A[O(n)]$, SAT^A can be solved in poly$(H($poly$(n)))$ time with an A-oracle, $H(H(n)) \leq 2^n$ and H is monotone increasing.*

In the following, we let $\mathbb{h}$ denote a half-exponential function, i.e., a strictly monotone increasing $\mathbb{h} : \mathbb{N} \to \mathbb{N}$ where $\mathbb{h}(\mathbb{h}(n)) = 2^n$.

Proof. Let $N^A(x)$ be a nondeterministic machine accepting a linear-time complete language for the class $\mathsf{NTIME}^A[O(n)]$. In particular, let N^A run in time $\leq \mu n$. For a sufficiently large constant $c \geq 1$, setting $s(n) := \mathbb{h}(n)^c$, we will construct an oracle A such that for all n and for all $x \in \{0,1\}^n$, $N^A(x) = A(1^{s(n)}0x)$. Then the language $L(N^A)$ can be decided in time poly$(\mathbb{h}(n))$ with the oracle A, by simply constructing $s(n)$ ones, and querying A on $1^{s(n)}0x$. By padding, it follows that SAT will be computable in poly$(\mathbb{h}($poly$(n)))$ relative to the oracle A.

All queries to A will have one of three types: fixed, unfixed, and *derived*. Initially all queries of the form $1^{s(n)}0x$ where $x \in \{0,1\}^n$ are *derived*. Throughout the construction of the oracle, we will never set the value of A on derived queries; rather, we will assume that on derived queries the oracle satisfies $A(1^{s(n)}0x) := N^A(x)$ for all $x \in \{0,1\}^n$. This means that whenever the oracle A is changed, we assume that the value of A on derived queries is also changed to satisfy $A(1^{s(n)}0x) := N^A(x)$. This can always be done, since $1^{s(n)}0x$ is too long to be queried directly by $N^A(x)$. Over the course of the oracle construction, some unfixed and derived queries will become fixed queries. Whenever a derived query of the form $1^{s(n)}0x$ with $|x| = n$ into a fixed query, we will ensure that the output of $N^A(x)$ does not change after that. This will ensure that the condition $N^A(x) = A(1^{s(n)}0x)$ is maintained.

Let M^A be a E^A machine which accepts a linear-time complete set for E^A. We wish to construct A so that M^A has small A-oracle circuits. Let M^A run in time at most 2^{cn} for a large enough constant c. (Note that this fixes $s(n) = \mathbb{h}(n)^c$).

We will proceed in stages. In stage 0, we begin with $A(y) = 0$ for all y, and all queries other than derived queries to A are unfixed. We will inductively prove that after stage n:

1. At most $2^{c(n+1)}$ queries of A have been fixed.
2. All derived queries have length at least $s(n+1)$.

These properties are clearly true after stage 0.

Stage n: Our aim is to modify A such that there is a string z_n of length $O(n)$ such that $M^A(x) = A(z_n x)$ for all $x \in \{0,1\}^n$. As M^A is a E^A machine, we may view its computation on n-bit input as a decision tree of depth at most 2^{cn} where each node of the tree queries A on some input. We can simplify the tree by substituting the values of all fixed queries to A. We are left with two kinds of queries: derived and unfixed. Note that the oracle value of each derived query $A(1^{s(|y|)}0y)$ is the same as $N^A(y)$. Because N^A runs in nondeterministic μn time, each query $A(1^{s(|y|)}0y)$ can in turn be replaced with a DNF D_y of at most $2^{\mu|y|}$ terms, of width at most $\mu|y|$ (each term has at most $\mu|y|$ literals). By substituting the values for the fixed queries, D_y can be simplified so that each literal of D_y corresponds to the value of A on an unfixed or derived query. We next prove that, because we took $s(|y|)$ to be sufficiently large, the following stronger condition holds: every literal in every DNF D_y must correspond to an unfixed query, i.e., not a derived query.

