Electronic Colloquium on Computational Complexity, Revision 1 of Report No. 167 (2019)

Unique Properties, Lower bounds & Derandomization

ANANT DHAYAL, University of California San Diego, USA
RUSSELL IMPAGLIAZZO, University of California San Diego, USA

The most difficult tasks in computational complexity are: proving uniform/non-uniform lower bounds, designing fast satisfiability/learning algorithms, and derandomizing probabilistic algorithms. Connections have been drawn between them to study the relative hardness of these tasks: (a) Uniform to non-uniform lower bounds, famously known as Karp-Lipton style theorems (KLT) [43]; (b) Fast algorithms to lower bounds; (c) Lower bounds to derandomization. Such connections were initially known for the class EXP, and then later were extended to NEXP. The key in the extension was the "easy-witness lemma (EWL) for NEXP" [31]. We extend these connections to the intermediate class UEXP, and some related classes, by deriving similar EWLs for them. In the 'fast algorithms to lower bounds' connection we also provide translation results that generalize the lower bound frameworks for unrestricted Boolean circuits, to all *typical* circuit classes, in a black box fashion (i.e. generalization only depends on the assumption set of the framework and not its working).

Circuit lower bound techniques that entail natural properties of Razborov and Rudich [61] are called natural, and are known to contradict widely believed cryptographic assumptions in the course of proving strong lower bounds. Thus attempts have been made to understand un-natural techniques. Natural properties satisfy three conditions: usefulness, constructiveness, and largeness. Usefulness is implicit in any lower-bound technique. In [57, 75] it was shown that P-constructivity is implicit in any NEXP or NEXP $\cap$ Co-NEXP lower bound technique. In this paper we introduce a new notion called unique properties: properties that contain exactly one element and thus avoid largeness. We show that P-constructivity and uniqueness are implicit in any UEXP or UEXP $\cap$ Co-UEXP lower bound technique. For the case of NP-constructivity, for different lower bound settings, we establish equivalences between: properties with arbitrary largeness and unique properties. In the process we obtain a variety of EWLs and KLTs for NEXP $\cap$ Co-NEXP and related classes.

Equivalences between deterministic lower bounds and derandomization has been studied extensively in the past. This was extended to non-deterministic circuits in [7] using an improved high-end KLT for NP/*poly*. Using the *higher* Arthur-Merlin classes from [38] we generalize this KLT to *general* circuit classes and obtain: (i) a wide spectrum of lower bounds vs derandomization equivalences; (ii) lower bounds for *higher* Arthur-Merlin classes against *general* circuit classes. Our KLT extends to EXP and UEXP, but not to NEXP due to the lack of EWLs. For the special case of NEXP $\not\subset$ (NP$\cap$Co-NP)/*poly* we prove equivalence with: witness lower bound, an uniform lower bound, and various useful properties. We extend results from [75] to show that: super-polynomial savings in exhaustive search for certain problems imply NEXP $\not\subset$ (NP $\cap$ Co-NP)/*poly*. This new connection yields an unconditional lower bound against a special restriction of non-deterministic ACC circuits.

CCS Concepts: • **Theory of computation → Complexity classes**; **Circuit complexity**.

Additional Key Words and Phrases: lower bounds, derandomization, KLT, EWL, unique properties, *higher* Arthur-Merlin classes, *typical* circuit classes, *general* circuit classes

ACM Reference Format:
Anant Dhayal and Russell Impagliazzo. 2020. Unique Properties, Lower bounds & Derandomization. 1, 1 (May 2020), 55 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

Authors' addresses: Anant Dhayal, adhayal@eng.ucsd.edu, University of California San Diego, La Jolla, CA, USA; Russell Impagliazzo, russell@eng.ucsd.edu, University of California San Diego, La Jolla, CA, USA.

XXXX-XXXX/2020/5-ART $15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

ISSN 1433-8092

1 INTRODUCTION

We often think of algorithm design and lower bounds as being antithetical, but there have been a series of results showing that in certain circumstances, efficient algorithms imply circuit lower bounds [31, 41, 53, 73, 74, 76]. Unfortunately, most of these results are only known to show circuit lower bounds or conditional lower bounds in relatively large complexity classes such as NEXP or E^{NP} (although [53] extends this to scaled-down versions of these classes). This raises the question of whether similar lower bounds for other classes, ideally deterministic or randomized classes such as EXP or BPEXP, could be obtained through improved algorithms. Here, we consider possible extensions to the class UEXP of languages recognized by unambiguous non-deterministic machines, and to related classes. Since UEXP lies between EXP and NEXP, lower bounds for UEXP based on algorithms would be progress towards making similar connections for EXP.

A key technique used to make these connections is the "easy witness technique" [31, 40, 53]. The easy witness technique relates the circuit complexity of witnesses for non-deterministic algorithms to the circuit or algorithmic complexity of decision problems. We give easy witness lemmas (or EWL for short) for UEXP and related classes; these are simpler than the analogous results for NEXP, which needed a rather indirect argument. We then explore consequences of these EWLs to normal forms for circuit lower bounds in these classes, in terms of useful properties in the sense of Razborov and Rudich [61] (see also [75]). We take a detour to explore the nature of these special properties we get, and draw some interesting conclusions. Then we show how different combinations of fast learning and SAT algorithms for a circuit class would imply a circuit lower bound for UEXP. We also show that: when circuit lower bounds for NEXP are replaced with circuit lower bounds for UEXP, we get better derandomization results.

An alternate way in which we can extend this "algorithm design vs lower bounds" connection is: by keeping the class NEXP same, but increasing the complexity of circuit classes against which we want to obtain the lower bounds. Our next ideal choice should be non-deterministic or single-valued non-deterministic circuits. We consider the intermediate class $(\mathsf{NP} \cap \mathsf{Co\text{-}NP})/poly$. We again take the route of easy witness technique and prove results that are analogous to the UEXP case. Here again, we take a detour and generalize the lower bounds vs derandomization connection to general circuit classes and get new unconditional lower bounds against fix-polynomial general circuit classes.

In the $(\mathsf{NP} \cap \mathsf{Co\text{-}NP})/poly$ extension, our results are much tighter compared to the UEXP extension. So we use it to establish an unconditional lower bound for NEXP against an ACC analogue of $(\mathsf{NP} \cap \mathsf{Co\text{-}NP})/poly$ (with non-determinism limited to sub-polynomial). Among numerous other results, we extend NEXP Karp-Lipton theorems for $\mathsf{P}/poly$ and $(\mathsf{NP} \cap \mathsf{Co\text{-}NP})/poly$ to $\mathsf{E}^{\mathsf{NP}}_{||}$. Using this we get better gap theorem for MA, and new gap theorems for $\mathsf{MA}^{\mathsf{NP} \cap \mathsf{Co\text{-}NP}}$ and CAPP. We also establish downward separation type results where: lower bounds for exponential classes imply lower bounds for sub-exponential classes.

1.1 EWL and KLT for UEXP

As a first step in extending the NEXP lower bound frameworks to UEXP, we design EWLs for UEXP and related classes. One crucial intermediate step used in many lower bound frameworks is, Karp-Lipton style theorem [43] (or KLT for short): it relates the non-uniform and uniform complexities of classes. An example is Meyer's Theorem from [43]: $\mathsf{EXP} \subset \mathsf{P/poly} \implies \mathsf{EXP} = \Sigma_2^{\mathsf{P}}$. An extension to NEXP was given in [31], using the easy witness technique.

Our results from Section 3: We derive analogous EWL and KLT for UTIME. Our results are fine-grained in terms of the time and size parameters, and work for all *typical* non-uniform circuit classes. We look at EWL as a special search to decision reduction, where the output of the search

problem is canonical in some natural way. We prove that such special search problems for UTIME verifiers belong to UTIME itself (for NE they belong to E^{NP} and are complete in some sense [34]). This gives us the desired EWL: UEXP $\subset$ P/poly implies UEXP verifiers have easy-witnesses, i.e., witnesses encoded as truth-tables of polynomial-size circuits. This in turn gives us the desired KLT: UEXP $\subset$ P/poly $\implies$ UEXP = MA. Similar results for UEXP $\cap$ Co-UEXP and FewEXP are also derived.

1.2 Useful properties

Before we extend the NEXP lower bound frameworks to UEXP, we discuss an important barrier that could limit the progress of any NEXP and UEXP lower bound technique.

Razborov and Rudich [61] defined the concept of natural property as a formalization of a barrier that circuit lower bounds need to circumvent. Natural Proofs (or properties) satisfy three conditions: they are constructive (an efficient algorithm $\mathcal{A}$ is embedded in them), have largeness ($\mathcal{A}$ accepts a large fraction of strings), and are useful ($\mathcal{A}$ rejects all strings which are truth tables of small circuits). Circuit lower bound techniques that entail natural properties are called natural, and are known to contradict widely believed cryptographic assumptions in the course of proving strong lower bounds. Thus they are self-limiting, and in order to prove stronger circuit lower bounds the techniques should be un-natural in some sense. Unfortunately, the vast majority of known circuit lower bound techniques are natural and can't be applied even to low-level complexity classes such as TC_0 [48, 52, 54].

Williams [75] showed, using the NEXP EWL, that any lower bound for a problem in NEXP implies a property with two of the conditions (constructivity and usefullness) of Razborov and Rudich, but not necessarily the third (largeness). So while natural properties for circuit classes cannot exist if there are strong pseudo-random functions in the class, it seems likely that dropping largeness means that such properties do exist.

Our results from Section 4: To understand properties that avoid largeness, we look at properties that go to the other extreme. We introduce a new notion called unique properties, those that contain exactly one function of each input length. Useful unique properties are implicitly proving a circuit lower bound for a specific function: the one function that has the property, but might not explicitly spell out which function the lower bound holds for.

We extend the proofs in [57, 75] to show that: obtaining NEXP lower-bounds or lower bounds for NEXP witnesses is equivalent to obtaining useful NP-unique properties that use $\log n$ advice; obtaining NEXP $\cap$ Co-NEXP lower-bounds is equivalent to obtaining useful NP-unique properties; and obtaining lower bounds for NEXP $\cap$ Co-NEXP witnesses is equivalent to obtaining useful NP properties (which may or may not be unique).

The next obvious step is to reduce the constructivity to P. In attempt to understand P-unique properties, we move to UEXP lower bounds. We show that P-constructivity and uniqueness both are unavoidable for UEXP lower bounds. We prove, UEXP $\cap$ Co-UEXP $\not\subset C$ if and only if there is a P-unique property useful against C. We also establish equivalences between lower bounds against UEXP (with and without advice), and the existence of different restrictions of P-unique properties that use advice.

Our results from Section 5: Our results from Section 4 show that obtaining NP/log n properties useful against polynomial size circuits, is equivalent to obtaining NP/log n-unique properties useful against polynomial size circuits. This can be viewed as some type of isolation of properties: where properties have equivalent unique-properties. In Section 5 we extend this isolation and get rid of the log n amount of advice on the expense of diluting the usefulness of the properties. More precisely we show the following equivalence/translations for properties useful against fix-polynomial size circuits:

(1) Obtaining NP/$O(1)$ useful properties of arbitrary largeness, is equivalent to obtaining NP/$O(1)$-unique useful properties.
(2) Obtaining promise-NP useful properties of arbitrary largeness, is equivalent to obtaining promise-NP-unique useful properties. Here, by promise we mean that the property is only required to satisfy the size restrictions when its useful.
(3) Obtaining NP properties of arbitrary largeness that are useful for all input lengths, is at least as difficult as obtaining NP-unique properties that are useful infinitely often. Note that by default, the properties are only useful infinitely often.

These new equivalences, combined with the equivalences from Section 4 imply: various NEXP ∩ Co-NEXP lower bounds are equivalent to corresponding NEXP ∩ Co-NEXP witness lower bounds. The equivalences between these lower bounds is nothing but a collection of EWLs for NEXP ∩ Co-NEXP. Using these EWLs we also get some interesting KLTs for NEXP ∩ Co-NEXP.

To capture the case of promise properties, we define a promise version of NEXP ∩ Co-NEXP, such that the NEXP and Co-NEXP algorithms only need to complement each other, infinitely often. We name this class ip-(NEXP ∩ Co-NEXP), where 'ip' stands for 'infinitely-often promise'. This is inspired by the robustly-often promise classes defined in [18]. We also get EWL and KLT for this class. From the various equivalences established between unique-properties and lower bounds: we infer that the lower bounds for ip-(NEXP ∩ Co-NEXP) strictly lie between the lower bounds for NEXP and NEXP ∩ Co-NEXP, unless new unknown collapses are proved. Later we will see that lower bounds for this class perfectly capture the derandomization (of certain randomized classes) without advice.

1.3 Fast algorithms to non-uniform lower bounds

Now we extend the NEXP lower bound frameworks to UEXP.

Ckt-SAT is the canonical NP complete problem [21, 49]. No algorithm significantly faster than the trivial brute-force algorithm is known to solve Ckt-SAT. One related interesting question is: does non-determinism helps in solving Ckt-TAUT faster than brute-force? After years of effort [59, 60], no progress in beating the brute-force strategy (significantly), lead to the formulation of several conjectures [19, 32, 33].

In [73] it was shown that: if we get super-polynomial savings in any non-deterministic Ckt-TAUT algorithm for polynomial size circuits, then NEXP $\not\subset$ SIZE($poly$). This shows that even a small progress in SAT algorithms is at least as hard as proving non-uniform circuit lower bounds. But to optimistic people, it gives a chance to prove lower bounds. In fact, this framework was used to prove lower bounds for weak circuit classes by designing fast SAT algorithms for them [74–76].

Our results from Section 6: We discuss possible extensions of the framework from [73], to yield UEXP lower bounds. We show similar extensions for the "learning vs lower bounds" framework of [24]. We use the UTIME EWL and KLT in our extensions.

Our extensions are weak in the sense that they demand fast algorithms that beat brute-force with a better margin than what was required in the NEXP case. As such algorithms have been designed, or seem plausible in the near future, only for restricted classes, we give results that generalize the lower bound frameworks. We provide translation results that generalize the lower bound frameworks for unrestricted Boolean circuits, to all *typical* circuit classes, in a black box fashion. That is, generalization only depends on the assumption set of the framework and not its working.

1.4 Unconditional lower bounds from KLTs

Before we extend the NEXP lower bound framework to $(\mathsf{NP} \cap \mathsf{Co\text{-}NP})/poly$, we design KLT for $(\mathsf{NP} \cap \mathsf{Co\text{-}NP})/poly$ and higher circuit classes. In this section we take a detour, and use these KLTs to prove some unconditional lower bounds against general circuit classes.

Proving lower bounds for nonuniform circuits remains one of the most difficult tasks in computational complexity. Proving super polynomial lower bounds for NP would separate NP from P and BPP. Proving fix-polynomial lower bounds for NP would separate NEXP form BPP. Currently we don't even have super-linear lower bounds for NP.

We have fix-polynomial lower bounds against higher classes. This work was started by Kannan in 1982 [42]. He used the low-end KLT, $\mathsf{NP} \subset \mathsf{P}/poly \implies \mathsf{PH} = \Sigma_2^{\mathsf{P}} \cap \Pi_2^{\mathsf{P}}$ [43], to prove fix-polynomial lower bounds for $\Sigma_2^{\mathsf{P}} \cap \Pi_2^{\mathsf{P}}$. New lower bounds were proved using new low-end KLTs [13, 16, 44, 51, 71]. The improvements made along the inclusion chain, $\mathsf{NP} \subseteq \mathsf{P}_{||}^{\mathsf{NP}} \subseteq \mathsf{P}^{\mathsf{NP}} \subseteq \mathsf{S}_2^{\mathsf{P}} \subseteq \mathsf{ZPP}^{\mathsf{NP}} \subseteq \Sigma_2^{\mathsf{P}} \cap \Pi_2^{\mathsf{P}}$, stopped at $\mathsf{S}_2^{\mathsf{P}}$ in [16] (based on an observation by Sengupta). Fix-polynomial lower bounds for NP and $\mathsf{P}_{||}^{\mathsf{NP}}$ are equivalent [27], and fix-polynomial lower bounds for P^{NP} are equivalent to obtaining an improved low-end KLT [20].

Another set of inclusions is $\mathsf{NP} \subseteq \mathsf{MA} \subseteq \mathsf{S}_2^{\mathsf{P}}$, , where the relationship between MA and P^{NP} (or $\mathsf{P}_{||}^{\mathsf{NP}}$) is not known. Santhnam [63] gave the fix-polynomial lower bound for MA/1 and prMA, using the high-end KLT of [9]: $\mathsf{PSPACE} \subset \mathsf{P}/poly \implies \mathsf{PSPACE} = \mathsf{MA}$. Fix-polynomial lower bounds for NP imply an improved high-end KLT [20].

We extend these results to general circuit classes. Lower bounds for these classes entail answers to equally important questions. For instance, proving super-polynomial lower bounds for NEXP (*resp.* Σ_2^{P}) against non-deterministic circuits would separate NEXP (*resp.* Σ_2^{P}) form AM. As KLTs seem the only way of proving lower bounds, we first establish KLTs for general circuit classes. One can find the definitions of SV non-deterministic circuits, (co-)non-deterministic circuits, adaptive/non-adaptive SAT-oracle circuits in [64]. We also define a new class of circuits to capture the class $(\mathsf{NP} \cap \mathsf{Co\text{-}NP})/poly$: promise SV non-deterministic circuits. We generalize these definitions further to the polynomial hierarchy (see Section 2 for formal definitions).

For non-deterministic circuits: (i) the best known low-end KLT [17] yields fix-polynomial lower bounds for $\mathsf{S}_2 \cdot \mathsf{P}^{\mathsf{NP}}$; (ii) the high-end KLT, $\mathsf{PSPACE} \subset \mathsf{NP}/poly \implies \mathsf{PSPACE} = \mathsf{M}(\mathsf{AM}||\mathsf{Co\text{-}NP})$, where $\Sigma_2^{\mathsf{P}} \subseteq \mathsf{M}(\mathsf{AM}||\mathsf{Co\text{-}NP}) \subset \mathsf{MA}^{\mathsf{NP}} \subset \mathsf{S}_2 \cdot \mathsf{P}^{\mathsf{NP}}$, was given in [7]; and (iii) this KLT was used to show fix-polynomial lower bounds for $\mathsf{prM}(\mathsf{AM}||\mathsf{Co\text{-}NP})$ in [65].

Our results from Section 7: We improve the result of [65] by replacing non-deterministic circuits with non-adaptive SAT-oracle circuits. We need two intermediate results for this:

(1) For the case $\mathsf{PSPACE} \not\subset \mathsf{P}_{||}^{\mathsf{NP}}/poly$: We use Santhanam's framework, and thus need his special PSPACE-complete language, that was instance-checkable with same-length queries. The 'same-length queries' part was essential for his argument. Using the fact that this language is non-adaptively random-self-reducible, we show that there is a fix constant c, such that any $s(n)$-size non-adaptive SAT-oracle circuit for the language has an equivalent $s(n)^c$-size non-deterministic circuit. We use the tools from [22] in similar fashion as used in [64]. The size $s(n)^c$ is essential, due to the same reason 'same-length queries' were essential, and thus we can't afford $s(n^c)^c$.

(2) For the other case: We use the improved KLT, $\mathsf{PSPACE} \subset \mathsf{P}_{||}^{\mathsf{NP}}/poly \implies \mathsf{PSPACE} = \mathsf{M}(\mathsf{AM}||\mathsf{Co\text{-}NP})$. This was implicit in [7, 23].

We generalize the KLT from [7] to higher Arthur-Merlin classes [38] and general circuit classes: $\mathsf{PSPACE} \subset \mathsf{P}_{||}^{\Sigma_i^{\mathsf{P}}}/poly \implies \mathsf{PSPACE} = \mathsf{M}(\mathsf{BP} \cdot \Sigma_i^{\mathsf{P}}||\Pi_i^{\mathsf{P}})$. Using this we prove lower bounds for $\mathsf{prM}(\mathsf{BP} \cdot \Sigma_i^{\mathsf{P}}||\Pi_i^{\mathsf{P}})$ against fix-polynomial size non-adaptive Σ_i^{P}-oracle circuits. Note that this class is

contained in $\mathsf{MA}^{\Sigma_i^\mathsf{P}}$ and $\mathsf{S}_2 \cdot \mathsf{P}^{\Sigma_i^\mathsf{P}}$: the lowest classes against which this lower bound was known until now.

Among some other fix-polynomial lower bounds, we also show: for constant k, $\mathsf{prAM} \not\subset (\mathsf{NTIME}(n^k) \cap \mathsf{Co\text{-}NTIME}(n^k))/n^k$. In our terminology: prAM doesn't have fix polynomial promise SV non-deterministic circuits. Note that, in this lower bound, the class has to have one language that beats all of the $(\mathsf{NTIME}(n^k) \cap \mathsf{Co\text{-}NTIME}(n^k))/n^k$ algorithms. The lack of complete problems in $(\mathsf{NTIME}(n^k) \cap \mathsf{Co\text{-}NTIME}(n^k))$ makes this task a bit difficult.

We also improve the current state of art for super-polynomial lower bounds. $\mathsf{MAEXP} \not\subset \mathsf{P}/poly$ [14] and $\mathsf{AMEXP} \not\subset (\mathsf{NP} \cap \mathsf{Co\text{-}NP})/poly$ [72] are the best results known before this work. From our KLTs we get: (i) $\mathsf{MAE}^{\mathsf{NP}\cap\mathsf{Co\text{-}NP}} \not\subset (\mathsf{NP} \cap \mathsf{Co\text{-}NP})/poly$, and (ii) $\mathsf{M}(\mathsf{AME}||\mathsf{Co\text{-}NE}) \not\subset \mathsf{P}_{||}^{\mathsf{NP}}/poly$. We also extend these lower bounds : (i) to sub-half-exponential circuit sizes; (ii) to super-half-exponential versions of the protocols; (iii) to general circuit classes and higher Arthur-Merlin protocols.

1.5 Derandomization vs lower bounds

In this section we extend the "lower bounds to derandomization" connection of NEXP, to UEXP and general circuit classes.

Relationship between derandomization and uniform/non-uniform lower bounds has been studied extensively in the past [7, 9, 18, 31, 35, 36, 55, 56, 66, 69, 75]. In this section we only focus on the lower end of this spectrum: derandomization that requires sub-exponential time.

In [35] it was shown:

$$\mathsf{EXP} \neq \mathsf{BPP} \iff \mathsf{BPP} \subset \cap_{\epsilon>0} \text{io-Heur-}\mathsf{TIME}(2^{n^\epsilon}) \tag{1}$$

In [31] its NEXP version was proved:

$$\mathsf{NEXP} \not\subset \mathsf{SIZE}(poly) \iff \mathsf{NEXP} \neq \mathsf{MA} \iff \mathsf{MA} \subset \cap_{\epsilon>0} \text{io-}\mathsf{NTIME}(2^{n^\epsilon})/n^\epsilon \tag{2}$$

In [75] another variant was proved:

$$\mathsf{NEXP} \neq \mathsf{BPP} \iff \mathsf{BPP} \subset \cap_{\epsilon>0} \text{io-Heur-}\mathsf{NTIME}(2^{n^\epsilon})/n^\epsilon \tag{3}$$

In [75] this was also extended to REXP lower bounds:

$$\mathsf{REXP} \neq \mathsf{BPP} \iff \mathsf{BPP} \subset \cap_{\epsilon>0} \text{io-Heur-}\mathsf{ZPTIME}(2^{n^\epsilon})/n^\epsilon \tag{4}$$

In [7], an extension to non-deterministic circuits was given:

$$\Sigma_2\mathsf{EXP} \not\subset \mathsf{NP}/poly \iff \mathsf{prAM} \subset \cap_{\epsilon>0} \text{io-}\Sigma_2\mathsf{TIME}(2^{n^\epsilon})/n^\epsilon \tag{5}$$

Our results from Section 8: We extend equations (2,3,4) to UTIME and ZUTIME using results from the Sections 3 and 4 (lower bounds vs unique properties). Unfortunately, we only get the "lower bounds to derandomization" connection and not the reverse connection. It is due to the lack of complete languages and strong hierarchies. For the case of REXP this problem was circumvented by using the relationship $\mathsf{NEXP} \neq \mathsf{BPP} \iff \mathsf{REXP} \neq \mathsf{BPP}$ (due to $\mathsf{NP} \subseteq \mathsf{BPP} \iff \mathsf{NP} = \mathsf{RP}$ [47]).

We use our general high-end KLT and extend this equivalence to: lower bounds against general circuit classes vs derandomization of higher Arthur-Merlin classes. We also extend equation (2) to lower bounds against $(\mathsf{NP} \cap \mathsf{Co\text{-}NP})/poly$ and other related circuit classes, and generalize all these relations too. In [18] equation (2) was extended to a tighter equivalence: $\mathsf{NE} \cap \mathsf{Co\text{-}NE}$ lower bounds vs derandomization without advice. We extend this tighter connection as well: to non-deterministic and higher circuit classes. For this, we use the promise class ip-$(\mathsf{NE} \cap \mathsf{Co\text{-}NE})$.

Graph non-isomorphism is in AM. In [1] it was shown that the complement of Boolean isomorphism is in $\mathsf{BP} \cdot \Sigma_2^\mathsf{P}$. Boolean isomorphism is the problem of deciding whether the input pair of Boolean circuits is isomorphic or not (upto renaming of the inputs). In [38], apart from other results on higher Arthur-Merlin classes, they also showed that non-Boolean isomorphism for Σ_i^P-oracle

circuits is in $\mathsf{BP} \cdot \Sigma_{i+2}^{\mathsf{P}} = \mathsf{AM}^{\Sigma_{i+1}^{\mathsf{P}}}$. So our general connection can also be thought of as: Boolean non-isomorphism for Σ_i^{P}-oracle circuits have non-deterministic sub-exponential proofs (verifiable with $\Sigma_{i+2}^{\mathsf{P}}$ oracles), unless PH collapses.

Finally, we combine these connections with the unconditional fix-polynomial lower bounds that we established in the Section 7, to yield downward separation type results where: lower bounds for exponential classes imply lower bounds for sub-exponential classes.

1.6 The special case of $(\mathsf{NP} \cap \mathsf{Co\text{-}NP})/poly$

Now we extend the NEXP lower bound frameworks to $(\mathsf{NP} \cap \mathsf{Co\text{-}NP})/poly$, and also prove some other interesting results.

For the special case of $(\mathsf{NP} \cap \mathsf{Co\text{-}NP})/poly$, we are able to prove an EWL. In the case of $\mathsf{P}/poly$, EWL was used to extend the high-end KLT (for PSPACE and EXP) to NEXP. For the case of $(\mathsf{NP} \cap \mathsf{Co\text{-}NP})/poly$, we already have a KLT for NEXP, in which it collapses to AM [72]. Now from [31], we get an EWL using the fact that NEXP search problems are contained in EXP, which is contained in $(\mathsf{NP} \cap \mathsf{Co\text{-}NP})/poly$. For more complex classes, such as $\mathsf{NP}/poly$ or $\mathsf{P}^{\mathsf{NP}}/poly$, the extensions to NEXP are not easy to obtain, because KLT for PSPACE collapses it to higher classes than AM. The hard-witnesses can't derandomize these higher classes in NSUBEXP or NE, which was essential for the arguments used in [31].

Our results from Section 9: We use the EWL for $(\mathsf{NP} \cap \mathsf{Co\text{-}NP})/poly$ to establish equivalences between: (i) various witness lower bounds for NEXP, (ii) various properties against $(\mathsf{NP} \cap \mathsf{Co\text{-}NP})/poly$, (iii) $\mathsf{NEXP} \not\subset (\mathsf{NP} \cap \mathsf{Co\text{-}NP})/poly$, (iv) $\mathsf{NEXP} \not\subset \mathsf{MA}^{\mathsf{NP} \cap \mathsf{Co\text{-}NP}}$, and (v) derandomization of $\mathsf{prBPP}^{\mathsf{NP} \cap \mathsf{Co\text{-}NP}}$ and $\mathsf{prMA}^{\mathsf{NP} \cap \mathsf{Co\text{-}NP}}$. We use the PSPACE KLT of [17] to extend the NEXP KLT of [72] to $\mathsf{MA}^{\mathsf{NP} \cap \mathsf{Co\text{-}NP}}$, which is essential for our results (we also argue $\mathsf{MA}^{\mathsf{NP} \cap \mathsf{Co\text{-}NP}} \subseteq \mathsf{AM}$).

We also extend the results to $\mathsf{E}_{||}^{\mathsf{NP}}$. Specifically, we give $\mathsf{E}_{||}^{\mathsf{NP}}$ KLTs (and their converses) for $\mathsf{P}/poly$ and $(\mathsf{NP} \cap \mathsf{Co\text{-}NP})/poly$.

We also replicate the $\mathsf{NEXP} \cap \mathsf{Co\text{-}NEXP}$ vs $\mathsf{P}/poly$ results to: $\mathsf{NEXP} \cap \mathsf{Co\text{-}NEXP}$ vs $(\mathsf{NP} \cap \mathsf{Co\text{-}NP})/poly$.

Using the EWL we extend the "algorithm design vs lower bounds" connection of [73] to $(\mathsf{NP} \cap \mathsf{Co\text{-}NP})/poly$. We show that: super polynomial savings in non-deterministic, tautology or CAPP algorithms, for $(\mathsf{NP} \cap \mathsf{Co\text{-}NP})$-oracle circuits, imply $\mathsf{NEXP} \not\subset (\mathsf{NP} \cap \mathsf{Co\text{-}NP})/poly$.

Note that, for any $A \in \mathsf{NP} \cap \mathsf{Co\text{-}NP}$, TAUT (or CAPP) for poly-size A-oracle circuits has a trivial algorithm that runs in non-deterministic $poly(n)2^n$-time: for all 2^n inputs, non-deterministically guess the answers to all the oracle queries, and guess their certificates (for A for any positive answer, for $\overline{A}$ for any negative answer).

Our assumption is weak in the sense that, we assume fast algorithm for harder problems. But, we get a stronger consequence.

We also extend our results to fast tautology algorithms for determinsitic circuits. Here by fast we mean, non-deterministic sub-exponential. In fact: (i) our algorithm only needs to work infinitely often, (ii) can be heuristic (i.e. works for high fraction of inputs on any poly-sampleable distribution), and (iii) is allowed to use sub-polynomial advice.

From the previous sections, we know that:

(1) $\mathsf{CAPP} \in \cap_{\epsilon>0}\ \text{io-}\mathsf{NTIME}(2^{n^\epsilon})/n^\epsilon \iff \mathsf{NEXP} \not\subset \mathsf{P}/poly$
(2) $\mathsf{CAPP}^{\mathsf{NP} \cap \mathsf{Co\text{-}NP}} \subset \cap_{\epsilon>0}\ \text{io-}\mathsf{NTIME}(2^{n^\epsilon})/n^\epsilon \iff \mathsf{NEXP} \not\subset (\mathsf{NP} \cap \mathsf{Co\text{-}NP})/poly$

Our result can be seen as an improvement of (2), as we are replacing the $(\mathsf{NP} \cap \mathsf{Co\text{-}NP})$-oracle circuits with deterministic circuits, but we are also replacing CAPP by TAUT. From (1), this indicates that TAUT is a harder problem than CAPP, as same (or less) improvement in TAUT algorithms yield stronger circuit lower bounds. From (2), this indicates that fast TAUT algorithm implies fast (or faster) CAPP algorithm for a more complex circuit class.

Such evidence was also implicit in [73]:

(1) $\mathtt{TAUT} \in \mathtt{NTIME}(2^n/sp(n)) \implies \mathtt{CAPP} \in \cap_{\epsilon>0}\ \text{io-}\mathtt{NTIME}(2^{n^\epsilon})/n^\epsilon$
(2) $\mathtt{TAUT} \in \cap_{\epsilon>0}\ \text{io-Heur-}\mathtt{NTIME}(2^{n^\epsilon})/n^\epsilon \implies \mathtt{CAPP} \in \cap_{\epsilon>0}\ \text{io-}\mathtt{NTIME}(2^{n^\epsilon})/n^\epsilon$

Here sp is any super-polynomial function. Our result is a strict improvement to (2), and (1) doesn't give the same message because: the implied CAPP algorithm is much faster, but it is weaker in the sense that, it uses sub-polynomial advice and works only infinitely often.

As a side product of our arguments, we also get gap theorems for $\mathtt{MA}^{\mathtt{NP}\cap\mathtt{Co\text{-}NP}}$ and $\mathtt{CAPP}^{\mathtt{NP}\cap\mathtt{Co\text{-}NP}}$, and an improved gap theorem for MA. The arguments are analogous to the ones given in [31, 73].

We also use the fast ACC algorithm from [74] to prove an unconditional lower bound: NEXP is not contained in the ACC analogue of $(\mathtt{NP}\cap\mathtt{Co\text{-}NP})/poly$ (with sub-polynomial non-determinism). We also show that: an extension to ACC analogue of SV non-deterministic circuits (with sub-polynomial non-determinism), will also imply an extension to ACC analogue of non-deterministic circuits (with sub-polynomial non-determinism).

2 PRELIMINARIES

Basic notations: Unless a new range is declared during the usage, we use t for time-constructible functions $n \le t(n) \le 2^{n^{O(1)}}$, a for advice functions $0 \le a(n) \le poly(n)$, s for circuit sizes (number of wires) $n \le s(n) \le 2^n$. For language L we use $L_n = \{x \mid x \in L \wedge |x| = n\}$ to denote the n^{th}-slice of L (or the characteristic function of L on n-length inputs). For circuit C, we use $tt(C)$ to denote its truth-table, and $|C|$ to denote its size.

Uniform classes: We assume that the reader is familiar with the standard complexity classes such as P, NP, RP, UP, BPP, ZPP, MA, AM, MAM, $\Sigma_i^{\mathsf{P}}, \Pi_i^{\mathsf{P}}$, PH (see [5]) and their corresponding complexity measures, DTIME, NTIME, RTIME, UTIME, BPTIME, ZPTIME, MATIME, AMTIME, Σ_iTIME, Π_iTIME. For the special cases of $\Sigma_i^{\mathsf{P}}, \Pi_i^{\mathsf{P}}$, we omit the superscript P and simply write Σ_i, Π_i. For $\mathtt{C} = \mathtt{D}, \mathtt{N}, \mathtt{R}, \mathtt{U}, \mathtt{BP}, \mathtt{ZP}$, MA, AM, Σ_i, Π_i: $\mathtt{CTIME}(t)$ denotes the class of languages accepted by CTIME machines that run in $O(t)$ time. CE, CEXP, CSUBE, CSUBEXP, denote the classes $\cup_{c\ge0}\mathtt{CTIME}(2^{cn})$, $\cup_{c\ge0}\mathtt{CTIME}(2^{n^c})$, $\cap_{c\ge0}\mathtt{CTIME}(2^{cn})$, $\cap_{c\ge0}\mathtt{CTIME}(2^{n^c})$ respectively. We assume familiarity with Ckt-SAT (circuit satisfiability), Ckt-TAUT (circuit tautology), k-SAT, k-TAUT, CNF-SAT, DNF-TAUT, Σ_i-SAT and Π_i-SAT.

Zero-error classes: We extend the concept of zero-error class to non-deterministic and unambiguous classes. We do this for the sake of clarity in certain arguments, and specially for distinguishing between certain non-uniform classes.

$L \in \mathtt{ZCTIME}(t)$ if there exists a Turing machine M, that for input (x, y) with $|x| = n$ and $|y| = c \cdot t(n)$ for some constant c, runs in time $c \cdot t(n)$ for $\forall n \in \mathbb{N}$, and whose output lies in $\{1, ?\}$ if $x \in L$, and in $\{0, ?\}$ if $x \notin L$. Additionally, the quantity $\Sigma_{y:M(x,y)\in\{0,1\}}1$ is equal to: 1 for $\mathtt{C} = \mathtt{U}$ (*uniqueness*), $\ge \frac{1}{2} \times 2^{c\cdot t(n)}$ for $\mathtt{C} = \mathtt{R}$ (*largeness*), ≥ 1 for $\mathtt{C} = \mathtt{N}$ (*existence*). The verifier/predicate corresponding to M is called zero-error non-deterministic. For the special cases of $\mathtt{C} = \mathtt{U}$ and $\mathtt{C} = \mathtt{R}$, its called zero-error unambiguous and zero-error randomized respectively.

Remark : $\mathtt{ZRTIME} = \mathtt{ZPTIME}$, and for $\mathtt{C} = \mathtt{N}, \mathtt{R}, \mathtt{U}$, $\mathtt{ZCTIME}(t) = \mathtt{CTIME}(t) \cap \mathtt{Co\text{-}CTIME}(t)$ follows by a similar argument that shows $\mathtt{ZPTIME}(t) = \mathtt{RTIME}(t) \cap \mathtt{Co\text{-}RTIME}(t)$.

Circuit classes: We assume basic familiarity with Boolean circuits and their sub-classes. We use $\mathcal{C}$ to denote any *typical* non-uniform circuit class, i.e., any class from the set $\{\mathtt{AC}_0, \mathtt{ACC}_0, \mathtt{TC}_0, \mathtt{NC}_1, \mathtt{NC}, \mathtt{P}/poly\}$. All these circuit classes are of polynomial size. We use $\mathcal{C}(s)$ to denote the class of $O(s)$-size $\mathcal{C}$ circuits. For truth-table tt, we use $ckt_{\mathcal{C}}(tt)$ to denote its exact $\mathcal{C}$ circuit complexity, i.e. the minimum size of any $\mathcal{C}$ circuit C whose truth-table (when concatenated to make the string $C(00\ldots0)\ldots C(11\ldots1)$) is tt. In the case of unrestricted Boolean circuits, instead of $\mathcal{C}(s)$ and $ckt_{\mathcal{C}}(tt)$, we use $\mathtt{SIZE}(s)$ and $ckt(tt)$ respectively.

Non-uniform classes: $L \in \Gamma/a$ if there exists an advice-taking Γ Turing Machine M, and advice sequence $\{a_n\}_{n\in\mathbb{N}}$ satisfying $\forall n\ |a_n| = a(n)$, such that: $x \in L \iff M(x)/a_{|x|} = 1$. For semantic classes, the machine M only needs to satisfy the semantic promise on the correct advice sequence $\{a_n\}_{n\in\mathbb{N}}$ (and not on all advice sequences). Below we define an exception for $\mathtt{C} = \mathtt{N}, \mathtt{R}, \mathtt{U}$ (points (1) and (3) also hold for $\Sigma_i\mathtt{TIME} \cap \Pi_i\mathtt{TIME}$):

(1) $L \in (\mathtt{CTIME}(t) \cap \mathtt{Co\text{-}CTIME}(t))/a$: If there are $\mathtt{NTIME}(t)$ Turing Machines M and M', and advice sequence $\{a_n\}_{n\in\mathbb{N}}$ satisfying $\forall n\ |a_n| = a(n)$, such that: (i) $x \in L \iff M(x)/a_{|x|} = 1$; (ii) both M and M' satisfy the semantic promise on $\{a_n\}_{n\in\mathbb{N}}$ (for $\mathtt{C} = \mathtt{N}$ there is no promise); and (iii) both accept complement languages. For the other advice sequences, M and M' are not required to satisfy the semantic promise, but are required to accept complement languages.