Claim 5.1. *For every derived query $A(1^{s(|y|)}0y)$ by M^A on any input x of length n,*

- $|y| < \mathbb{h}(n)$.
- *Each literal of the DNF D_y corresponds to an unfixed (*not *derived) query to the oracle A.*

Proof. Let $1^{s(|y|)}y$ be a derived query of $M^A(x)$. As $M^A(x)$ runs in at most 2^{cn} time, we have that $s(|y|) < 2^{cn}$. Since $s(|y|) = \mathbb{h}(|y|)^c$, this implies that $\mathbb{h}(|y|) < 2^n$. Since $\mathbb{h}(\mathbb{h}(n)) = 2^n$ and $\mathbb{h}$ is monotone increasing, we must have $|y| < \mathbb{h}(n)$. Furthermore,

$$|y| < \mathbb{h}(n) = s(n)^{1/c} < s(n)/\mu,$$

where the last inequality holds for all large enough n. As $N^A(y)$ runs in time at most $\mu|y|$, it follows that all queries by $N^A(y)$ are of length at most $\mu|y| < s(n)$. The literals of the DNF D_y correspond to the oracle queries of $N^A(y)$, therefore hence all literals in D_y correspond to queries to A of length less than $s(n)$. As our induction hypothesis says that derived queries are of length greater than $s(n)$, each literal of D_y corresponds to an unfixed query. □

Given the claim, our E^A machine M^A can be viewed as a decision tree where each node of the tree is labelled by a DNF over the *unfixed* queries to A. We have now reached the same situation that arose in Theorem 1.10, which also arises in Wilson's [27] construction of an oracle B such that $\mathsf{E}^{\mathsf{NP}^B} \subset \mathsf{SIZE}[O(n)]^B$. We will use the same proof technique to embed M^A in $\mathsf{SIZE}^A[O(n)]$.

We will handle each $x \in \{0,1\}^n$ one by one. For each x, the output of $M^A(x)$ can be viewed as a decision tree of depth 2^{cn} where each node of the decision tree is a DNF with at most $2^{\mu\mathbb{h}(n)} < 2^{2^n}$ terms and each term has width at most $\mu\mathbb{h}(n) < 2^n$. Each literal in each DNF corresponds to an unfixed (but not derived) value of A. We start from the DNF corresponding to the top node of the decision tree. We just set an arbitrary term of this DNF to true by setting values of A on unfixed queries and fixing them. As each term has width at most 2^n, this fixes at most 2^n values of A. As long as we only change unfixed queries, the DNF will not change its output. Next, we move to the next active node in the decision tree and repeat the procedure, until we reach a leaf node of the tree. Since the tree has depth at most 2^{cn}, the number of queries to A that were fixed is at most $2^{cn} \cdot 2^n = 2^{(c+1)n}$. As long as we change only unfixed queries from now on, $M^A(x)$ will not change its output. Repeating the above for all n-bit x, in total we fix at most $2^{(c+1)n} \cdot 2^n = 2^{(c+2)n}$ queries of A. Finally, we convert all inputs of the form $1^{s(n)}0x$ from derived to fixed, proving the second inductive hypothesis for stage n. Combining this with the induction hypothesis from stage $n-1$, the total number of fixed inputs is $2^{cn} + 2^{(c+2)n} + 2^n < 2^{(c+4)n}$ for a large enough constant

c. Hence there exists a string z_n of length $O(n)$ such that no string of the form $z_n \ldots$ is fixed. We set $A[z_n x] \coloneqq M^A(x)$ for all $x \in \{0,1\}^n$, and declare $z_n x$ to be fixed. This is a valid assignment, as doing so does not change the values of $M^A(x)$ (we have only changed the value of A on unfixed inputs). This completes stage n.

Note that the total number of queries that are fixed is at most $2^{(c+4)n} + 2^n \leq 2^{c(n+1)}$ for a large enough constant c. This proves the first inductive hypothesis for stage n.

Finally, we note that M^A has low circuit complexity. For all x of length n, we have $M^A(x) = A(z_n x)$. Since $|z_n| \leq O(n)$, the query to A can be implemented as a linear sized circuit on n-bit inputs. Since M^A recognizes a linear time complete set for E^A, we also have $\mathsf{E}^{\mathsf{NP}^A} \subset \mathsf{SIZE}^A[O(n)]$. We also showed that for all $x \in \{0,1\}^n$, $N^A(x) = A(1^{s(n)}0x)$, hence the language $L(N^A)$ is in $\mathsf{TIME}[O(s(n)]$. It follows that SAT is computable in time $s(n^c) \leq \mathbb{h}(n^c)^c$ for a constant $c \geq 1$. (Indeed, all of NP is in $\mathsf{TIME}[\mathbb{h}(n^{O(1)})^{O(1)}]$.) $\square$

6 Acknowledgements

We thank Dylan McKay and Brynmor Chapman for helpful discussions on the missing string problem. We thank Jiatu Li for pointing out a bug (Theorem 1.10) in the previous version of this paper.