(2) $L \in \mathtt{ZCTIME}(t)/a$: It's the same as (1), except that in the "other advice sequences" part, M and M' are not required to accept complement languages. That is, both M and M' are required to accept complement languages just for some correct advice sequence, and simultaneously satisfy the semantic promise. Equivalently, there is a $\mathtt{ZCTIME}(t)$ Turing Machines N, and advice sequence $\{a_n\}_{n\in\mathbb{N}}$ satisfying $\forall n\ |a_n| = a(n)$, such that: $x \in L \iff N(x)/a_{|x|} = 1$; and N satisfy the semantic promise. For other advice sequences N might: (i) fail to provide uniqueness/largeness/existence; (ii) for some input, output both 0 and 1 (on different non-deterministic branches); or (iii) for some input, output just '?' (on all non-deterministic branches).

(3) $L \in \mathtt{CTIME}(t)/a \cap \mathtt{Co\text{-}CTIME}(t)/a$: It's further relaxed than (2). We need two advice sequences $\{a_n\}_{n\in\mathbb{N}}$ and $\{b_n\}_{n\in\mathbb{N}}$, satisfying $\forall n\ |b_n| = b(n)$ and $\forall n\ |b_n| = b(n)$ (these sequences need not be the same). M satisfy the semantic promise on $\{a_n\}_{n\in\mathbb{N}}$ and accept L. M' satisfy the semantic promise on $\{b_n\}_{n\in\mathbb{N}}$ and accept $\overline{L}$. There are no other conditions.

Remark: $L \in \mathtt{ZCTIME}(t)/a$ is equivalent to L having $\mathtt{CTIME}(t)/a$ and $\mathtt{Co\text{-}CTIME}(t)/a$ algorithms that both use the same advice. This shows:

$$\mathtt{CTIME}(t) \cap \mathtt{Co\text{-}CTIME}(t))/a \subseteq \mathtt{ZCTIME}(t)/a \subseteq \mathtt{CTIME}(t)/a \cap \mathtt{Co\text{-}CTIME}(t)/a \subseteq \mathtt{ZCTIME}(t)/2a$$

So the difference between $\mathtt{ZCTIME}(t)/a$ and $\mathtt{CTIME}(t)/a \cap \mathtt{Co\text{-}CTIME}(t)/a$ only matters when the amount of advice is precise.

Heuristic classes: For uniform/non-uniform class Λ, $L \in Heur\text{-}\Lambda$ if $\exists L' \in \Lambda$, such that for all polynomially samplable distributions $\mathcal{D}$, $\forall n\ Pr_{x\sim\mathcal{D},|x|=n}[L_n(x) = L'_n(x)] \geq 1 - \frac{1}{n}$.

Infinitely-often classes: For uniform/non-uniform, heuristic/non-heuristic class Λ, $L \in$ io-Λ if $\exists L' \in \Lambda$, and an infinite subset $S \subset \mathbb{N}$, such that $n \in S \implies L_n = L'_n$.

Variety of witness complexities for $\mathtt{CTIME}$**:** We define different ways of measuring complexity of witnesses for non-determinstic verifiers. We will later see that lower-bounds based on these measures are not always the same (see Table1).

(1) **Witnesses:** A non-deterministic verifier V for L, has witnesses in s-size C circuits, if for every $x \in L$, there is an $s(|x|)$-size C circuit C_x, such that $V(x, tt(C_x)) = 1$. If V uses a amount of advice, then we say that V/a has witnesses in s-size C circuits, if for some correct advice sequence $\{a_n\}_{n\in\mathbb{N}}$ satisfying $\forall n\ |a_n| = a(n)$, for every $x \in L$, there is an $s(|x|)$-size C circuit C_x, such that $V(x, tt(C_x))/a_{|x|} = 1$.

(2) **Hitting-sets for witnesses (all witnesses in one):** A non-deterministic verifier V for L has l-size hitting-sets in s-size C circuits, if $\forall n \in \mathbb{N}$, there is an $s(n)$-size C circuit C_n such that $tt(C_n)$ when partitioned into l strings $\{str_1, \ldots, str_l\}$ of equal lengths, satisfies $\forall(x : |x| = n \wedge x \in L)\ \exists(i \in [1, l])\ V(x, str_i) = 1$. The default value of l is 2^n. If V uses advice, hitting-sets are defined analogous to witnesses in the advice setting.

(3) **Oblivious witnesses (ordered hitting-sets for witnesses):** Let $y_1, \ldots, y_{2^n}$ denote the n-length strings arranged in the lexicographical order. A non-deterministic verifier V for L

has oblivious witnesses in s-size C circuits, if $\forall n \in \mathbb{N}$, there is an $s(n)$-size C circuit C_n such that $tt(C_n)$ when partitioned into 2^n strings $\{str_1, \ldots, str_{2^n}\}$ of equal lengths, satisfies $\forall (i \in [1, 2^n])\ y_i \in L \implies V(y_i, str_i) = 1$. For i with $y_i \notin L$, str_i is the all 0s string. If V uses advice, oblivious witnesses are defined analogous to the witnesses in advice setting.

We use $\texttt{CTIME}(t)/a \subset_w C(s)$ to say that, any $\texttt{CTIME}(t)$ verifier, for some correct advice sequence, has witnesses in $C(s)$ circuits. $\texttt{CTIME}(t)/a \not\subset_w C(s)$ means that, there is a $\texttt{CTIME}(t)$ verifier that: for any correct advice sequence, doesn't have witness in $C(s)$ circuits infinitely often. $\subset_{hw}$ ($\not\subset_{hw}$) and $\subset_{ow}$ ($\not\subset_{ow}$) are defined similarly for "hitting-sets for witnesses" and "oblivious witnesses" respectively.

Variety of seed complexities for ZCTIME**:** Any language in ZCTIME has two, a CTIME algorithm and a Co-CTIME algorithm deciding it. So instead of witnesses, we define a stronger notion: seeds, which is nothing but a technical way of combining witnesses from the two algorithms.

(1) **Seeds:** A zero-error non-deterministic verifier V for L has seeds in s-size C circuits, if for every x, there is an $s(|x|)$-size C circuit C_x, such that $V(x, tt(C_x)) \in \{0, 1\}$. If V uses a amount of advice, then we say that V/a has seeds in s-size C circuits, if for some correct advice sequence $\{a_n\}_{n\in\mathbb{N}}$ satisfying $\forall n\ |a_n| = a(n)$, for every x, there is an $s(|x|)$-size C circuit C_x, such that $V(x, tt(C_x))/a_{|x|} \in \{0, 1\}$.

(2) **Hitting-sets for seeds (all seeds in one):** A zero-error non-deterministic verifier V for L has l-size hitting-sets in s-size C circuits, if $\forall n \in \mathbb{N}$, there is an $s(n)$-size C circuit C_n such that $tt(C_n)$ when partitioned into l strings $\{str_1, \ldots, str_l\}$ of equal lengths, satisfies $\forall (x : |x| = n)\ \exists (i \in [1, l])\ V(x, str_i) \in \{0, 1\}$. The default value of l is 2^n. If V uses advice, hitting-sets are defined analogous to seeds in the advice setting.

(3) **Oblivious seeds (ordered hitting-sets for seeds):** Let $y_1, \ldots, y_{2^n}$ denote the n-length strings arranged in the lexicographical order. A zero-error non-deterministic verifier V for L has oblivious seeds in s-size C circuits, if $\forall n \in \mathbb{N}$, there is an $s(n)$-size C circuit C_n such that $tt(C_n)$ when partitioned into 2^n strings $\{str_1, \ldots, str_{2^n}\}$ of equal lengths, satisfies $\forall (i \in [1, 2^n])\ V(y_i, str_i) \in \{0, 1\}$. If V uses advice, oblivious seeds are defined analogous to seeds in the advice setting.

We use $\texttt{ZCTIME}(t)/a \subset_w C(s)$ to say that, any $\texttt{ZCTIME}(t)$ verifier, for some correct advice sequence, has seeds in s-size C circuits. $\texttt{ZCTIME}(t)/a \not\subset_w C(s)$ means that, there is a $\texttt{ZCTIME}(t)$ verifier that: for any correct advice sequence, doesn't have seeds in s-size C circuits infinitely often. $\subset_{hs}$ ($\not\subset_{hs}$) and $\subset_{os}$ ($\not\subset_{os}$) are defined similarly for "hitting-sets for seeds" and "oblivious seeds" respectively.

Useful properties: We define a generalized version of the natural properties.

Definition 2.1 (Useful uniform properties). A uniform Γ algorithm $\mathcal{A}$ is a Γ-C property if it satisfies the first condition stated below, on the inputs that are powers of 2 (interpreted as truth-tables of Boolean functions). $\mathcal{A}$ is said to be useful against s-size C circuits if it satisfies the second condition stated below.

(1) *Size restrictions:*
   (a) *Uniqueness for* $\mathsf{C} = \mathsf{U}$: $\forall n \in \mathbb{N}\ \Sigma_{x:|x|=2^n \wedge \mathcal{A}(x)=1} 1 = 1$
   (b) *Largeness for* $\mathsf{C} = \mathsf{R}$: $\forall n \in \mathbb{N}\ Pr_{x:|x|=2^n}[\mathcal{A}(x) = 1] \geq \frac{1}{2^n}$
   (c) *Existence for* $\mathsf{C} = \mathsf{N}$: $\forall n \in \mathbb{N}\ \Sigma_{x:|x|=2^n \wedge \mathcal{A}(x)=1} 1 \geq 1$

(2) *Usefulness:* for infinitely many $n \in \mathbb{N}$, $\forall (x : |x| = 2^n)\ \mathcal{A}(x) = 1 \implies ckt_C(x) > s(n)$

Note that, in the case where s is $poly(n)$, a single algorithm $\mathcal{A}$ should be useful against n^k-size C circuits for all k. That is, for each k, there should be infinitely many $n \in \mathbb{N}$, such that $\forall (x : |x| = 2^n)\ \mathcal{A}(x) = 1 \implies ckt_C(x) > n^k$.

Definition 2.2 (Useful properties that use advice). A Γ/a algorithm $\mathcal{A}$ is a Γ/a-C property if it satisfies the first condition of Definition 2.1 on an advice sequence $\{a_n\}_{n\in\mathbb{N}}$ that satisfies $\forall n\ |a_n| = a(n)$. $\mathcal{A}$ is said to be useful against s-size C circuits if it satisfies the second condition of Definition 2.1 on the same advice sequence $\{a_n\}_{n\in\mathbb{N}}$. For C = U, based on how $\mathcal{A}$ behaves on the advice sequences other than $\{a_n\}_{n\in\mathbb{N}}$, it has two special categories:

(1) Γ/a-**strong-unique or** Γ/a-$u_{=1}$**:** $\forall n \in \mathbb{N}\ \forall(b_n : |b_n| \leq a(n))\ \Sigma_{x:|x|=2^n \wedge \mathcal{A}(x)/b_n=1} 1 = 1$
(2) Γ/a-**mild-unique or** Γ/a-$u_{\leq 1}$**:** $\forall n \in \mathbb{N}\ \forall(b_n : |b_n| \leq a(n))\ \Sigma_{x:|x|=2^n \wedge \mathcal{A}(x)/b_n=1} 1 \leq 1$

Definition 2.3 (Promise useful properties). A Γ/a algorithm $\mathcal{A}$ is a Γ/a-prC property useful against s-size C circuits if there is an advice sequence $\{a_n\}_{n\in\mathbb{N}}$ satisfying $\forall n\ |a_n| = a(n)$ such that: for infinitely many $n \in \mathbb{N}$, $\mathcal{A}$ satisfies the first and second conditions of Definition 2.1. Informally, the only condition property satisfies is: it is simultaneously non-trivial/large/unique and useful for infinitely many input lengths, and no conditions for the other input lengths (property is even allowed to be empty).

For $\sqcap \in \{\mathsf{N}, \mathsf{R}, \mathsf{U}, u_{=1}, u_{\leq 1}, \mathsf{prN}, \mathsf{prR}, \mathsf{prU}\}$, we use Γ/a-$\sqcap \not\subset_{tt} C$ to say that there is a Γ/a-$\sqcap$ property useful against $C(s)$ circuits. In other words, there is a Γ/a algorithm, whose set of accepting inputs satisfies the size restrictions of $\sqcap$, and on infinitely many input lengths 2^n, algorithm accepts no truth-tables tt with $ckt_C(tt) \leq s(n)$.

For $\sqcap \in \{\mathsf{N}, \mathsf{R}, \mathsf{U}, u_{=1}, u_{\leq 1}\}$ we use Γ/a-$\sqcap \not\subset_{tt}$ io-C to denote properties that are useful on all but finitely many input lengths (i.e., there is a point after which the property is always useful).

Circuit lower bounds for complexity classes: The following three types of NEXP lower bounds are equivalent: (i) $\mathsf{NEXP} \not\subset \mathsf{SIZE}(poly)$, i.e., an $L \in \mathsf{NEXP}$ satisfies $\forall k\ L \notin \mathsf{SIZE}(n^k)$; (ii) $\mathsf{NE} \not\subset \mathsf{SIZE}(poly)$; (iii) $\forall k\ \mathsf{NE} \not\subset \mathsf{SIZE}(n^k)$, i.e., a fix slice of NEXP (in this case NE) has L_k for each k, such that $L_k \notin \mathsf{SIZE}(n^k)$. The equivalence of first two follows from a simple padding argument. Third is again implied by the first using a padding argument. The third implies the first two using a complete language for NE (under linear-time reductions).

The equivalence holds for the three analogous lower bounds for EXP, Σ_iEXP and Π_iEXP too. In fact, it holds for any exponential time/polynomial space syntactic class, but not for semantic classes. Due to the lack of any complete problem, the third lower bound is not known to imply the first two. Important examples of semantic classes are: ZNEXP, UEXP, ZUEXP, REXP, ZREXP, BPEXP and $\Sigma_i\mathsf{EXP} \cap \Pi_i\mathsf{EXP}$.

We say $\mathsf{NEXP} \not\subset$ io-$\mathsf{SIZE}(s)$ if there is an $L \in \mathsf{NEXP}$ that satisfies $L \notin$ io-$\mathsf{SIZE}(s)$: L doesn't have $\mathsf{SIZE}(s)$ circuits on all but finitely many input lengths (i.e., there is a point after which the language slices always have high circuit complexity). Similar lower bounds are defined for other classes.

BP **quantifier:** For any syntactic class Γ, we say L is in $\mathsf{BP}\cdot\Gamma$, if there is a polynomial p and a language $L' \in \Gamma$ such that,

$$x \in L \implies Pr_{r\in\{0,1\}^{p(n)}}[(x,r) \in L'] \geq 2/3$$
$$x \notin L \implies Pr_{r\in\{0,1\}^{p(n)}}[(x,r) \in L'] \leq 1/3$$

The R, Co-R, and ZP quantifiers are defined in similar fashion.

Promise problems: A promise problem $\Pi = (\Pi_Y, \Pi_N)$ is a pair of disjoint sets Π_Y and Π_N. In the special case where $\Pi_Y \cup \Pi_N = \{0,1\}^*$, Π is also a language. We say that a language L agrees with Π if:

$$x \in \Pi_Y \implies x \in L$$
$$x \in \Pi_N \implies x \notin L$$

Infinitely-often promise classes: For semantic class Λ, a promise problem L is in the class ip-Λ if: (i) the set of promise inputs include an infinite subset $S \subseteq \mathbb{N}$ of input lengths (contains the entire

n^{th}-slice for any $n \in S$, and nothing from the other slices); (ii) there exists a Turing machine M such that: for $n \in S$ and for n-length input x, M is a Λ-type machine on x, and $M(x) = 1 \iff L(x) = 1$.

Remark: An infinitely-often promise class is different from an infinitely-often class in the sense that: the Turing machine M should be of Λ-type for all input lengths. For example, io-MA is the class of languages that match any MA language for infinitely many input lengths. But, ip-MA is the class of languages that have an MA algorithm for infinitely many input lengths, and for all the other input lengths the algorithm is not required to satisfy the semantic promise of MA.

Special ip-classes: We say that $L \in$ ip-ZNE or $L \in$ ip-(NE $\cap$ Co-NE) if there is an infinite subset $S \subseteq \mathbb{N}$, $L_1 \in$ NE and $L_2 \in$ NE such that: for $n \in S$ and n-length input x, $L(x) = 1 \iff L_1(x) = 1 \iff L_2(x) = 0$. We define the classes ip-(Σ_iE $\cap$ Π_iE) and ip-(MA $\cap$ Co-MA) similarly (for the MA case, the MA protocols for L_1 and L_2 are not required to satisfy the semantic promise on inputs lengths not in the set S).

Upper and lower bounds for ip-classes: We say that ip-$\Lambda \subset C(s)$ if any $L \in$ ip-Λ, has $C(s)$ circuits for inputs lengths where the promise is met. We say that ip-$\Lambda \not\subset C(s)$ if there is an $L \in$ ip-Λ, that for infinite subset of promise input lengths (where the promise is met) does not have $C(s)$ circuits.

The upper and lower bounds for the ip-Λ seeds / hitting-sets for seeds / oblivious-seeds are defined in the similar fashion.

Oracle and advice for MA, AM and MAM protocols: MA^O/a (*resp.* AM^O/a, MAM^O/a) is the class of languages with *standard* Merlin-Arthur (*resp.* Arthur-Merlin, Merlin-Arthur-Merlin) protocols where Arthur has access to an oracle O and a amount of non-uniform advice in the final verification step. The protocols only need to satisfy the semantic promise on a correct advice sequence.

Generalization of AM: For $i \geq 0$, $\Pi = (\Pi_Y, \Pi_N)$ is in prAM_i (promise AM_i), if there is a polynomial p and a language $L \in \Sigma_i$, such that:

$$x \in \Pi_Y \implies Pr_{r \in \{0,1\}^{p(n)}}[(x,r) \in L] \geq 2/3$$
$$x \in \Pi_N \implies Pr_{r \in \{0,1\}^{p(n)}}[(x,r) \in L] \leq 1/3$$

AM_i consists of problems in prAM_i that are languages. We use $\mathsf{AM}_i\mathsf{TIME}$ to denote the corresponding complexity measure. $\mathsf{AM}_i = \mathsf{BP} \cdot \Sigma_i$, and from [38, 77] we know that $\mathsf{AM}_i = \mathsf{AM}^{\Sigma_{i-1}} = \mathsf{BP} \cdot \mathsf{NP}^{\Sigma_{i-1}} = \mathsf{Co\text{-}R} \cdot \mathsf{NP}^{\Sigma_{i-1}} \subseteq \mathsf{Co\text{-}RP}^{\Sigma_i} \subseteq \mathsf{BPP}^{\Sigma_i}$. Two special cases are: $\mathsf{AM}_0 = \mathsf{BPP}$ and $\mathsf{AM}_1 = \mathsf{AM}$.