References

[1] Scott Aaronson. "Closed-form" functions with half-exponential growth. Math Overflow, https://web.archive.org/web/20210514235809/https://mathoverflow.net/questions/45477/closed-form-functions-with-half-exponential-growth, 2010.

[2] Scott Aaronson. G. phi. fo. fum. Shtetl-Optimized, URL: https://web.archive.org/web/20220221023022/https://scottaaronson.blog/?p=2521, 2015.

[3] Dana Angluin. On counting problems and the polynomial-time hierarchy. *Theor. Comput. Sci.*, 12:161–173, 1980. doi:10.1016/0304-3975(80)90027-4.

[4] Sanjeev Arora and Boaz Barak. *Computational Complexity - A Modern Approach*. 2009. URL: http://www.cambridge.org/catalogue/catalogue.asp?isbn=9780521424264.

[5] J. M. Barzdiņš and R. V. Freivalds. On the prediction of general recursive functions (in russian). *Doklady Akademii Nauk*, 206(3):521–524, 1972.

[6] Richard Bird. *Pearls of Functional Algorithm Design*. 2010. URL: http://www.cambridge.org/gb/knowledge/isbn/item5600469.

[7] Harry Buhrman, Lance Fortnow, and Thomas Thierauf. Nonrelativizing separations. In *Proceedings of the 13th Annual IEEE Conference on Computational Complexity, Buffalo, New York, USA, June 15-18, 1998*, pages 8–12, 1998. doi:10.1109/CCC.1998.694585.

[8] Lance Fortnow. Comparing notions of full derandomization. In *Proceedings of the 16th Annual IEEE Conference on Computational Complexity, Chicago, Illinois, USA, June 18-21, 2001*, pages 28–34, 2001. doi:10.1109/CCC.2001.933869.

[9] Lance Fortnow and Rahul Santhanam. Time hierarchies: A survey. *Electron. Colloquium Comput. Complex.*, (004), 2007.

[10] Merrick L. Furst, James B. Saxe, and Michael Sipser. Parity, circuits, and the polynomial-time hierarchy. *Mathematical Systems Theory*, 17(1):13–27, 1984.

[11] 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.

[12] Valentine Kabanets and Russell Impagliazzo. Derandomizing polynomial identity tests means proving circuit lower bounds. *Computational Complexity*, 13(1-2):1–46, 2004. doi:10.1007/s00037-004-0182-6.

[13] Ravi Kannan. Circuit-size lower bounds and non-reducibility to sparse sets. *Inf. Control.*, 55(1-3):40–56, 1982. Preliminary version in FOCS'81. doi:10.1016/S0019-9958(82)90382-5.

[14] Ravi Kannan, H. Venkateswaran, V. Vinay, and Andrew Chi-Chih Yao. A circuit-based proof of Toda's theorem. *Inf. Comput.*, 104(2):271–276, 1993. doi:10.1006/inco.1993.1033.

[15] Richard Karp and Richard Lipton. Turing machines that take advice. *L'Enseignement Mathématique*, 28(2):191–209, 1982.

[16] Robert Kleinberg, Oliver Korten, Daniel Mitropolsky, and Christos H. Papadimitriou. Total functions in the polynomial hierarchy. In *12th Innovations in Theoretical Computer Science Conference, ITCS*, volume 185 of *LIPIcs*, pages 44:1–44:18, 2021. doi:10.4230/LIPIcs.ITCS.2021.44.

[17] Oliver Korten. The hardest explicit construction. In *62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2021, Denver, CO, USA, February 7-10, 2022*, pages 433–444, 2021. doi:10.1109/FOCS52979.2021.00051.

[18] Peter Bro Miltersen, N. V. Vinodchandran, and Osamu Watanabe. Super-polynomial versus half-exponential circuit size in the exponential hierarchy. In *Computing and Combinatorics, 5th Annual International Conference, COCOON '99, Tokyo, Japan, July 26-28, 1999, Proceedings*, volume 1627 of *Lecture Notes in Computer Science*, pages 210–220, 1999. doi:10.1007/3-540-48686-0_21.