Generalization of MA: For $i \geq 1$, $\Pi = (\Pi_Y, \Pi_N)$ is in prMA_i (promise MA_i), if there is a $\Psi = (\Psi_Y, \Psi_N)$ in prAM_{i-1}, a $\Gamma \in \Pi_{i-1}$, and a polynomial p, such that:

$$x \in \Pi_Y \implies \exists (y \in \{0,1\}^{p(n)})[(x,y) \in \psi_Y \wedge (x,y) \in \Gamma]$$
$$x \in \Pi_N \implies \forall (y \in \{0,1\}^{p(n)})[(x,y) \in \psi_N \vee (x,y) \notin \Gamma]$$

MA_i consists of problems in prMA_i that are languages. We use $\mathsf{MA}_i\mathsf{TIME}$ to denote the corresponding complexity measure. Borrowing notation from [7] we define $\mathsf{MA}_i = \mathsf{M}(\mathsf{AM}_{i-1}||\Pi_{i-1})$, where the symbol "||" stands for the "logical and" operation. One special case is: $\mathsf{MA}_1 = \mathsf{MA}$. Also, $\mathsf{MA}_i \subseteq \mathsf{MA}^{\Sigma_{i-1}}$ since $\mathsf{AM}_{i-1} \subseteq \mathsf{BPP}^{\Sigma_{i-1}}$.

Remark: Any language in MA_i has a protocol where the all-powerful prover Merlin sends a non-deterministic proof to two verifiers: (i) the usual randomized verifier Arthur (which again interacts with Merlin), and (ii) a Π_{i-1} verifier Henry. The two verifiers can't communicate with each other. In the advice setting, only Arthur uses the advice in the final verification step.

Important inclusions: For $i \geq 1$, the following inclusions follow from the definitions (and from the fact noted in [38] that the proofs of $\mathsf{MA} \subseteq \mathsf{S}_2 \cdot \mathsf{P} = \mathsf{P}^{\mathsf{S}_2 \cdot \mathsf{P}} \subseteq \mathsf{ZPP}^{\mathsf{NP}}$ [6, 16, 28, 62] and $\mathsf{MA} \subseteq \mathsf{MAM} = \mathsf{AM}$ [8, 10] relativizes):

$$(1)\quad \begin{array}{ccccccc} & \Sigma_i & & & & \Pi_{i+1} & \\ & \nwarrow\ \nearrow & & & & \nwarrow\ \nearrow & \\ \mathsf{AM}_{i-1} & & \mathsf{MA}_i \subseteq \mathsf{MA}^{\Sigma_{i-1}} \subseteq \mathsf{AM}^{\Sigma_{i-1}} = \mathsf{AM}_i & & & & \text{and } \Sigma_i/poly = \mathsf{AM}_i/poly \\ & & & & & \nwarrow\ \nearrow & \\ & & & & & \mathsf{MA}_{i+1} & \end{array}$$

$$(2)\quad \begin{array}{ccccccc} & \mathsf{P}^{\Sigma_i}_{||} & \subseteq & \mathsf{P}^{\Sigma_i} & & & \\ \Sigma_i \nearrow & & & & \nwarrow & & \\ \nwarrow & & & & \nearrow & \mathsf{S}_2 \cdot \mathsf{P}^{\Sigma_{i-1}} = \mathsf{P}^{\mathsf{S}_2 \cdot \mathsf{P}^{\Sigma_{i-1}}} \subseteq \mathsf{ZPP}^{\Sigma_i} \subseteq \Sigma_{i+1} \cap \Pi_{i+1} \subseteq \Sigma_{i+1} & \\ & \mathsf{MA}_i & \subseteq & \mathsf{MA}^{\Sigma_{i-1}} & & & \end{array}$$

$$(3)\quad \begin{array}{ccccc} & \Sigma_i/poly & & & \\ (\Sigma_i \cap \Pi_i)/poly \subseteq \Sigma_i/poly \cap \Pi_i/poly \nearrow & & & \nwarrow & \mathsf{P}^{\Sigma_i}_{||}/poly \subseteq \mathsf{P}^{\Sigma_i}/poly \\ \nwarrow & & & \nearrow & \\ & \Pi_i/poly & & & \end{array}$$

The subscript "||" in $\mathsf{P}^{\Sigma_i}_{||}$ implies that the oracle queries are "non-adaptive" (or "parallel").

General Circuit Classes: For any $i \geq 1$ we generalize the definitions of the circuits classes from [64] in the most natural way, with an addition of promise single-valued non-deterministic/Σ_i circuits:

(1) **Σ_i-oracle Circuits:** A Σ_i-oracle circuit is a Boolean circuit C that also has special gates: the $\Sigma_i\mathsf{SAT}$-oracle gates. A $\Sigma_i\mathsf{SAT}$-oracle gate outputs 1 if and only if its input is a satisfying instance of $\Sigma_i\mathsf{SAT}$. We say $L \in \mathsf{SIZE}^{\Sigma_i}(s)$, if for $n \in \mathbb{N}$, L_n has $O(s(n))$-size Σ_i-oracle circuit deciding it. The equation $\mathsf{SIZE}^{\Sigma_i}(poly) = \mathsf{P}^{\Sigma_i}/poly$ follows from the definitions.

(2) **Non-adaptive Σ_i-oracle Circuits:** A non-adaptive Σ_i-oracle circuit is a pair of Boolean circuits (C_{pre}, C_{post}). On n-length input x, C_{pre} outputs a number of queries: $q_1, q_2, \ldots, q_m$. C_{post} receives $m+1$ inputs: x, and bits $a_1, a_2, \ldots, a_m$ where $\forall i\ a_i = 1 \iff q_i \in \Sigma_i\mathsf{SAT}$. C_{post} outputs the final answer in a single bit. We say $L \in \mathsf{SIZE}^{\Sigma_i}_{||}(s)$, if for $n \in \mathbb{N}$, L_n has $O(s(n))$-size non-adaptive Σ_i-oracle circuit deciding it. The equation $\mathsf{SIZE}^{\Sigma_i}_{||}(poly) = \mathsf{P}^{\Sigma_i}_{||}/poly$ follows from the definitions.

(3) **Σ_i (*resp.* Π_i) Circuits:** A Σ_i (*resp.* Π_i) circuit is a Boolean circuit C that receives $n+i$ inputs: x of length n, and $y_1, \ldots, y_i$. The function $f_C : \{0,1\}^n \to \{0,1\}$ computed by C satisfies: $f_C(x) = 1 \iff \exists y_1 \forall y_2 \ldots C(x, y_1, \ldots, y_i) = 1$(*resp.* $\forall y_1 \exists y_2 \ldots C(x, y_1, \ldots, y_i) = 0$). We say $L \in \Sigma_i\mathsf{SIZE}(s)$ (*resp.* $L \in \Pi_i\mathsf{SIZE}(s)$), if for $n \in \mathbb{N}$, L_n has $O(s(n))$-size Σ_i (*resp.* Π_i) circuit deciding it. The equation $\Sigma_i\mathsf{SIZE}(poly) = \Sigma_i/poly$ (*resp.* $\Pi_i\mathsf{SIZE}(poly) = \Pi_i/poly$) follows from the definitions.

(4) **Single-Valued Σ_i or $\mathsf{SV}\Sigma_i$ Circuits:** An $\mathsf{SV}\Sigma_i$ circuit is a Boolean circuit C that receives $n+i$ inputs: x of length n, and $y_1, \ldots, y_i$. It has two output gates: $value_C$ and $flag_C$. Let $Value_C(x, y_1) = \forall y_2 \exists y_3 \ldots value_C(x, y_1, \ldots, y_i)$ and $Flag_C(x, y_1) = \forall y_2 \exists y_3 \ldots flag_C(x, y_1, \ldots, y_i)$. The circuit C computes function $f_C : \{0,1\}^n \to \{0,1\}$ if it satisfies the following two promises for any input x: (a) $\exists y_1\ Flag_C(x, y_1) = 1$; (b) $\forall y_1\ Flag_C(x, y_1) = 1 \implies Value_C(x, y_1) = f_C(x)$. We say $L \in \mathsf{SV}\Sigma_i\mathsf{SIZE}(s)$, if for $n \in \mathbb{N}$, L_n has $O(s(n))$ size $\mathsf{SV}\Sigma_i$ circuit deciding it. Note that, a function has an $O(s)$-size $\mathsf{SV}\Sigma_i$ circuit if and only if it has both, an $O(s)$-size Σ_i circuit and an $O(s)$-size Π_i circuit. The equation $\mathsf{SV}\Sigma_i\mathsf{SIZE}(poly) = \Sigma_i/poly \cap \Pi_i/poly$ follows from the definitions.

(5) **Promise $\mathsf{SV}\Sigma_i$ or $\mathsf{prSV}\Sigma_i$ Circuits / Promise SV_i or prSV_i Algorithms:** A linear-time algorithm $\mathcal{A}$ is called prSV_i if: on each input it outputs an $\mathsf{SV}\Sigma_i$ circuit (that satisfies the two promises for some n'-bit function $f_{n'}$ for $n' \leq n$). We say $L \in \mathsf{prSV}^{\mathcal{A}}\Sigma_i\mathsf{SIZE}(s)$, if for $n \in \mathbb{N}$, L_n has $O(s(n))$ size $\mathsf{SV}\Sigma_i$ circuit C_n deciding it. Additionally, the circuit sequence $\{C_n\}_{n\in\mathbb{N}}$ is produced by the prSV_i algorithm $\mathcal{A}$ on some input sequence $\{x_{s(n)}\}_{n\in\mathbb{N}}$ (where x_j is of size j). $L \in \mathsf{prSV}\Sigma_i\mathsf{SIZE}(s)$ means that $L \in \mathsf{prSV}^{\mathcal{A}}\Sigma_i\mathsf{SIZE}(s)$ for some prSV_i algorithm $\mathcal{A}$. The equation $\mathsf{prSV}\Sigma_i\mathsf{SIZE}(poly) = (\Sigma_i \cap \Pi_i)/poly$ follows from the definitions.

Why do we need promise algorithms to produce $\mathtt{prSV}\Sigma_i$ circuits? The purpose of $\mathtt{prSV}\Sigma_i$ circuits is to capture the class $(\Sigma_i \cap \Pi_i)/poly$. This class is weaker than the class $\Sigma_i/poly \cap \Pi_i/poly$: the class of all non-uniform $\mathtt{SV}\Sigma_i$ circuits. $(\Sigma_i \cap \Pi_i)/poly$ only contains $\mathtt{SV}\Sigma_i$ circuit sequences that are produced by: some pair of Σ_i and Π_i algorithms that use non-uniform advice, and are complement on each input and advice sequence. So to capture this restricted class of $\mathtt{SV}\Sigma_i$ circuits, we deal with each pair of such Σ_i and Π_i algorithms. A $\mathtt{prSV}_i$ algorithm is nothing but a pair of such algorithms.

For example: the class $(\mathtt{NTIME}(n^{k/2}) \cap \mathtt{Co\text{-}NTIME}(n^{k/2}))/n^k$ has $\mathtt{prSV}_i\mathtt{SIZE}(n^k)$ circuits, and any language in $\mathtt{prSV}_i\mathtt{SIZE}(n^k)$ has an $(\mathtt{NTIME}(n^k) \cap \mathtt{Co\text{-}NTIME}(n^k))/n^k$ algorithm. Here the n^k-size advice is the input to the underlying $\mathtt{prSV}_i$ algorithm. The algorithm combines the deterministic verifiers of the $\mathtt{NTIME}(n^{k/2})$ and $\mathtt{Co\text{-}NTIME}(n^{k/2})$ predicates, and produces a circuit that captures the $(\mathtt{NTIME}(n^{k/2}) \cap \mathtt{Co\text{-}NTIME}(n^{k/2}))$-computation on that advice.

Another point of view is: to look at $(\Sigma_i \cap \Pi_i)/poly$ as $\mathtt{P}^{\Sigma_i \cap \Pi_i}/poly$. In this case, instead of dealing with each $\mathtt{prSV}_i$ algorithm, one can deal with each $(\Sigma_i \cap \Pi_i)$-oracle. It is again different from the case $\mathtt{P}^{\Sigma_i}/poly$ because all of the Σ_i oracles can be replaced by just one oracle, the $\Sigma_i\mathtt{SAT}$ oracle. In case of $(\Sigma_i \cap \Pi_i)$-oracles we don't have this luxury due to the lack of complete problems.

Lower bounds and properties against $\mathtt{prSV}\Sigma_i$ circuits: All notations for expressing lower bounds and properties against a typical circuit class C extend to a general circuit class Λ in the most natural way. We extend the measures $ckt_C(tt)$ and $ckt(tt)$ to define: $ckt_{\Sigma_i}(tt)$ / $ckt_{\Pi_i}(tt)$ / $ckt^M(tt)$ / $ckt^M_{||}(tt)$ as the minimum size of any Σ_i / Π_i / M-oracle circuit / non-adaptive M-oracle circuit, whose truth-table is tt. For the special oracle of $\Sigma_i\mathtt{SAT}$, we use $ckt^{\Sigma_i}(tt)$ and $ckt^{\Sigma_i}_{||}(tt)$.

The promise $\mathtt{SV}\Sigma_i$ circuits need special care. This is because we need to deal with each $\mathtt{prSV}_i$ algorithm separately. For any class Γ, we define the following upper bounds and lower bounds against $\mathtt{prSV}\Sigma_i$ circuits:

- Let $\mathcal{A}$ be a $\mathtt{prSV}_i$ algorithm. For any 2^n-length truth-table tt, we use $ckt_{\mathtt{SV}_i(\mathcal{A})}(tt)$ to denote the minimum size $s(n)$ such that: $\mathcal{A}$ outputs C with $tt(C) = tt$ on an $s(n)$-size input. We use $ckt_{\mathtt{prSV}_i}(tt)$ to denote the minimum size $s(n)$ such that: any $\mathtt{prSV}_i$ algorithm with description length at most $\log n$, outputs C with $tt(C) = tt$ on an $s(n)$-size input.
- $\Gamma \subset \mathtt{prSV}\Sigma_i(s)$: For $L \in \Gamma$ there is a $\mathtt{prSV}_i$ algorithm $\mathcal{A}$, an input sequence $\{x_{s(n)}\}_{n\in\mathbb{N}}$, such that $\forall n \in \mathbb{N}\ tt(\mathcal{A}(x_{s(n)})) = L_n$.
- $\Gamma \not\subset \mathtt{prSV}\Sigma_i\mathtt{SIZE}(s)$: There is an $L \in \Gamma$, such that for every $\mathtt{prSV}_i$ algorithm $\mathcal{A}$, there is an infinite subset $S \subset \mathbb{N}$ of input lengths, such that $n \in S \implies \forall x : |x| = s(n)\ tt(\mathcal{A}(x)) \neq L_n$.
- $\Gamma \not\subset \text{w-}\mathtt{prSV}\Sigma_i\mathtt{SIZE}(s)$: For every $\mathtt{prSV}_i$ algorithm $\mathcal{A}$, there is an $L^{\mathcal{A}} \in \Gamma$, there is an infinite subset $S \subset \mathbb{N}$ of input lengths, such that $n \in S \implies \forall x : |x| = s(n)\ tt(\mathcal{A}(x)) \neq L^{\mathcal{A}}_n$. The prefix 'w' in the circuit class stands for 'weaker'. The lower bound is weaker in the sense that, $\mathtt{SV}\Sigma_i$ circuits produced by two different $\mathtt{prSV}_i$ algorithms, are allowed to beaten by two different Γ languages (infinitely-often).
- $\Gamma \subset_{ow/os/hw/hs/w/s} \mathtt{prSV}\Sigma_i\mathtt{SIZE}(s)$: For $L \in \Gamma$ and Γ verifier V for L, there is a $\mathtt{prSV}_i$ algorithm $\mathcal{A}$, there is an input sequence $\{x_{s(n)}\}_{n\in\mathbb{N}}$ such that $\forall n \in \mathbb{N}\ tt(\mathcal{A}(x_{s(n)}))$ is the 'oblivious witness / oblivious seed / hitting-set for witnesses / hitting-set for seeds' for V on n-length inputs. For the case of 'witnesses / seeds', $\forall n \in \mathbb{N}\ \forall y : |y| = n$ there is an input $x_{s(n)}$ such that $tt(\mathcal{A}(x_{s(n)})\}$ is the 'witness / seed' for V on input y.
- $\Gamma \not\subset_{ow/os/hw/hs} \mathtt{prSV}\Sigma_i\mathtt{SIZE}(s)$: There is an $L \in \Gamma$ and Γ verifier V for L, such that for every $\mathtt{prSV}_i$ algorithm $\mathcal{A}$, there is an infinite subset $S \subset \mathbb{N}$ of input lengths, such that $n \in S \implies \forall x : |x| = s(n)\ tt(\mathcal{A}(x))$ is not the 'oblivious witness / oblivious seed / hitting-set for witnesses / hitting-set for seeds' for V on n-length inputs. For the case of 'witnesses /

seeds', $n \in S \implies \exists y : |y| = n \; \forall x : |x| = s(n)\ tt(\mathcal{A}(x))$ is not the ' witness / seed' for V on input y.

- Γ-N/R/U $\not\subset_{tt}$ prSVΣ_iSIZE(s): A Γ algorithm $\mathcal{B}$ is called Γ-N/R/U property, if it satisfies the size restrictions of Definition 2.1. $\mathcal{B}$ is useful against prSV$\Sigma_i(s)$ circuits if for any prSV$_i$ algorithm $\mathcal{A}$, there is an infinite subset $S \subset \mathbb{N}$ of input lengths, such that $2^n \in S \implies \forall x : |x| = s(n)\ \mathcal{B}(tt(\mathcal{A}(x))) = 0$.

Hardness vs randomness: The process of using a function that is hard for a circuit class Λ (i.e. requires large size of Λ circuits) to yield pseudo random generator (PRG) that fools Λ circuits (i.e. creates a sparse subset of inputs with roughly same fraction of inputs resulting in 1) is well known in the literature. A PRG G creates this sparse subset by mapping a small input length to the required larger output length (same as the input length of the circuit).

A PRG $G : s(n) \to n$ is computable in Γ means: the language $L_G = \{(s, i, b) \mid \text{the } i^{th}\text{-bit of } G(s) \text{ is } b\}$ is in Γ. Inputs to G are called seeds, and their size (here $s(n)$) is called the seed length of G.

A PRG G is fooling a circuit C means: the fraction of inputs from the $2^{s(n)}$ size image of G that C accepts, is same as the fraction of all the inputs that C accepts (with error $\pm 1/n$). We use the following theorem in all our derandomization results.

Theorem 2.4. *[46, 56, 64, 69] There exists a universal constant g such that the following holds for any class O of oracles and oracle M, and any constants $\epsilon > 0$ and $d \geq 1$: if a Boolean function family $f = \{f_n\}_{n\in\mathbb{N}}$ computable in E^O that satisfies $\forall n \in \mathbb{N}\ ckt^M(f(n)) \geq n^{gd/\epsilon}$, then there exists a PRG family $G = \{G_n\}_{n\in\mathbb{N}}$ computable in E^O, such that $G_n : n^\epsilon \to n^d$ fools n^d-size B-oracle circuits. Moreover, any subset of the following is true (simultaneously):*

- *if $f \in \mathsf{E}^O_{||}$, then G is computable in $\mathsf{E}^O_{||}$.*
- *if $ckt^M_{||}(f(n)) \geq n^{gd/\epsilon}$, then G is secure against non-adaptive M-oracle circuits.*
- *if circuit lower bound holds infinitely often, then G fools circuits infinitely often.*

3 EWL AND KLT FOR UTIME AND RELATED CLASSES

We first give a specific search to decision reduction for UTIME (Section 3.1). Using this reduction we give the EWL and KLT for UTIME (Section 3.2). Then we describe similar results for ZUTIME (Section 3.3) and FewTIME (Section 3.4).