[19] Hanlin Ren, Rahul Santhanam, and Zhikun Wang. On the Range Avoidance problem for circuits. In *63rd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2022 (to appear)*, 2022.

[20] Walter L. Ruzzo. On uniform circuit complexity. *J. Comput. Syst. Sci.*, 22(3):365–383, 1981. doi:10.1016/0022-0000(81)90038-6.

[21] Larry J. Stockmeyer and Albert R. Meyer. Cosmological lower bound on the circuit complexity of a small problem in logic. *J. ACM*, 49(6):753–784, 2002.

[22] Larry Joseph Stockmeyer. *The complexity of decision problems in automata theory and logic*. PhD thesis, Massachusetts Institute of Technology, Department of Electrical Engineering, https://dspace.mit.edu/handle/1721.1/15540, 1974.

[23] Iannis Tourlakis. Time-space tradeoffs for SAT on nonuniform machines. *J. Comput. Syst. Sci.*, 63(2):268–287, 2001.

[24] H. Venkateswaran. Circuit definitions of nondeterministic complexity classes. *SIAM J. Comput.*, 21(4):655–670, 1992. doi:10.1137/0221040.

[25] V. Vinay, H. Venkateswaran, and C. E. Veni Madhavan. Circuits, pebbling and expressibility. In *Proceedings: Fifth Annual Structure in Complexity Theory Conference*, pages 223–230, 1990. doi:10.1109/SCT.1990.113970.

[26] Ryan Williams. Improving exhaustive search implies superpolynomial lower bounds. *SIAM Journal on Computing*, 42(3):1218–1244, 2013.

[27] Christopher B. Wilson. Relativized circuit complexity. *J. Comput. Syst. Sci.*, 31(2):169–181, 1985. doi:10.1016/0022-0000(85)90040-6.

A Simple Lower Bound for Local Algorithms Solving Missing-String

Recall M is the number of bit strings in our input list to Missing-String, and N is the length of the bit strings. Recall that an algorithm is *k-probe* if for all $i = 1, \ldots, N$, the i-th output bit of the missing string can be determined by only probing k bits of the input list.

Theorem A.1. *Every k-probe algorithm for* Missing-String *can only be correct when $M \leq kN$.*

Proof. First, we prove that any procedure for Missing-String must make at least M probes overall into the list. In particular, the procedure must probe each of the M strings in at least one bit. Otherwise, the procedure did not probe some string x at all; then, given whatever string y the procedure produces, the string x could be set equal to y, contradicting the assumption that the procedure computes Missing-String.

Therefore, since any k-probe Missing-String algorithm makes kN overall probes into the list, we must have $M \leq kN$, i.e. $k \geq M/N$. ☐

The above simple theorem can be applied to show that our (relativizing) time hierarchy with advice (Theorem 1.1) is extremely close to optimal for some parameters. For example, by setting $\alpha(n) = cn$, $\beta(n) = \log n$, $t(n) = 10n$ in Theorem 1.1, we have for all $c > 0$:

$$\mathsf{TIME}[(n^3 \log n) \cdot 2^{cn}] \not\subset \text{i.o.-}\mathsf{TIME}[O(n)]/(cn + n).$$

Using the simple lower bound of Theorem A.1, one can argue that there **is** an oracle A such that for all $c > 0$, and all unbounded functions $\alpha(n)$,

$$\mathsf{TIME}^A[2^{cn}/\alpha(n)] \subset \mathsf{TIME}^A[O(n)]/(cn + n),$$

showing that the generic argument of Theorem 1.1 cannot be improved significantly.

Let us sketch the idea of how this works, from the perspective of the Missing-String problem. We build up the oracle A in stages, starting from $n = 1$ and increasing n after each stage. On a given input length n, consider a list L of all $M = O(2^{(c+1)n})$ truth tables of length $N = O(2^n)$, one truth table for every possible query to the A oracle of type (i, x, y), where i is the index of a machine M_i^A (a string of length at most $\log(\alpha(n))/2$), x is an input string of length at most n, and y is of length at most $(c+1)n - \log(\alpha(n))/2$. (Some of the values of the bits in L have been determined in previous stages, but some have not.) Consider an algorithm B that on input (i, x') attempts to simulate an algorithm M_i^A on x' of length n, making $k \leq 2^{cn}/\alpha(n)$ queries to the list L to determine its output. If a probe into L is a value of the oracle A that was already determined in previous stages, we answer consistently; otherwise, we output that the value of A on the new query is 0.

Following the proof of Theorem A.1, for every fixed choice of i, the total number of queries into the list L (across all inputs of length up to n) is at most $\Theta(2^n \cdot 2^{cn}/\alpha(n)) < 2^{(c+1)n-\log(\alpha(n)/2} = M/\sqrt{\alpha(n)}$. Therefore there is always *at least one* truth table corresponding to strings of the form $(i, \cdot, y)$ in L that was not queried by M_i over the n-bit strings, even in previous stages. The oracle A can therefore be assigned such that this completely-unqueried truth table equals the truth table of M_i on n-bit inputs. Thus the algorithm B (and M_i for all i) on n-bit inputs can always be simulated by an A-oracle circuit of one gate of the form $A(i, x, y)$, where y has length $(c+1)n - \log(\alpha(n))/2$, and x is an input of length n. Thus B can be simulated in $O(n)$ time with oracle A and $(c+1)n$ advice.

B Fast-enough SAT Implies EXP Lower Bounds

Here we recall a proposition showing that if SAT is solvable in "sub-half-exponential" time, then circuit lower bounds for EXP follow. The proof relativizes.

Proposition B.1 ([26]). *If SAT is in $O(H(n))$ time then $\mathsf{EXP} \not\subset \mathsf{P}/poly$, for any function $H(n)$ satisfying $H(H(n^k)^2) \leq 2^n$ for all k.*

Proof. Recall $\mathsf{EXP} \subset \mathsf{P}/\text{poly}$ implies $\mathsf{EXP} = \Sigma_2\mathsf{P}$ [15], and recall that $\Sigma_2\mathsf{P} = \exists \cdot \forall \cdot \mathsf{P}$, where the $\exists$ and $\forall$ denote existential and universal alternations over $n^{O(1)}$ bits. Assuming SAT is in $O(H(n))$ time, we have $\Sigma_2\mathsf{P} \subseteq \exists \cdot \mathsf{TIME}[H(n^{O(1)})]$. Then, an $\exists \cdot \mathsf{TIME}[H(n^{O(1)})]$ computation can be expressed as a SAT instance of length at most $H(n^{O(1)})^2$. Applying the SAT algorithm once again implies

$$\mathsf{EXP} = \Sigma_2\mathsf{P} \subseteq \mathsf{TIME}[H(H(n^{O(1)})^2)] \subseteq \mathsf{TIME}[2^n \cdot n^{O(1)}],$$

contradicting the time hierarchy theorem. □

Erratum: *On Oracles and Algorithmic Methods for Proving Lower Bounds*

Jiatu Li Nikhil Vyas Ryan Williams

We report an error in a theorem of Vyas and Williams from ITCS 2023 [VW23]. Their goal was to show that the "algorithmic method" for proving circuit lower bounds from CAPP algorithms does not relativize.

Reminder (Theorem 11 of [VW23])**.** *There is an oracle A such that CAPP (for A-oracle circuits) is in P^A but $\mathsf{EXP}^{\mathsf{NP}^A}$ has polynomial-size A-oracle circuits.*

First of all, not only is there a flaw in the proof of the above theorem, but we can show that *the negation of the theorem is true*. We prove unconditionally that such an oracle *does not* exist. In other words, **there is** a "relativizing" version of the algorithmic method for circuit lower bounds.

Theorem 1. *For every oracle A such that CAPP (for A-oracle circuits) is in P^A, $\mathsf{E}^{\mathsf{NP}^A}$ cannot be computable by polynomial-size A-oracle circuits.*

Proof. Towards a contradiction, we assume that $\mathsf{E}^{\mathsf{NP}^A}$ has polynomial-size A-oracle circuits for an oracle A such that CAPP is in P^A. Chakaravarthy and Roy [CR08] proved that if NP has polynomial-size circuits, then there is a promise language $L \in \mathsf{prMA}$ such that PH collapses to P^L, and their proof relativizes. In particular, there is a promise language $L \in \mathsf{prMA}^A$ such that PH^A collapses to P^L under the assumption.