3.1 Search to decision reduction for UTIME

For $L \in \mathsf{NP}$ and verifier V for L, there is a standard P^{NP} algorithm for the corresponding search problem. This algorithm implicitly decides the following language:

$$L_{ewl(V)} = \{(x, i) \mid \exists y\ [V(x, y) = 1 \wedge (i^{th}\text{-bit of } y \text{ is } 1) \wedge \forall (z <_{l.o.}\ y)\ V(x, z) = 0]\} \tag{6}$$

where $z <_{l.o.}\ y$ stands for "z is lecigraphically smaller than y", and the subscript $ewl(V)$ in $L_{ewl(V)}$ stands for "easy-witness language for V".

So if $\mathsf{P} = \mathsf{NP}$, then $L_{ewl(V)} \in \mathsf{P}$. For $L \in \mathsf{NEXP}$ and verifier V for L, such results are not known. In particular, it is not known whether $\mathsf{NEXP} = \mathsf{EXP}$ yields an EXP algorithm for the corresponding search problem, let alone $L_{ewl(V)}$.

$\mathsf{NEXP} \subset_w \mathsf{SIZE}(poly)$ yields EXP algorithms for the NEXP search problems, by a simple brute-force argument. It is known that $\mathsf{NEXP} \subset_w \mathsf{SIZE}(poly)$ is equivalent to $\mathsf{NEXP} \subset \mathsf{SIZE}(poly)$ [31, 75], and to $\mathsf{NEXP} = \mathsf{MA}$ [31] (reverse implication was attributed to van Melkebeek). In [31] it was also shown that a weaker collapse, namely $\mathsf{NEXP} = \mathsf{AM}$, is sufficient to give EXP algorithms for NEXP search problems. This is the weakest collapse known so far.

From [34] we get: $\forall(L \in \mathsf{NEXP})\ \forall$ (NEXP verifier V for L) $L_{ewl(V)} \in \mathsf{EXP} \iff \mathsf{EXP}^{\mathsf{NP}} = \mathsf{EXP}$.

In this section we show that for $L \in \mathtt{UTIME}(t)$ and unambiguous verifier V for L, $L_{ewl(V)} \in \mathtt{UTIME}(t)$. $\mathtt{UTIME}(t)$ languages also have $O(t)$-time verifiers that are ambiguous. We show why it would be difficult to extend this result to all $O(t)$-time ambiguous verifiers for $\mathtt{UTIME}(t)$ languages.

Theorem 3.1. *For $L \in \mathtt{UTIME}(t)$ and unambiguous verifier V for L, $L_{ewl(V)} \in \mathtt{UTIME}(t(n))$ (where n is the input size for L and not $L_{ewl(V)}$). Moreover if this statement is true for every $O(t)$-time non-deterministic verifier (ambiguous and unambiguous) for every $\mathtt{UTIME}(t)$ language, then $\mathtt{ZNTIME}(t) = \mathtt{ZUTIME}(t)$.*

Proof. *Algorithm for $L_{ewl(V)}$:* For input (x, i), guess a certificate y and simulate $V(x, y)$. Accept if V accepts and the i^{th}-bit of y is 1, otherwise reject. This algorithm is correct and unambiguous as V is unambiguous. It runs in time $O(t(|x|))$.

The moreover part: For $L \in \mathtt{ZNTIME}(t)$, let V_1 and V_0 be its $\mathtt{NTIME}(t)$ and $\mathtt{Co\text{-}NTIME}(t)$ verifiers respectively. Consider the $\mathtt{UTIME}(t)$ language $L' = \{0,1\}^*$.

Using V_1 and V_0 we first construct a verifier V' for L': if the first bit of the certificate is i, V' simulates V_i using the rest of the certificate.

Now using a $\mathtt{UTIME}(t(n))$ algorithm $\mathcal{A}$ for $L'_{ewl(V')}$ we give a $\mathtt{UTIME}(t)$ algorithm for L: on input x, simulate $\mathcal{A}$ on $(x, 1)$. Accept iff $\mathcal{A}$ accepts.

If $\mathcal{A}$ accepts then we know that $x \in L$ because there is no positive certificate for V' that starts with 0 (or in other words, no positive certificate for V_0). If $\mathcal{A}$ rejects, then $x \in L$ because there is a positive certificate for V' that starts with 0 (or in other words, a positive certificate for V_0).

Similarly, there is a $\mathtt{UTIME}(t)$ algorithm for $\overline{L}$, and thus $L \in \mathtt{ZUTIME}(t)$. □

For the advice setting same proof goes through for the following adaptation of $L_{ewl(V)}$. For non-deterministic verifier V/a, that uses a amount of advice to decide a language L, for any correct advice sequence $\{a_n\}_{n\in\mathbb{N}}$:

$$L_{ewl(V/a)} = \{(x,i) \mid \exists y\ [V(x,y)/a_{|x|} = 1 \wedge (i^{th}\text{-bit of } y \text{ is } 1) \wedge \forall (z <_{l.o.} y)\ V(x,z)/a_{|x|} = 0]\} \quad (7)$$

Using this adaptation we get the following stronger corollary.

Corollary 3.2. *For $L \in \mathtt{UTIME}(t)/a$ and unambiguous verifier V/a for L, $L_{ewl(V/a)} \in \mathtt{UTIME}(t)/a$.*

3.2 EWL and KLT for UTIME

Using the search to decision reduction from Theorem 3.1 we derive EWL for unambiguous verifiers of languages in $\mathtt{UTIME}(t)$. Here again we see why it might be difficult to extend this to all ambiguous verifiers. Using the EWL we also get a KLT for UTIME.

Theorem 3.3. *For time-constructible $t \in 2^{O(n)}$, and constants c and k:*

1. $\mathtt{UTIME}(t) \subset C(n^k) \implies \mathtt{UTIME}(t) \subset_{ow} C(n^k)$. *Moreover if this statement is true for every $O(t)$-time non-deterministic verifier (ambiguous and unambiguous) for every $\mathtt{UTIME}(t)$ language, even just for witnesses (let alone oblivious-witnesses), then $\mathtt{ZNTIME}(t) \subseteq \mathtt{DTIME}(2^{n^{k+1}})$.*
2. $\mathtt{UTIME}(t)/a \subseteq C(n^k) \implies \mathtt{UTIME}(t)/a \subset_{ow} C(n^k)$.
3. $\mathtt{UTIME}(2^{n^c})/a \subseteq C(n^k) \implies \mathtt{UTIME}(2^{n^c})/a \subset_{ow} C(n^{ck})$.
4. $\mathtt{UEXP}/a \subseteq \mathtt{SIZE}(poly) \implies \mathtt{UEXP}/a = \mathtt{MA}/a$.

Proof. *Proof of (1):* For $L \in \mathtt{UTIME}(t)$, let $x \in L$ be an n-length input, and V be an unambiguous verifier for L whose certificate length is $\leq d \cdot t$ for some constant d. The $\mathtt{UTIME}(t)$ algorithm for $L_{ewl(V)}$ from Theorem 3.1 puts it into $C(m^k)$ for input size m. The C circuit for input length $m = (|x| + \log t + \log d) \in O(n)$ is the oblivious witness circuit for n-length inputs.

The moreover part: For $L \in \mathtt{ZNTIME}(t)$, construct the same verifier V' for the language $L' = \{0,1\}^*$ as in the proof of Theorem 3.1. As $L' \in \mathtt{UTIME}(t)$, V' will have witness in $C(n^k)$. Now a $\mathtt{DTIME}(2^{n^{k+1}})$

algorithm for L is: for n-length input x, go through all the circuits in $C(n^k \log n)$ one at a time, compute their truth-tables tt, and then compute $V'(x, tt)$. Due to the way V' is constructed, all of its positive certificates have the same first bit. If V accepts on any tt whose first bit is 1, then $x \in L$. Else $x \notin L$.

Proofs of (2) & (3): They are analogous to the proof of (1), except that they use Corollary 3.2.

Proof of (4): Let $L \in \mathtt{UEXP}/a$, and V/a be an unambiguous (given the correct advice) verifier V for L that runs in time $O(2^{n^c})$ for some constant c. Since $\exists k\ L \in \mathtt{SIZE}(n^k)$, from the proof of part (3) we know that V/a has witnesses in $\mathtt{SIZE}(n^{ck})$ for some constant k.

Using this we first give an $\mathtt{EXP}/a$ algorithm for L. On n-length input x, go through all the circuits in $\mathtt{SIZE}(n^{ck} \log n)$ one at a time, compute their truth-tables tt, and then compute $V(x, tt)/a$. Accept if V/a accepts for any tt, else reject. This is an $\mathtt{EXP}/a$ algorithm as simulation of V/a needs the original advice.

Once we get $\mathtt{UEXP}/a = \mathtt{EXP}/a$, $\mathtt{EXP}/a \subseteq \mathtt{SIZE}(poly)$ gives $\mathtt{UEXP}/a = \mathtt{MA}/a$ [43]. □

3.3 EWL and KLT for ZUTIME

We extend the techniques from the previous section to give similar results for ZUTIME. The main difference in the proof of our search to decision reduction is that, we adapt our definition of $L_{ewl(V)}$ to capture seeds of zero-error non-deterministic verifiers. First let's define this adaptation. For zero-error non-deterministic verifier V for language L:

$$L_{ewl(V)} = \{(x, i) \mid \exists y\ [V(x, y) \in \{0, 1\} \wedge (i^{th}\text{-bit of } y \text{ is } 1) \wedge \forall (z <_{l.o.} y)\ V(x, z) = ?]\} \quad (8)$$

The difference is that $L_{ewl(V)}$ captures the lexicographically first certificate that gives the correct answer (doesn't matter whether the answer is 1 or 0). Once the search to decision reduction is established, the EWL and KLT follow from similar arguments as in the previous section. Here again we see why it might be difficult to extend these results to all ambiguous zero-error verifiers.

Theorem 3.4. *For time-constructible $t \in 2^{O(n)}$, and constants c and k:*

(1) *For $L \in \mathtt{ZUTIME}(t)$ and zero-error unambiguous verifier V for L, $L_{ewl(V)} \in \mathtt{ZUTIME}(t(n))$ (where n is the input size for L). Moreover if this statement is true for all $O(t)$-time zero-error non-deterministic verifiers (ambiguous and unambiguous) for every $\mathtt{ZUTIME}(t)$ language, then $\mathtt{ZNTIME}(t) = \mathtt{ZUTIME}(t)$.*

(2) *$\mathtt{ZUTIME}(t) \subset C(n^k) \implies \mathtt{ZUTIME}(t) \subset_{os} C(n^k)$. Moreover if this statement is true for all $O(t)$-time zero-error non-deterministic verifiers (ambiguous and unambiguous) for every $\mathtt{ZUTIME}(t)$ language, even just for seeds (let alone oblivious-seeds), then $\mathtt{ZNTIME}(t) \subseteq \mathtt{DTIME}(2^{n^{k+1}})$.*

(3) *$\mathtt{ZUTIME}(2^{n^c}) \subset C(n^k) \implies \mathtt{ZUTIME}(2^{n^c}) \subset_{os} C(n^{ck})$.*

(4) *$\mathtt{ZUEXP} \subset \mathtt{SIZE}(poly) \implies \mathtt{ZUEXP} = \mathtt{MA}$.*

Proof. *Proof of (1): Algorithm for $L_{ewl(V)}$:* For input (x, i), guess a certificate y and simulate $V(x, y)$. Output '?' is V outputs '?'. Output 1 if V outputs in $\{0, 1\}$ and the i^{th}-bit of y is 1. Output 0 if V outputs in $\{0, 1\}$ and the i^{th}-bit of y is 0. This algorithm is correct and zero-error unambiguous as V is zero-error unambiguous. It runs in time $O(t(|x|))$.

The moreover part: For $L \in \mathtt{ZNTIME}(t)$, let V be its $\mathtt{ZNTIME}(t)$ verifier. Consider the $\mathtt{ZUTIME}(t)$ language $L' = \{0, 1\}^*$.

Using V we first construct a $\mathtt{ZNTIME}(t)$ verifier V' for L': V' ignores the first bit of the certificate and simulates V using the rest of the certificate. V' outputs '?' if V outputs '?', it outputs 1 if V outputs in $\{0, 1\}$ and its output matches the first bit of the certificate.

Now using a $\mathtt{ZUTIME}(t(n))$ algorithm $\mathcal{A}$ for $L'_{ewl(V')}$ we give a $\mathtt{ZUTIME}(t)$ algorithm for L: on input x, simulate $\mathcal{A}$ on $(x, 1)$. Output whatever $\mathcal{A}$ outputs.

Proofs of (2), (3) & (4): They are analogous to the proofs in Theorem 3.3. □

3.4 EWL and KLT for FewTIME

One other well studied variant of $\mathtt{UTIME}(t)$ is $\mathtt{FewTIME}(t)$, which was first defined in [2]. $L \in \mathtt{FewTIME}(t)$, if there exists a constant c and a non-deterministic verifier V, such that the number of accepting certificates on any input is bounded by t^c. The search to decision reduction of UTIME doesn't work here, because we don't know the exact number of accepting certificates (and only know an upper bound). We get rid of this problem by: either (i) assuming NE = ZUE (Theorem 3.5); or (ii) using advice that encodes the total number of accepting certificates for all the inputs of that length (Theorem 3.6). After the search to decision reduction is obtained, arguments for the EWL and the KLT are similar to the ones used for UTIME.

Note that, in the following theorems, the collapse assumptions/results don't directly give the 'S-to-D' (search to decision) reduction.

For instance: NE = ZUE only says that every FewE language has a ZUE verifier, but it doesn't say anything about the complexity of the language $L_{ewl(V)}$ (from the Equation (6)) for FewE verifiers V. This is analogous to the case: NE = E doesn't show $L_{ewl(V)} \in \mathtt{E}$ for NE verifiers V (we need the stronger collapse $\mathtt{E}^{\mathtt{NP}} = \mathtt{E}$ [34]).

Theorem 3.5. *The following statements are true if* NE = ZUE :

(1) S-to-D : $L \in$ FewE $\implies \forall$(FewE *verifier* V *for* L) $L_{ewl(V)} \in$ ZUE
(2) EWL : ZUE $\subset C \implies$ *every* FewE *verifier has oblivious witnesses in* C
(3) KLT : ZUE $\subset$ SIZE$(poly) \implies$ FewE $\subset$ MA

Proof. *Proof of (1):* For any $L \in$ FewE, let V be a verifier whose number of accepting certificates, and running time, both are bounded by 2^{cn}, for some constant c. We construct a new language $L' = \{(x, p) \mid 1 \leq p \leq \sum_y V(x, y) \leq 2^{cn}\}$. Its easy to check that L' has an NE algorithm: guess p distinct accepting certificates for V on input x. Under the assumption NE = ZUE, L' and $\overline{L'}$ have UE algorithms $\mathcal{A}$ and $\mathcal{A}'$, respectively. Now we use these to design an NE algorithm for $L_{ewl(V)}$, which in turn will imply a ZUE algorithm.

On input (x, i), non-deterministically guess a $p \in [1, 2^{cn}]$. Run $\mathcal{A}$ on (x, p) and $\mathcal{A}'$ on $(x, p + 1)$. If they give complementary results, then we known that there are exactly p positive certificates for V on the input x. So we guess p distinct certificates and if V accepts all of them: we accept, if the i^{th}-bit of the lexicographically first certificate is 1.

Proof of (2): From (1), we know that for every $L \in$ FewE, and every FewE verifier V for L, $L_{ewl(V)} \in$ ZUE. Thus, ZUE $\subset C \implies L_{ewl(V)} \in C$, and V has oblivious witnesses in C (due to similar arguments that were given in Theorem 3.3).

Proof of (3): This follows directly from the ZUTIME KLT (Theorem 3.4). □

Theorem 3.6. *The following statements are true (unconditionally):*

(1) Collapse : FewE$/O(n)$ = UE$/O(n)$ = ZUE$/O(n)$
(2) S-to-D : $L \in$ FewE$/O(n) \implies \forall$ (FewE$/O(n)$ *verifier* $V/O(n)$ *for* L) $L_{ewl(V/O(n))} \in$ ZUE$/O(n)$
(3) EWL : ZUE$/O(n) \subset C \implies$ *every* FewE$/O(n)$ *verifier has oblivious witness in* C
(4) KLT : ZUE$/O(n) \subset$ SIZE$(poly) \implies$ FewE$/O(n) \subset$ MA$/O(n)$

Proof. *Proof of (1):* For $L \in$ FewE$/O(n)$, we give a ZUE$/O(n)$ algorithm for L. Let V be a FewE verifier for L that uses $O(n)$ amount of advice. The advice of the ZUE algorithm is: the original advice a used by V, plus a number p to encode the sum of the total number of accepting certificates of V on all n-length inputs. It's easy to check that p only requires extra $O(n)$ bits. On any n-length input x, guess a set S of p pairs (c, d). Output '?' if $\exists (c, d) \in S : V(c, d)/a = 0$, else proceed further. Output 1 if $\exists d : (x, d) \in S$. Else output 0. Its easy to check that exactly one non-deterministic branch outputs in the set $\{0, 1\}$, and that the branch outputs correctly. Thus $L \in$ ZUE$/O(n)$.

Proof of (2): Now we give a $\mathtt{ZUE}/O(n)$ algorithm for $L_{ewl(V/O(n))}$. For input (x, i): the advice, and the algorithm except the final output step, are same as that for L on input x, from the proof of (1). In the final step: output '?' if $\exists(c, d) \in S : \ V(c, d)/a = 0$, else proceed further; output 1 if the i^{th} bit of the lexicographically smallest d such that $(x, d) \in S$ is 1; else output 0. It's easy to check that this is a $\mathtt{ZUE}/O(n)$ algorithm for $L_{ewl(V/O(n))}$. Thus $L_{ewl(V/O(n))} \in \mathtt{ZUE}/O(n)$.

Proof of (3): From (2), we know that for every $L \in \mathtt{FewE}/O(n)$, and every $\mathtt{FewE}/O(n)$ verifier V for L, $L_{ewl(V/O(n))} \in \mathtt{UE}/O(n)$. Thus, $\mathtt{UE} \subset \mathcal{C} \implies L_{ewl(V/O(n))} \in \mathcal{C}$, and $V/O(n)$ has oblivious witnesses in $\mathcal{C}$ (due to similar arguments that were given in Theorem 3.3).

Proof of (4): This follows directly from the $\mathtt{UTIME}$ $\mathtt{KLT}$ (Theorem 3.3). □

4 UNIQUE PROPERTIES VS LOWER BOUNDS

In this section we establish relationships between different types of unique properties and lower bounds against $\mathtt{UTIME}$, $\mathtt{ZUTIME}$, $\mathtt{NTIME}$ and $\mathtt{ZNTIME}$. In many of our connections we use the following connection (Lemma 4.1) between $\mathtt{UP}$-U and $\mathtt{P}$-U properties. The proof of this connection is along the same lines as the original connection [3, 57, 75]: an useful $\mathtt{NP}$ (*resp.* $\mathtt{RP}$-natutal) property yields an useful $\mathtt{P}$ (*resp.* $\mathtt{P}$-natural) property.

Lemma 4.1. $\mathtt{UP}/a$ *property* $\mathcal{U}$ *can be converted into a* $\mathtt{P}/a$ *property* $\mathcal{P}$ *such that:*

(1) $\mathcal{U}$ *is* $\mathtt{UP}/a$-$\mathtt{U}$ *property* $\implies$ $\mathcal{P}$ *is* $\mathtt{P}/a$-$\mathtt{U}$ *property;*
(2) $\mathcal{U}$ *is* $\mathtt{UP}/a$-$u_{=1}$ *property* $\implies$ $\mathcal{P}$ *is* $\mathtt{P}/a$-$u_{=1}$ *property;*
(3) $\mathcal{U}$ *is* $\mathtt{UP}/a$-$u_{\leq 1}$ *property* $\implies$ $\mathcal{P}$ *is* $\mathtt{P}/a$-$u_{\leq 1}$ *property;*
(4) $\mathcal{U}$ *is useful against* $\mathcal{C}$ $\implies$ $\mathcal{P}$ *is useful against* $\mathcal{C}$.

Proof. Let V be the unambiguous verifier corresponding to $\mathcal{U}$'s algorithm. Let c be a constant such that $2^{cn} - 2^n$ is the length of the certificates that V guesses for the inputs of size 2^n. Now we design $\mathcal{P}$ which satisfies the promises of the theorem statement. For m which is not a multiple of c, among all the inputs of length 2^m, $\mathcal{P}$ only accepts the all 0s string. For $m = cn$ for some n, for any input xy where $|x| = 2^n$ and $|y| = 2^{cn} - 2^n$, $\mathcal{P}$ simulates V on (x, y), and accepts if and only if V accepts. For any $n \in \mathbb{N}$, $\mathcal{P}$ uses the same advice for 2^{cn}-size inputs, that $\mathcal{U}$ uses for 2^n-size inputs.

Proofs of (1), (2) & (3): The construction of $\mathcal{P}$ ensures this for the inputs of size 2^m, where m is not a multiple of c. For all the other input sizes this is ensured by the fact that $\mathcal{U}$ is a $\mathtt{UP}$ property, and the behavior of $\mathcal{U}$ on different advice strings. For any $n \in \mathbb{N}$, and any advice string, the number of 2^{cn}-size inputs $\mathcal{P}$ accepts, is same as the number of 2^n-size inputs $\mathcal{U}$ accepts.

Proof of (4): If $\mathcal{U}$ is useful against $\mathcal{C}$, then for each k there exists an infinite subset S_k such that for each $n \in S_k$, $\mathcal{U}(x) = 1 \implies ckt_{\mathcal{C}}(x) > n^k$. For any x, let y be the unique certificate such that $V(x, y) = 1$. Since $ckt_{\mathcal{C}}(x) > n^k \implies ckt_{\mathcal{C}}(xy) > n^k \geq (cn)^{k-1}$ for each k, $\mathcal{P}$ is useful against n^{k-1}-size $\mathcal{C}$ circuits for each k, and hence is useful against $\mathcal{C}$. □

Main results of this section are summarized in the Table 1. Lower bounds in any particular column are all equivalent, and lower bounds from any column imply the lower bounds in the column just below it (except the columns that are separated by double lines). Note that, as the lower bounds get weaker, the properties become less restrictive (or the constructivity goes higher).

Note that, at any place in the Table 1 we can remove the $\log n$ advice by assigning each advice a different input-length. But then the property no more remains a property, instead gets converted into an useful algorithm (see [75]). An useful algorithm, unlike an useful property, accepts inputs of all lengths and not just powers of 2. It appends zeros on the inputs that are not powers of 2, to make it a truth-table. We stick to properties in our presentation.

Table 1. Properties vs Lower Bounds

Type of Properties useful against C	Witness LB	HS LB	Ob-witness LB	Set LB
$\mathsf{P}/\log n\text{-}u_{=1} \equiv \mathsf{P}\text{-}\mathsf{U}$	$\mathsf{ZUE} \not\subset_s C$	$\mathsf{ZUE} \not\subset_{hs} C$	$\mathsf{ZUE} \not\subset_{os} C$	$\mathsf{ZUE} \not\subset C$
$\mathsf{P}/\log n\text{-}u_{\leq 1}$	$\mathsf{UE} \not\subset_w C$			
		$\mathsf{UE} \not\subset_{hw} C$		
			$\mathsf{UE} \not\subset_{ow} C$	$\mathsf{UE} \not\subset C$
$\mathsf{P}/\log n\text{-}\mathsf{U}$	$\mathsf{UE}/n \not\subset_w C$	$\mathsf{UE}/n \not\subset_{hw} C$	$\mathsf{UE}/n \not\subset_{ow} C$	$\mathsf{UE}/n \not\subset C$
$\mathsf{NP}/\log n\text{-}u_{=1} \equiv \mathsf{NP}\text{-}\mathsf{U}$				$\mathsf{ZNE} \not\subset C$
$\mathsf{NP}\text{-}\mathsf{N} \equiv \mathsf{P}\text{-}\mathsf{N}$	$\mathsf{ZNE} \not\subset_s C$	$\mathsf{ZNE} \not\subset_{hs} C$	$\mathsf{ZNE} \not\subset_{os} C$	
$\mathsf{NP}/\log n\text{-}\mathsf{U} \equiv \mathsf{NP}/\log n\text{-}\mathsf{N} \equiv \mathsf{P}/\log n\text{-}\mathsf{N}$	$\mathsf{NE} \not\subset_w C$	$\mathsf{NE} \not\subset_{hw} C$	$\mathsf{NE} \not\subset_{ow} C$	$\mathsf{NE} \not\subset C$
$\mathsf{NP}\text{-}\mathsf{prU}$				$\text{ip-}\mathsf{ZNE} \not\subset C$
$\mathsf{NP}\text{-}\mathsf{prN} \equiv \mathsf{P}\text{-}\mathsf{prN}$	$\text{ip-}\mathsf{ZNE} \not\subset_s C$	$\text{ip-}\mathsf{ZNE} \not\subset_{hs} C$	$\text{ip-}\mathsf{ZNE} \not\subset_{os} C$	

4.1 ZUE lower bounds vs P-U properties

Theorem 4.2. *[Row 1 of Table 1] The following statements are equivalent:*

(1) $\mathsf{ZUE} \not\subset C$
(2) $\mathsf{ZUE} \not\subset_{os} C$
(3) $\mathsf{ZUE} \not\subset_{hs} C$
(4) $\mathsf{ZUE} \not\subset_s C$
(5) $\mathsf{P}\text{-}\mathsf{U} \not\subset_{tt} C$
(6) $\mathsf{P}/\log n\text{-}u_{=1} \not\subset_{tt} C$

Proof. (1) $\implies$ (5) Let $L \in \mathsf{ZUE} \setminus C$, and let V be $2^{O(n)}$-time zero-error unambiguous verifier for L. For any n, L_n can be viewed as a function f_n, where $f_n^{-1}(1) = \{x \in L \mid |x| = n\}$.

Now using V we give a UP-U property $\mathcal{U}$ that is useful against C. Then the result follows from the lemma 4.1.

For any input y of length 2^n, $\mathcal{U}$ simulates V on all the n-length strings, one by one. For each i, it matches the i^{th} bit of y, and the output of V on the i^{th} n-length string. $\mathcal{U}$ accepts if and only if it succeeds in all 2^n verifications.

Constructivity & uniqueness: For $n \in \mathbb{N}$, $\mathcal{U}$ unambiguously accepts the truth table of function f_n, and rejects all the other strings. As it runs for $2^{O(n)}$-time on 2^n-length inputs, it is UP-U (as V is ZUE).

Usefulness: As $L \notin C$, for each k, there are infinitely many input lengths n, such that f_n doesn't have n^k-size C circuits. Thus $\mathcal{U}$ is useful against C.

(5) $\implies$ (4) Let $\mathcal{P}$ be a P-unique property useful against C. Using $\mathcal{P}$ we construct a zero-error unambiguous verifier V for the ZUE language $\{0,1\}^*$ such that V doesn't have seeds in C.

For any n-length input x, V guesses a string y of length 2^n and accepts if and only if $\mathcal{P}$ accepts y. Since $\mathcal{P}$ is P-unique property useful against C, the unique accepting witnesses of V are not in C.

(4) $\implies$ (3) This follows from the definitions.

(3) $\implies$ (2) This follows from the definitions.

(2) $\implies$ (1) The contrapositive follows from the ZUTIME EWL (Theorem 3.4).

(5) $\iff$ (6) The forward direction is trivial. For the reverse direction, for $\mathsf{P}/\log n\text{-}u_{=1}$ property $\mathcal{P}$, we convert $\mathcal{P}$ to a P-U property $\mathcal{P}'$.

For odd m, among all the inputs of length 2^m, $\mathcal{P}'$ only accepts the all 0s string. For $m = 2n$ for some n, for any 2^m-length input $x_1x_2 \dots x_{2^{2n}}$ where $\forall i\ |x_i| = 2^n$, $\mathcal{P}'$ accepts if and only if for each i: $\mathcal{P}$ accepts input x_i with the advice y_i (i^{th} n-length string in the lexicographical order).

Constructivity & uniqueness: These both follow for $\mathcal{P}'$ directly from the fact that $\mathcal{P}$ is a strong-unique P-property.

Usefulness: If $\mathcal{P}$ is useful against C with advice sequence $\{a_n\}_{n\in\mathbb{N}}$, then for each k there exists an infinite subset S_k such that for each $n \in S_k$, $\mathcal{P}(x, a_n) = 1 \implies ckt_C(x) > n^k$. Among all the 2^{2n}-length strings, let y be the unique string that $\mathcal{P}'$ accepts. $y = x_1 \dots x_{b_n} \dots x_{2^{2n}}$, where $\forall i \in [1, 2^n]\ |x_i| = 2^n$, b_n is the lexicographical rank of a_n among all the n-length strings, and $x = x_{b_n}$. Since $ckt_C(x) > n^k \implies ckt_C(y) > n^k \geq (2n)^{k-1}$ for each k, $\mathcal{P}'$ is useful against n^{k-1}-size C circuits for each k, and hence is useful against C. □

4.2 UE lower bounds vs $\mathsf{P}/\log n\text{-}u_{\leq 1}$ properties

We use a fine-grained version of the techniques from [75], to prove the following two theorems: (i) "witness lower bound $\iff$ $\mathsf{P}/\log n\text{-}u_{\leq 1}$ useful property" and (ii) "oblivious witness lower bound $\iff$ UE lower bound". Unfortunately, the "oblivious witness lower bound $\implies$ witness lower bound" connection of NTIME doesn't go through in the case of UTIME. If we try to establish a "oblivious witness lower bound $\implies$ $\mathsf{P}/\log n\text{-}u_{\leq 1}$ useful property" connection, we don't get a mild-unique property, but just a unique property. In the next section we will see that this connection can be established in the presence of advice.

Theorem 4.3. *[Row 2 of Table 1] The following statements are equivalent:*

(1) $\mathsf{UE} \not\subset_w C$
(2) $\mathsf{P}/\log n\text{-}u_{\leq 1} \not\subset_{tt} C$

Proof. (1) $\implies$ (2) Let L' be a UE language, and V' be an unambiguous verifier for L' that doesn't have witnesses in C. By a simple padding argument we can construct $L \in \mathsf{UTIME}(2^n)$, and an unambiguous verifier V for L with certificate length 2^n, that doesn't have witnesses in C.

If the inputs are given as advice, and the certificates are given as inputs, then V becomes a $\mathsf{P}/\log n$ property $\mathcal{P}$, that is useful against C.

For $\mathcal{P}$ to be a $u_{\leq 1}$ property, it should be unique with respect to the same advise that makes it useful. At this point, all we know is that for every input length and every advise, $\mathcal{P}$ accepts at most one truth-table (since V is unambiguous). The advise that makes $\mathcal{P}$ useful may not be present for all input lengths. For some of these input lengths n where no such advise is present, it is also possible that L_n is empty (i.e. no advice is present that makes the property non-empty).

We will be done if L is non-empty for all input lengths. Consider the two modifications of V: (i) V_0, that changes its behavior on the all 0s strings and always accepts them (unambiguously); and (ii) V_1, that changes its behavior on the all 1s strings and always accepts them (unambiguously). The modified languages and their corresponding verifiers are also $\mathsf{UTIME}(2^n)$, and have 2^n-length certificates. We show that at least one of these two modifications doesn't have witness in C. If V_0 has witnesses in C, then V's witnesses corresponding to the all 0s strings must be the ones that were not in C (at least infinitely often), so then V_1 doesn't have witnesses in C (infinitely often).

(2) $\implies$ (1) Let $\mathcal{P}$ be a $\mathsf{P}/\log n\text{-}u_{\leq 1}$ property that is useful against C. Define $L = \{x \mid \exists y\, \mathcal{P}(y)/x = 1\}$. Let V be the verifier for L, that on any n-length input x, guesses a string y of length 2^n, and simulates $\mathcal{P}$ on y using x as advice. V is a UE verifier since $\mathcal{P}$ is a mild-unique property. V doesn't have witnesses in C since $\mathcal{P}$ is useful against C. □

Theorem 4.4. *[Row 4 of Table 1] The following statements are equivalent:*

(1) $\mathsf{UE}/a \not\subset C$
(2) $\mathsf{UE}/a \not\subset_{ow} C$

Proof. $\neg(1) \implies \neg(2)$ This follows from the UTIME EWL (Theorem 3.3).

$\neg(2) \implies \neg(1)$ Assume that UE/a has oblivious witnesses (for all verifiers that are unambiguous given the correct advice) in C. Let L be a UE/a language and V/a be a UE/a verifier for L. By our assumption, V/a has oblivious witnesses in n^k-size C circuits. Now we show that $L \in C$.

Using V/a we construct an unambiguous verifier V'/a for the UEXP/a language $\{0,1\}^*$ such that: for $n \in \mathbb{N}$, an oblivious witness circuit for V'/a on n-length inputs, computes L_n.

For $x \in L$ (this can be verified by brute forcing through all the n^{k+1}-size circuits), $V'(x,y)/a = 1$ only when y is the all 1s string. For $x \notin L$, $V'(x,y)/a = 1$ only when y is the all 0s string. Since UE/a has oblivious witnesses in C, by a simple padding argument UEXP/a too has oblivious witnesses in C. Let $\{C_n\}_{n\in\mathbb{N}}$ be the C circuit family encoding the oblivious witnesses of V'. Then, the C circuit family defined by $D_n(x) = C_n(x,1)$ encodes the language L (since the first bit of the unique accepting certificate of V'/a on input x, dictates whether $x \in L$ or not). □

4.3 UE/n lower bounds vs $\mathsf{P}/\log n$-U properties

The arguments from Section 4.1, when extended to the advice setting, circumvent the problems from Section 4.2, and yield the following theorem.

Theorem 4.5. *[Row 5 of Table 1] The following statements are equivalent for any constant $k \geq 1$:*

(1) $\mathsf{UE}/n^k \not\subset C$
(2) $\mathsf{UE}/n^k \not\subset_{ow} C$
(3) $\mathsf{UE}/n^k \not\subset_{hw} C$
(4) $\mathsf{UE}/n^k \not\subset_{w} C$
(5) $\mathsf{P}/\log^k n$-U $\not\subset_{tt} C$

Proof. (1) $\implies$ (5) Let $L \in \mathsf{UE}/n^k \setminus C$, and let V be $2^{O(n)}$-time unambiguous verifier for L. For any n, L_n can be viewed as a function f_n, where $f_n^{-1}(1) = \{x \in L \mid |x| = n\}$.

Now using V we give a $\mathsf{UP}/\log^k m$-U property $\mathcal{U}$ that is useful against C. Then the result follows from the lemma 4.1.

For odd m, among all the inputs of length 2^m, $\mathcal{U}$ only accepts the all 0s string. For $m = 2n$ for some n, for any 2^m-length input yz with $|y| = 2^n$ and $|z| = 2^{2n} - 2^n$, $\mathcal{U}$ goes through all the n-length strings, one by one. If the i^{th} bit of y is 0, it does nothing. If the i^{th} bit of y is 1, it simulates V on the i^{th} n-length string in the lexicographical order (to verify its inclusion in L). The first n^k bits of advice is the advice required for the simulation of V. The rest of the $\log^k(2^{2n}) - n^k = (2n)^k - n^k \geq n$ bits of advise encodes the number of n-length inputs that V accepts. $\mathcal{U}$ accepts if and only if: (i) it succeeds in all 2^n verifications; (ii) the hamming weight of y is equal to the number encoded by the last $(2n)^k - n^k$ bits of advise; and (iii) z is an all 0s string.

Constructivity & uniqueness: For $n \in \mathbb{N}$, $\mathcal{U}$ unambiguously accepts the truth table of function f_n (followed by an all 0s string of length $2^{2n} - 2^n$), and rejects all the other strings. As it runs for $2^{O(n)}$-time for 2^n-length inputs with n^k-size advice, it is $\mathsf{UP}/\log^k n$-U (as V is UE).

Usefulness: As $L \notin C$, for each l, there are infinitely many input lengths n, such that f_n doesn't have n^{l+1}-size C circuits. Corresponding to each such n, for the inputs of length 2^{2n}, $\mathcal{U}$ accepts strings y that doesn't have $(2n)^l$-size C circuits because: any $(2n)^l$-size circuit C with $tt(C) = y$, decides L_n after we fix the first half of its input wires to 1s, and $(2n)^l \leq n^{l+1}$.

(5) $\implies$ (4) Let $\mathcal{P}$ be a $\mathsf{P}/\log^k n$-U property useful against C. We construct an unambiguous verifier V for the UE/n^k language $\{0,1\}^*$, that doesn't have witnesses in C. For any n-length input x, guess a 2^n-length string y and simulate $\mathcal{P}$ on y, and accept if an only if $\mathcal{P}$ accepts.

Since $\mathcal{P}$ is useful against C, V doesn't have witnesses in C. As $\mathcal{P}$ is unique, V is UE/n^k.

(4) $\implies$ (3) This follows from the definitions.

(3) $\implies$ (2) This follows from the definitions.

(2) $\implies$ (1) The contrapositive follows from the UTIME EWL (Theorem 3.3). □

4.4 ZNE lower bounds vs NP-U properties

In [57] it was conjectured, "$\mathsf{ZNE} \not\subset C \iff \exists$ P-N (or NP-N) property useful against C" while only forward direction was proved. We use a fine-grained version of the proof to establish the equivalence in the case of unique properties. So if this conjecture is true, then any NP-N property has an equivalent NP-U property.

In the Section 4.5 we establish the equivalence between: NP-N properties and lower bounds for ZNE seeds. Since NP-U properties imply NP-N properties, this result can be viewed as an reverse-EWL for ZNE. Moreover, if the conjuncture of [57] is true, then we also get an EWL for ZNE.

In the Section 4.6 we show an equivalence between NP-N and NP-U properties, when they are allowed to use $\log n$ amount of advice. This equivalence uses the EWL for NE. In the Section 5 we reduce this advice to $O(1)$, on the expense of making the lower bounds fix-polynomial. This gives us an EWL and reverse-EWL for $\mathsf{ZNE}/O(1)$ for fix-polynomial upper/lower bounds.

Theorem 4.6. *[Row 6 of Table 1] The following statements are equivalent:*

(1) $\mathsf{ZNE} \not\subset C$
(2) $\mathsf{NP}/\log n$-$u_{=1}$ $\not\subset_{tt}$ C
(3) NP-U $\not\subset_{tt}$ C

Proof. (2) $\iff$ (3) The reverse direction is trivial. For the forward direction, we convert any $\mathsf{NP}/\log n$-$u_{=1}$ property $\mathcal{P}$ into an NP-U property $\mathcal{P}'$. The conversion from the proof of Theorem 4.2 works for NP properties as well.

(1) $\implies$ (3) Let $L \in \mathsf{ZNE} \setminus C$, and let V be $2^{O(n)}$-time zero-error non-deterministic verifier for L. For any n, L_n can be viewed as a function f_n, where $f_n^{-1}(1) = \{x \in L \mid |x| = n\}$.