Moreover, since CAPP for A-oracle circuits is in P^A, it follows that $\mathsf{prMA}^A = \mathsf{prNP}^A$, as we can derandomize the verifier using CAPP. This further implies that PH^A collapses to $\mathsf{P}^{\mathsf{NP}^A}$ and thus $\Sigma_4\mathsf{E}^A = \mathsf{E}^{\mathsf{NP}^A}$. Since $\Sigma_4\mathsf{E}^A$ requires maximum A-oracle circuit complexity by direct diagonalization, we conclude that $\mathsf{E}^{\mathsf{NP}^A}$ also requires maximum A-oracle circuit complexity, contradicting our assumption. □

The flaw in the proof. We briefly explain the flaw in the proof of Theorem 11 of [VW23]. The proof attempts to follow an oracle construction by Fortnow [For01] that works for EXP^A.

Theorem 2 ([For01])**.** *There is an oracle A such that CAPP (for A-oracle circuits) is in P^A but E^A has linear-size A-oracle circuits.*

Fortnow [For01] constructed the oracle A in stages. We start with an all-zero oracle A_0. In the j-th stage, we will find some fixed string z_j of length $5j$, choose some queries of form $[j, z_j, \dots]$, and set the answers to the chosen queries to be 1. By cleverly selecting the queries to update, we can hardcode whether $K^A(x) = 1$ for a E^A-complete machine K^A at $A([|x|, z_{|x|}, x, 1])$ and hardcode whether C^A accepts at least $2/3$ of its input for every A-oracle circuit C^A.

At a high level, we will choose a string z_j (called the *key*) in stage j such that the updates to the queries $[j, z_j, \dots]$ do not affect the behavior of K^A on inputs of length at most j. The existence of such z_j follows from a counting argument, which does not work in [VW23] due to the existence of the NP oracle. A trick is proposed in [VW23] to resolve this issue by introducing additional updates to the queries to "force" undetermined NP oracle calls to output 1, which may not be of form $[j, z_j, \dots]$. However, the correctness of the oracle construction relies on the fact that the updates must be of form $[j, z_j, \dots]$. Concretely speaking:

- In the paragraph "Our Gap-SAT algorithm works as follows", page 22:21, we need to deal with the case where C^{A_k} is satisfiable and C^A is unsatisfiable. The proof claims that $C^{A_k}(x)$ must query some string of form $[j, z_j, \dots]$ for $j > k$. This is true in [For01], since the only updates to the oracle are of form $[j, z_j, \dots]$. However, in [VW23], the additional updates may also change the satisfiability of C^A.

- In the paragraph "If no z in S has the property", page 22:21, we consider the case where C^{A_k} is unsatisfiable and no $z \in S$ is a "key". It is claimed that C^{A_k} and C^k differ on at most one half of the inputs. This relies on the observation that $C^{A_k}(x) \neq C^A(x)$ only if C^{A_k} makes a query with key z_j for some $j > k$, which is true in [For01]. However, given the existence of the additional updates, it is also possible that some of the queries made by $C^{A_k}(x)$ are changed by the additional updates.

References

[CR08] Venkatesan T. Chakaravarthy and Sambuddha Roy. Finding irrefutable certificates for s_2^p via arthur and merlin. In Susanne Albers and Pascal Weil, editors, *STACS 2008, 25th Annual Symposium on Theoretical Aspects of Computer Science, Bordeaux, France, February 21-23, 2008, Proceedings*, volume 1 of *LIPIcs*, pages 157–168. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany, 2008.

[For01] Lance Fortnow. Comparing notions of full derandomization. In *Proceedings of the 16th Annual IEEE Conference on Computational Complexity, Chicago, Illinois, USA, June 18-21, 2001*, pages 28–34. IEEE Computer Society, 2001.

[VW23] Nikhil Vyas and Ryan Williams. On oracles and algorithmic methods for proving lower bounds. In Yael Tauman Kalai, editor, *14th Innovations in Theoretical Computer Science Conference, ITCS 2023, January 10-13, 2023, MIT, Cambridge, Massachusetts, USA*, volume 251 of *LIPIcs*, pages 99:1–99:26. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023.

ECCC ISSN 1433-8092

https://eccc.weizmann.ac.il