Now using V we give an NP-U property $\mathcal{U}$ that is useful against C. For any input y of length 2^n, $\mathcal{U}$ simulates V on all the n-length strings, one by one. If the i^{th} bit of y is 0, it verifies the inclusion of the i^{th} (lexicographically) n-length string in $\overline{L}$. If the i^{th} bit of y is 1, it verifies the inclusion of the i^{th} (lexicographically) n-length string in L. $\mathcal{U}$ accepts if and only if it succeeds in all 2^n verifications.

Constructivity & uniqueness: For $n \in \mathbb{N}$, $\mathcal{U}$ accepts the truth table of function f_n, and rejects all the other strings. As it runs for $2^{O(n)}$-time on 2^n-length inputs, it is NP-U.

Usefulness: As $L \notin C$, for each k, there are infinitely many input lengths n, such that f_n doesn't have n^k-size C circuits. Thus $\mathcal{U}$ is useful against C.

(3) $\implies$ (1) Let $\mathcal{U}$ be an NP-unique property useful against C. We construct a language L in $\mathsf{ZNE} \setminus C$, whose ZNE verifier uses $\mathcal{U}$.

For any n-length input x, V guesses a string y of length 2^n and simulates $\mathcal{U}$ on it. Let the lexicographical rank of x (among all n-bit strings) be i. V outputs '?' if $\mathcal{U}$ rejects, else it proceeds further. It outputs 1, if y's i^{th}-bit is equal to 1. Else it outputs 0.

Since $\mathcal{U}$ is NP-unique property, for each n the 2^n-length string y_n it accepts is unique. Thus V accepts the language whose slices are represented by the strings y_n (let's call this language L), and satisfies the promises of a ZNE verifier.

Since $\mathcal{U}$ is useful against C, for each k, there are infinitely many values of n where y_n doesn't have n^k-size C circuits. Thus $L \notin C$. □

The above proof also gives an equivalence between NP-promise-unique properties and lower bounds against ip-ZNE (the promise version of ZNE).

Theorem 4.7. *[Row 9 of Table 1] The following statements are equivalent:*

(1) $ip\text{-ZNE} \not\subset C$
(2) $\text{NP-prU} \not\subset_{tt} C$

Proof Idea: ($\implies$) Any $L \in \text{ip-ZNE} \setminus C$ that satisfies the ZNE or (NE $\cap$ Co-NE)-promise on n-length inputs for some n, yields a property that is unique on the input lengths 2^n. Since L satisfies the lower bound on the promise inputs, the property is useful on the inputs on which it is unique.
($\impliedby$) Any NP-unique useful property $\mathcal{U}$ that is unique on 2^n-length inputs for some n, yields a ZNE verifier that satisfies the ZNE-promise on n-length inputs. Since useful inputs of $\mathcal{U}$ also satisfy the promise, the verifier satisfies the lower bound on the promise inputs. □

4.5 ZNE lower bounds vs NP-N properties

Theorem 4.8. *[Row 7 of Table 1] The following statements are equivalent:*

(1) $\text{ZNE} \not\subset_{os} C$
(2) $\text{ZNE} \not\subset_{hs} C$
(3) $\text{ZNE} \not\subset_{s} C$
(4) $\text{NP-N} \not\subset_{tt} C$
(5) $\text{P-N} \not\subset_{tt} C$

Proof. (4) $\iff$ (5) It is proved in [57].
(3) $\implies$ (2) This follows from the definitions.
(2) $\implies$ (1) This follows from the definitions.
(1) $\implies$ (5) Let V be a ZNE verifier that doesn't have oblivious seeds in C. Let V's certificate length be 2^{cn}, for some constant c. Using V we construct a P-N property $\mathcal{P}$ useful against C. View any $2^{(c+1)n}$-length input as a collection of 2^n certificates. $\mathcal{P}$ accepts if and only if, for each $i \in [1, 2^n]$, V outputs in $\{0, 1\}$ on the i^{th} (lexicographically) n-length input when given the i^{th} certificate from the collection. Clearly, $\mathcal{P}$ is a P-N property. It is useful against C as it only accepts oblivious witnesses of V.
(5) $\implies$ (3) Let $\mathcal{P}$ be a P-N property useful against C. For each k, let S_k be the infinite set of inputs where $\mathcal{P}$ only accepts strings str with $ckt_C(str) \geq n^k$. Using $\mathcal{P}$ we construct a ZNE verifier V for $\{0, 1\}^*$ that doesn't have seeds in C. For n-length input x, V guesses a string y of length 2^n. V outputs 1, if $\mathcal{P}$ accepts the string y. Otherwise V outputs '?'. For each k, for any n with $2^n \in S_k$, due to the way it is constructed, V doesn't have seeds in n^k-size C. Thus V doesn't have seeds in C. □

The above proof also gives an equivalence between NP-promise properties and lower bounds for ip-ZNE seeds. The proof idea is similar to the one given for the Theorem 4.7.

Theorem 4.9. *[Row 10 of Table 1] The following statements are equivalent:*

(1) $ip\text{-ZNE} \not\subset_{os} C$
(2) $ip\text{-ZNE} \not\subset_{hs} C$
(3) $ip\text{-ZNE} \not\subset_{s} C$
(4) $\text{NP-prN} \not\subset_{tt} C$
(5) $\text{P-prN} \not\subset_{tt} C$

4.6 NE lower bounds vs NP/log n-U properties

Theorem 4.10. *[Row 8 of Table 1] The following statements are equivalent:*

(1) $\mathsf{NE} \not\subset C$
(2) $\mathsf{NE} \not\subset_{ow} C$
(3) $\mathsf{NE} \not\subset_{hw} C$
(4) $\mathsf{NE} \not\subset_{w} C$
(5) $\mathsf{NP}/\log n\text{-U} \not\subset_{tt} C$
(6) $\mathsf{NP}/\log n\text{-N} \not\subset_{tt} C$
(7) $\mathsf{P}/\log n\text{-N} \not\subset_{tt} C$

Proof. *Equivalence of (1), (2), (4), (6) & (7):* It is proved in [57, 75].
Equivalence with (3): The implications, (4) $\implies$ (3) and (3) $\implies$ (2), follow from the definitions.
Equivalence with (5): The implication (5) $\implies$ (6) follows from the definitions. We will now show the implication (1) $\implies$ (5).

Let $L \in \mathsf{NE} \setminus C$, and let V be $2^{O(n)}$-time non-deterministic verifier for L. For any n, L_n can be viewed as a function f_n, where $f_n^{-1}(1) = \{x \in L \mid |x| = n\}$.

Now using V we give an NP/log n-U property $\mathcal{U}$ that is useful against C. For any input y of length 2^n, $\mathcal{U}$ goes through all the n-length strings, one by one. If the i^{th} bit of y is 0, it does nothing. If the i^{th} bit of y is 1, it simulates V on the i^{th} n-length string in the lexicographical order (to verify its inclusion in L). $\mathcal{U}$ accepts if and only if it succeeds in all 2^n verifications and the hamming weight of y is equal to the number encoded by the advice. If the advice is equal to the size of L_n, $\mathcal{U}$ accepts the truth table corresponding to the function f_n, and rejects all the other strings. Since it runs in $2^{O(n)}$-time on 2^n-length inputs, it is NP/log n-U. As $L \notin C$, $\mathcal{U}$ is useful against C. □

5 ISOLATION OF PROPERTIES: EWL & KLT FOR ZNE

In this section we discuss the consequences of isolating properties with different constructivity. By isolation we mean: extracting (*resp.* proving existence of) an useful unique property from (*resp.* from the existence of) an arbitrary useful property. From the Table 1 we know that:

(1) Isolation of P-properties is equivalent to: $\mathsf{ZUE} \subset C \iff \mathsf{ZNE} \subset_{os} C$.
(2) Isolation of P/log n-properties is equivalent to: $\mathsf{UE}/n \subset C \iff \mathsf{NE} \subset C$.
(3) Isolation of NP-properties is equivalent to: $\mathsf{ZNE} \subset C \iff \mathsf{ZNE} \subset_{os} C$.
(4) Isolation of NP-promise-properties is equivalent to: ip-$\mathsf{ZNE} \subset C \iff$ ip-$\mathsf{ZNE} \subset_{os} C$.
(5) Isolation of NP/log n-properties was already achieved in the Theorem 4.10.

In this section we focus on the points (4) and (5). For the case of fix-polynomial lower bounds: we merge the rows 6 and 7 (in presence of $O(1)$ advice), and rows 9 and 10 of the Table 1. Most of the equivalences follow from the arguments from the previous section: Theorems 4.6, 4.7, 4.8 and 4.9. The main technical results of this section are the implications: (i) $\forall\, k \geq 1$ $\mathsf{ZNE} \not\subset_{os} C(n^k) \implies \forall\, k \geq 1$ $\mathsf{ZNE}/1 \not\subset C(n^k)$; (ii) $\forall\, k \geq 1$ $\mathsf{ZNE} \not\subset_{os}$ io-$C(n^k) \implies \forall\, k \geq 1$ $\mathsf{ZNE} \not\subset C(n^k)$; and (iii) $\forall\, k \geq 1$ ip-$\mathsf{ZNE} \not\subset_{os} C(n^k) \implies \forall\, k \geq 1$ ip-$\mathsf{ZNE} \not\subset C(n^k)$. Contrapositive of these can be viewed as EWLs for ZNE. Using these EWLs we derive KLTs for ZNE, and isolation results for NP-properties. We also use the following folklore result to make our EWLs work for typical circuit classes.

Lemma 5.1. *If $\mathsf{P} \subset C$, then there exists a constant c such that: for large enough n, any s-size circuit has an equivalent s^c-size C circuit.*

Proof. Ckt-Eval is a problem in P whose input is a Boolean circuit C and a string x, and the output is the output of C on x. If $\mathsf{P} \subset C$, then there is a constant c such that Ckt-Eval has $n^{c/2}$-size C circuits.

Let B be a P/poly circuit of size s. Let E be $(n + s \log s)^{c/2}$-size circuit corresponding to the $(n + s \log s)^{th}$-slice of Ckt-Eval. Define $D(x) = E(B, x)$. It is easy to check that: (i) D is an s^c-size C circuit; and (ii) D is equivalent to B. □

Now we prove the ZNE EWL and KLT.

Theorem 5.2. *For constant $k \geq 1$:*

(1) $\mathtt{ZNE}/1 \subset C(n^k) \implies \exists\, k' \geq 1\ \mathtt{ZNE} \subset_{os} C(n^{k'})$
(2) $\mathtt{ZNE}/O(1) \subset C(n^k) \implies \exists\, k' \geq 1\ \mathtt{ZNE}/O(1) \subset_{os} C(n^{k'})$
(3) $\mathtt{ZNE} \subset C(n^k) \implies \exists\, k' \geq 1\ \mathtt{ZNE} \subset_{os} \textit{io-}C(n^{k'})$
(4) $\textit{ip-}\mathtt{ZNE} \subset C(n^k) \implies \exists\, k' \geq 1\ \mathtt{ZNE} \subset_{os} C(n^{k'})$
(5) $\textit{ip-}\mathtt{ZNE} \subset C(n^k) \implies \exists\, k' \geq 1\ \textit{ip-}\mathtt{ZNE} \subset_{os} C(n^{k'})$

Proof. We prove these results for the unrestricted Boolean circuits. The result for the circuit class C follows from the Lemma 5.1. All the assumptions imply $\mathtt{P} \subset C$, thus any $\mathtt{SIZE}(n^{k'})$ circuit has an equivalent $C(n^{ck'})$ circuit, for some constant c.

Proof of (1): $\mathtt{ZNE}/1 \subset \mathtt{SIZE}(n^k)$ implies $\mathtt{EXP} \subset \mathtt{SIZE}(poly)$, and thus $\mathtt{EXP} = \mathtt{MA} \cap \mathtt{Co\text{-}MA}$ [9].

Now we show that, if $\forall k' \geq 1\ \mathtt{ZNE} \not\subset_{os} \mathtt{SIZE}(n^{k'})$, then $\mathtt{MA} \cap \mathtt{Co\text{-}MA} \subset \text{io-}\mathtt{ZNE}/1$. Combined with the above statement it leads to the contradiction $\mathtt{EXP} \subset \text{io-}\mathtt{SIZE}(n^k)$ (since we can diagonalize against fix-polynomial size circuits in EXP).

$\forall k \geq 1\ \mathtt{ZNE} \not\subset_{os} \mathtt{SIZE}(n^k) \implies \forall k \geq 1\ \mathtt{P\text{-}N} \not\subset_{tt} \mathtt{SIZE}(n^k)$ (the arguments from Theorem 4.8 apply to the fix-polynomial lower bounds as well). For any $L \in \mathtt{MA} \cap \mathtt{Co\text{-}MA}$: we derandomize the MA protocols for L and $\overline{L}$ using the useful P-N properties to give an ZNE/1 algorithm that works infinitely often. For any constant p let $\mathcal{N}_p$ be a P-N property useful against n^p-size circuits.

The ZNE/1 *algorithm:* After including the non-determinism of Merlin into Arthur's input: let the size of the circuit C (*resp.* C') that captures the BP computation of Arthur for L (*resp.* L') be bounded by n^l, for some constant l. We use the property $\mathcal{N}_{lg}$, where g is the constant from Theorem 2.4. The 1-bit of advice indicates whether the property is useful or not. If it's 0, the algorithm always outputs 0. If it's 1, the algorithm guesses a 2^n-bit string Y and simulates $\mathcal{N}_{lg}$ on Y. It outputs '?' if $\mathcal{N}_{lg}$ rejects Y, else it proceeds further and guesses another bit b. If $b = 0$: it derandomizes C' (after guessing Merlin's non-determinism) and outputs 0 if the acceptance probability is $\geq 1/2$, else outputs '?'. If $b = 1$: it derandomizes C (after guessing Merlin's non-determinism) and outputs 1 if the acceptance probability is $\geq 1/2$, else outputs '?'.

Derandomization: The property $\mathcal{N}_{lg}$ yields truth-tables that don't have n^{lg}-size circuits, infinitely often. Once we have access to these truth-tables, we construct a PRG $G : n \to n^l$ using the Theorem 2.4, that fools n^l-size circuits. We brute-force through the seeds of G to compute the acceptance probability of the circuits C and C' in $2^{O(n)}$-time (within $\pm 1/n^l$ error).

Proof of (2): It's same as (1), except we use the fact that the arguments from Theorem 4.8 also apply in the advice setting and yield: $\forall k \geq 1\ \mathtt{ZNE}/O(1) \not\subset_{os} \mathtt{SIZE}(n^k) \implies \forall k \geq 1\ \mathtt{P}/O(1)\text{-}\mathtt{N} \not\subset_{tt} \mathtt{SIZE}(n^k)$. The extra 1-bit of advice used to indicate the usefulness of the property during derandomization, now hides in the $O(1)$ advice.

Proof of (3): It's same as (1), except we use the fact that the arguments from Theorem 4.8 also yield: $\forall k \geq 1\ \mathtt{ZNE} \not\subset_{os} \text{io-}\mathtt{SIZE}(n^k) \implies \forall k \geq 1\ \mathtt{P\text{-}N} \not\subset_{tt} \text{io-}\mathtt{SIZE}(n^k)$. That is, if we start with a ZNE verifier V whose oblivious-seeds have high circuit complexity on all input lengths, then we get a P-N property that is useful everywhere. Now, when we derandomize any $L \in \mathtt{MA} \cap \mathtt{Co\text{-}MA}$, we don't need that one bit of advice.

Proof of (4): It's same as (1), except if we don't use that 1-bit of advice to encode the usefulness of the property during derandomization, we get an ip-ZNE algorithm. The ZNE-promise is met only when the property is useful.

Proof of (5): Since in (4) we don't use advice to encode the usefulness, we might as well use promise property. So we use the Theorem 4.9 instead, to get: $\forall k \geq 1$ ip-ZNE $\not\subset_{os}$ SIZE$(n^k) \implies \forall k \geq 1$ P-prN $\not\subset_{tt}$ SIZE(n^k). □

Using the above EWL we give the following KLT.

Theorem 5.3. *For constant $k \geq 1$:*

(1) ZNE$/1 \subset$ SIZE$(n^k) \implies$ ZNE $\subset$ MA
(2) ZNE$/O(1) \subset$ SIZE$(n^k) \implies$ ZNE$/O(1) \subset$ MA$/O(1)$
(3) ZNE $\subset$ SIZE$(n^k) \implies$ ZNE $\subset$ *io*-MA
(4) *ip*-ZNE $\subset$ SIZE$(n^k) \implies$ ZNE $\subset$ MA
(5) *ip*-ZNE $\subset$ SIZE$(n^k) \implies$ *ip*-ZNE $\subset$ MA

Proof Idea: All the assumptions give EXP $\subset$ SIZE$(poly)$ or EXP$/O(1) \subset$ SIZE$(poly)$. This gives EXP = MA or EXP$/O(1)$ = MA$/O(1)$ from [9]. From the Theorem 5.2, all these assumptions give circuit upper bounds on the oblivious-seeds of ZNE, ZNE$/O(1)$ or ip-ZNE. Brute-forcing through these circuits that encode the seeds, we get collapses to EXP, EXP$/O(1)$ or io-EXP. □

Using the above EWL we also give the following isolation results.

Theorem 5.4. *For constant $k \geq 1$:*

(1) $\forall\, k \geq 1$ NP-N $\not\subset_{tt} C(n^k) \implies \forall\, k \geq 1$ NP/1-U $\not\subset_{tt} C(n^k)$
(2) $\forall\, k \geq 1$ NP$/O(1)$-N $\not\subset_{tt} C(n^k) \iff \forall\, k \geq 1$ NP$/O(1)$-U $\not\subset_{tt} C(n^k)$
(3) $\forall\, k \geq 1$ NP-N $\not\subset_{tt}$ *io*-$C(n^k) \implies \forall\, k \geq 1$ NP-U $\not\subset_{tt} C(n^k)$
(4) $\forall\, k \geq 1$ NP-N $\not\subset_{tt} C(n^k) \implies \forall\, k \geq 1$ NP-prU $\not\subset_{tt} C(n^k)$
(5) $\forall\, k \geq 1$ prN-N $\not\subset_{tt} C(n^k) \iff \forall\, k \geq 1$ NP-prU $\not\subset_{tt} C(n^k)$

Proof Idea: The result follows if we replace the hypothesizes and the conclusions, in the contrapositive of these statements, by equivalent hypothesizes and conclusions from the Theorems 4.6, 4.7, 4.8 and 4.9. □

This points (2) and (5) of the above theorem also yield the following more general result.

Theorem 5.5. *The following statements are equivalent:*

(1) $\forall k \geq 1$ ZNE$/O(1) \not\subset C(n^k)$ *(resp.* $\forall k \geq 1$ *ip*-ZNE $\not\subset C(n^k)$*)*
(2) $\forall k \geq 1$ ZNE$/O(1) \not\subset_{os} C(n^k)$ *(resp.* $\forall k \geq 1$ *ip*-ZNE $\not\subset_{os} C(n^k)$*)*
(3) $\forall k \geq 1$ ZNE$/O(1) \not\subset_{hs} C(n^k)$ *(resp.* $\forall k \geq 1$ *ip*-ZNE $\not\subset_{hs} C(n^k)$*)*
(4) $\forall k \geq 1$ ZNE$/O(1) \not\subset_{s} C(n^k)$ *(resp.* $\forall k \geq 1$ *ip*-ZNE $\not\subset_{s} C(n^k)$*)*
(5) $\forall k \geq 1$ NP-U$/O(1) \not\subset_{tt} C(n^k)$ *(resp.* $\forall k \geq 1$ NP-prU $\not\subset_{tt} C(n^k)$*)*
(6) $\forall k \geq 1$ NP-N$/O(1) \not\subset_{tt} C(n^k)$ *(resp.* $\forall k \geq 1$ NP-prN $\not\subset_{tt} C(n^k)$*)*
(7) $\forall k \geq 1$ P-N$/O(1) \not\subset_{tt} C(n^k)$ *(resp.* $\forall k \geq 1$ P-prN $\not\subset_{tt} C(n^k)$*)*

6 UEXP LOWER BOUNDS FROM FAST UNAMBIGUOUS ALGORITHMS

In this section we show how to get lower bounds from fast unambiguous algorithms: from half-sub-exponential algorithms combined with the UTIME KLT (Section 6.1); by a generalization of the "tight reductions to lower-bounds" connection of [73] combined with the UTIME EWL (Section 6.2); by a generalization of the "learning to lower bounds" connection of [24] combined with the UTIME KLT (Section 6.3). Finally, in Section 6.4 we show some generalizations of lower bound frameworks. We use the following 'UTIME hierarchy' in our proofs.

Theorem 6.1 (Heirarchy for UTIME [26]). *For any time bound t such that $n \leq t \leq 2^n$, there is a constant $\epsilon > 0$ and an advice bound $a \in O(\log(t)\log(\log(t)))$ such that* UTIME$(t)/a \not\subset$ UTIME$(t^\epsilon)/(a+1)$.

6.1 UEXP lower bounds from UTIME KLT: a warm-up case

From [73] we know: a deterministic half-sub-exponential 3-SAT algorithm imply EXP $\not\subset$ SIZE($poly$). Two interesting algorithm design questions are: "Does fast non-deterministic algorithms exist for TAUT/CAPP? [19]" and "Does fast unambiguous non-deterministic algorithms exist for SAT?". Here we show that (strong) positive answers to these questions imply UEXP lower bounds.

Main idea: UEXP/$a \subset$ SIZE($poly$) with UTIME KLT implies UEXP/$a = \Sigma_2/a = \Pi_2/a =$ MA/a. Then, for $L \in$ UEXP/a, we unfold the quantifiers of the MA, Σ_2, Π_2 algorithms, and use the faster algorithms (from the assumptions) to contradict the UTIME hierarchy.

Theorem 6.2. *Let $f, g : \mathbb{N} \to \mathbb{N}$ satisfy $f(g(n^{O(1)})^{O(1)})^{O(1)} \in O(2^n)$ and $h(n^{O(1)})^{O(1)} \in O(2^n)$, and $b \in \{0, 1\}$. Then* UE/$\omega(n \log n) \not\subset$ SIZE($poly$) *if:*

(1) 3-SAT $\in$ UTIME(f)/b and 3-TAUT $\in$ NTIME(g)/$1 - b$; or
(2) Σ_2-SAT $\in$ UTIME(h)/1 (*resp.* Π_2-SAT $\in$ UTIME(h)/1); or
(3) 3-SAT $\in$ UTIME(f)/b and CAPP $\in$ NTIME(g)/$1 - b$.

Here, CAPP (circuit approximation probability problem) is the problem where the input is an n-input $s(n)$-size circuit C, and the output is the fraction of the 2^n inputs that C accepts (within $\pm 1/s$ error). $s(n)$ is $poly(n)$ by default, and the complexity is measured in terms of n and not $s(n)$. For $s(n) = poly(n)$, these problems turn into classes of problems: one problem for each polynomial. We abuse the notation and treat these classes as single problems.

6.2 Lower bounds from UTIME EWL

We use the following 'tight reductions to 3-USAT' in this section.

Theorem 6.3 (Efficient local reductions [25, 37, 67]). *Every language $L \in$ UTIME(2^n) can be reduced to* 3-USAT *(uniquely satisfiable* 3-SAT*) instances of $2^n n^c$-size, for some constant c. Moreover, given an instance of L there is an n^c-size C (*P*-uniform) circuit that, on an integer $i \in [2^n n^c]$ in binary as input, outputs the i^{th}-clause of the resulting* 3-USAT *formula.*

In [73] they showed: any super-polynomial savings in designing non-deterministic algorithms for TAUT imply NEXP $\not\subset$ SIZE($poly$). We extend this to: faster unambiguous algorithms for, TAUT and canonization, imply UEXP lower bounds. We first formally define canonization.

Canonization : A subset S of circuits is called CAN$_{(s,C,p)}$, if for any s-size C circuit C, there exists a unique circuit $C' \in S$ with $tt(C) = tt(C')$, and $|C'| \leq p(ckt_C(tt(C')))$. CAN$_{(s,C,p)} \in \Gamma/a$ means there is a Γ/a algorithm that decides CAN$_{(s,C,p)}$.

TAUT$_{(s,C)}$ (*resp.* SAT$_{(s,C)}$) denotes the TAUT (*resp.* SAT) for s-size C circuits.

In these definitions we omit, the parameter s when it is $poly(n)$, and the circuit class when C = Boolean. For $s = poly(n)$, here again we abuse the notation and treat these classes as single problems.

Main idea: In [73] they combined the witness circuit with the reduction circuit (from Theorem 9.5), and used a fast TAUT algorithm. We do the same except: we use UTIME EWL, and we use canonization to unambiguously guess the witness circuit.

Theorem 6.4. *For $\delta \leq 1$, let a, c and ϵ be the parameters of Theorems 6.1 and 6.3 for the time bound $t = 2^{\delta n}$. Then for constant k and function $p(n) \geq n$,* UTIME($2^{\delta n}$)/$a \not\subset C(n^k)$ *if:*

(1) TAUT$_{(p(n^{k+1})n+n^c,C)} \in$ UTIME($2^{\epsilon n}$) *and* CAN$_{(n^{k+1},C,p)} \in$ UTIME($2^{\epsilon n}$)/1*; or*
(2) TAUT$_{(p(n^{k+1})n+n^c,C)} \in$ UTIME($2^{\epsilon n}$)/1 *and* CAN$_{(n^{k+1},C,p)} \in$ UTIME($2^{\epsilon n}$).

Proof. Using the assumptions (1 or 2), we will contradict the UTIME hierarchy (Theorem 6.1) by designing a UTIME($2^{\epsilon n}$)/(a + 1) algorithm for arbitrary $L \in$ UTIME($2^{\delta n}$)/a.

Reduction circuit: For $L \in \mathtt{UTIME}(2^{\delta n})/a$ and input x, let F_x be the $2^n n^c$-size 3-SAT formula we get by reducing from x (Theorem 6.3). There is an n^c-size (P-uniform) C circuit D with $n + c \log n$ input wires, that outputs the i^{th}-clause of F when given the input $i \in [1, 2^n n^c]$.

Special verifier: Let V be the verifier for L that first reduces input x to the 3-SAT formula F_x, and then non-deterministically guesses a satisfying assignment for F_x.

Easy-witness circuit: From UTIME EWL (Theorem 3.3) and the assumption $\mathtt{UTIME}(2^{\delta n})/a \subset C(n^k)$ we know that V has witness circuits in $C(n^k)$. Let E be a witness circuit of this verifier for the input length $|x| = n$.

Final circuit C: Combining D and E we construct a circuit C that satisfies: "C is a tautology $\iff$ $x \in L$". On input i, the output of D is $3n + 3c \log n + 3$ bits. The first $3n + 3c \log n$ bits are the three variables of the i^{th}-clause of F. Plug these output bits to three separate copies of E. The last three bits indicate whether the corresponding literals are positive or negative. Use these three bits and the three output bits from the three copies of E to compute the value of the i^{th}-clause (based on the assignment encoded by $tt(E)$).

Contradicting the first assumption: Non-deterministically guess a $p(n^{k+1})$-size C circuit E. Simulate the $\mathtt{CAN}_{(n^{k+1},C,p)}$ algorithm on E. This requires $\mathtt{UTIME}(2^{\epsilon n})/1$. Reject if the answer is negative. Continue if it's positive, and construct C as described above. $|C| \leq p(n^{k+1})n + n^c$. Note that, for any truth-table only one non-deterministic branch will lead to a non-rejecting path. Now simulate the $\mathtt{TAUT}_{(p(n^{k+1})n+n^c,C)}$ algorithm on C. This requires $\mathtt{UTIME}(2^{\epsilon n})$. Note that, C is accepted if and only if, $x \in L$, and $tt(E)$ is the unique witness of V. This whole process requires the advice used in the $\mathtt{UTIME}(2^{\delta n})/a$ algorithm for L. So we get a $\mathtt{UTIME}(2^{\epsilon n})/(a+1)$ algorithm.

Contradicting the second assumption: The algorithm is exactly the same, expect that the extra 1-bit of advice is used at an later stage of the algorithm. □

We get the following corollary that is cleaner in presentation.

Corollary 6.5. *$\mathtt{UE}/\omega(n \log n) \not\subset C$ (resp. $\mathtt{USUBE}/\omega(n \log n) \not\subset C$), if $\mathtt{TAUT}_C \in \mathtt{USUBE}/b$ (resp. $\mathtt{TAUT}_C \in \mathtt{USUBEXP}/b$) and $\mathtt{CAN}_{(C,p)} \in \mathtt{USUBE}/(1-b)$ (resp. $\mathtt{CAN}_{(C,p)} \in \mathtt{USUBEXP}/(1-b)$), for any $b \in \{0,1\}$ and polynomial $p(n) \geq n$.*

6.3 Lower bound from fast learning algorithms

The two commonly studied learning models are: the Angluin's exact learning model [4], and the Valiant's PAC model [70]. Fast learning algorithms in these models have been known to yield lower bounds [24, 29, 45, 57, 58]. Before giving our results, we first formally define UTIME exact learning.

Exact UTIME learning with membership and equivalence queries: Let s be the size of the target concept C (the circuit to be learned). A $\mathtt{UTIME}(t)$ algorithm is called $\mathtt{Learn}_{(s,C,p)}$, if for any s-size C circuit C, it outputs a circuit C' of size at most $p(s)$ in time at most $t(s)$ with $tt(C) = tt(C')$, on exactly one of its non-deterministic branches, and rejects all the other branches. The algorithm is allowed to make "membership" and "equivalence" queries. A membership query is: "What is the value of $C(x)$?". An equivalence query is: "Is the current hypothesis (H) equal to C?". On any positive equivalence query, it halts and outputs the current hypothesis. On any negative query, it gets x from the oracle, such that $H(x) \neq C(x)$. If the output, and the equivalence queries are all C circuits, the algorithm is called $\mathtt{P\text{-}Learn}_{(s,C,p)}$ (proper learning). Here again, we omit the size parameter when $s(n) = poly(n)$, and the circuit class when C = Boolean. Here we omit $p(n)$ too, if it is $poly(n)$. Unlike in $\mathtt{CAN}_{(C,p)}$, in $\mathtt{Learn}_{(C,p)}$ p decides the size of the output (and not the input).

We extend the result of [24] for this exact UTIME learning. The proof is along the same lines except: we use the SAT and TAUT algorithms for solving the equivalence queries, and we use them in a clever order to get the result of the query in an unambiguous fashion.

Theorem 6.6. $\mathtt{UEXP}/n^{\delta} \not\subset C$ *for any* $\delta > 0$, *if* $\mathtt{SAT}, \mathtt{TAUT}$, *and* $\mathtt{Learn}_C$ *belong to* $\mathtt{USUBEXP}$.

Proof. Fix a $\delta > 0$. Then, for $\epsilon < \delta' < \delta$, there exists an $a < n^{\delta}$, such that $\mathtt{UEXP}/a \not\subset \mathtt{UTIME}(2^{n^{\delta'}})/(a+1)$ (Theorem 6.1). We contradict this, using the assumption $\mathtt{UEXP}/a \subset C$. $\mathtt{UEXP}/a \subset C$ implies that $\mathtt{UEXP}/a = \mathtt{P}^{\#\mathtt{P}}/a$ (using $\mathtt{UTIME}$ $\mathtt{KLT}$). For $L \in \mathtt{UEXP}/a$ we have a polynomial time algorithm for L that uses a amount of advice and makes oracle queries to $\mathtt{Permanent}$. Since $\mathtt{P}^{\#\mathtt{P}}/a \subset C$, $\mathtt{Permanent}$ has n^c-size C circuits, for some constant c. $\mathtt{Permanent}$ is the problem, where input is a matrix M, and output is the permanent of M.

Using $\mathtt{UTIME}(2^{n^{\epsilon}})$ learning algorithm for n^c-size C circuits, we learn and compute $\mathtt{Permanent}$ in $\mathtt{UTIME}(2^{n^{\delta'}})$. This will give a $\mathtt{UTIME}(2^{n^{\delta'}})/a$ algorithm for L.

Algorithm for computing $\mathtt{Permanent}$ *on input* x*:* For $i = 1$ to $|x|$, let c_i be a circuit that computes permanent on $i \times i$ matrix. We will inductively compute c_i for all $|x| = n$ values of i. Then we will compute $c_n(x)$ to get the final result. For $i = 1$ to $|x|$, do the following:

(1) If $i = 1$, let c_i be the trivial circuit (that outputs the input bit itself).
(2) Else, run the learning algorithm for c_i and simulate the queries in the following way:
   (a) *Membership queries:* For any query y of length i, using downward self-reducibility of permanent we can get the answer by making i queries to the circuit c_{i-1}.
   (b) *Equivalence queries:* Let's assume that our current hypothesis is h. We want to know "Does there exists an input z such that $h(z) \neq c_i(z)$?". This is an $\mathtt{NP}$ query as we can compute $c_i(z)$ in polynomial time using c_{i-1}. Convert this query and its compliment to $\mathtt{SAT}$ and $\mathtt{TAUT}$ instances of $poly(n)$ size. Guess a non-deterministic bit z. If $z = 0$, run the $\mathtt{UTIME}$ algorithm for $\mathtt{TAUT}$, and in the case of acceptance output h as the c_i circuit. If $z = 1$, run the $\mathtt{UTIME}$ algorithm for $\mathtt{SAT}$. If it accepts, then we actually need to give a certificate z such that $h(z) \neq c_i(z)$. Now we make two new $\mathtt{NP}$ queries (search to decision): "Does there exists an input z starting with b such that $h(z) \neq c_i(z)$?", one for $b = 0$, and one for $b = 1$. Guess answers to both the queries. Create two $poly(n)$ size $\mathtt{SAT}$ instances, and two $poly(n)$ size $\mathtt{TAUT}$ instances, by reducing these queries and their compliments. Run the $\mathtt{UTIME}$ algorithms on theses queries to verify the two guesses. Note that, at least one guess has to have a positive answer. Repeat this procedure again after fixing the first bit of z unambiguously (fix it to 0 if possible, else fix it to 1). This way we get a $\mathtt{UTIME}$ algorithm for the original equivalence query.

This algorithm puts $L \in \mathtt{UTIME}(2^{n^{\delta'}})$ as the $\mathtt{UTIME}(2^{n^{\epsilon}})$ learning algorithm is used $poly(n)$ times (once for each c_i), and each time it makes $O(2^{n^{\epsilon}})$ $\mathtt{SAT}$ and $\mathtt{TAUT}$ queries, each of which can be computed in $\mathtt{UTIME}(2^{n^{\epsilon}})$. □

6.4 Generalization of lower bound frameworks

In the above sections we saw that fast $\mathtt{UTIME}$ algorithms for certain C circuit related problems ($\mathtt{CAN}$, $\mathtt{TAUT}$, $\mathtt{SAT}$, $\mathtt{Learn}$), were fed to certain frameworks to yield lower bounds for $\mathtt{UTIME}$ against C. Consider the scenario where: a framework is altogether different, or is a fine-grained version of one of the current ones, and works for Boolean circuits, but not for some restriction C. Also consider that, the assumptions of these frameworks are satisfied for that C, but not for unrestricted Boolean circuits. Do we get any lower bounds? In this section we prove that this question has a positive answer.

We use a win-win type argument. We show that, either $\mathtt{P} \not\subset C$ (i.e., a stronger lower bound exists against C), or fast algorithms for C circuits imply fast algorithms for Boolean circuits (i.e., frameworks that only work for Boolean circuits can now be used). To prove our results, we use the Lemma 5.1.

In [74], assumed fast algorithms were applied on witness circuits. To extend their framework, they used the Lemma 5.1 to show, "either $\mathsf{P} \not\subset C$, or the Boolean witnesses have equivalent C circuits, and thus fast algorithms for C are sufficient". Note that, unlike our result, that approach was local to that particular framework.

Theorem 6.7. *Either* $\mathsf{P} \not\subset C$, *or* $\exists c\ \forall k$:

(1) $\mathsf{CAN}_{(C,n^k)} \in \mathsf{UTIME}(t(n)) \implies \mathsf{CAN}_{n^{ck}} \in \mathsf{UTIME}(t(n))$
(2) $\mathsf{CAN}_{(C,n^k)} \in \mathsf{UTIME}(t(n)) \wedge \mathsf{TAUT}_C \in \mathsf{UTIME}(t'(n)) \implies \mathsf{TAUT} \in \mathsf{UTIME}((t(n^{ck}) + t'(n^{ck}))n)$.
(3) $\mathsf{CAN}_{(C,n^k)} \in \mathsf{UTIME}(t(n)) \wedge \mathsf{TAUT}_C \in \mathsf{UTIME}(t'(n)) \wedge \mathsf{SAT}_C \in \mathsf{UTIME}(t''(n))$
$\implies \mathsf{SAT} \in \mathsf{UTIME}((t(n^{ck}) + t'(n^{ck}))n + t''(n^{ck}))$
(4) $\mathsf{P\text{-}Learn}_{(C,n^k)} \in \mathsf{UTIME}(t(n)) \wedge \mathsf{TAUT}_C \in \mathsf{UTIME}(t'(n)) \wedge \mathsf{SAT}_C \in \mathsf{UTIME}(t''(n))$
$\implies \mathsf{CAN}_{(C,n^k)} \in \mathsf{UTIME}(t(n)(t'(n^k) + t''(n^k))n)$

Proof. If $\mathsf{P} \subset C$, from the Lemma 5.1 we know there exists a constant c such that: for each s-size Boolean circuit B, there is an equivalent s^c-size C circuit C (for large enough n).

Proof of 1: By a simple modification of an algorithm $\mathcal{A}$ for $\mathsf{CAN}_{(C,n^k)}$, we obtain an algorithm $\mathcal{A}'$ for $\mathsf{CAN}_{n^{ck}}$. On input B, the algorithm $\mathcal{A}'$ first checks whether B belongs to C. It rejects if the answer is negative. If the answer is positive it simulates $\mathcal{A}$ on B and accepts if and only if $\mathcal{A}$ accepts.

Proof of 2: Let $\mathcal{A}$ be a $\mathsf{UTIME}(t)$ algorithm for $\mathsf{CAN}_{(C,n^k)}$, $\mathcal{A}'$ be a $\mathsf{UTIME}(t')$ algorithm for TAUT_C. Using $\mathcal{A}$ and $\mathcal{A}'$, we construct a UTIME algorithm $\mathcal{A}''$ for TAUT.

For input B to $\mathcal{A}''$, for each gate g of B, let B_g be the circuit corresponding to the output wire of gate g. For the output gate o, $\mathcal{A}''$ first guesses an equivalent C circuit C'_o. To make sure that its guess is unambiguous, it simulates $\mathcal{A}$ on C'_o and rejects if $\mathcal{A}$ rejects. Then it simulates $\mathcal{A}'$ on C'_o (to check if C'_o is a tautology) and rejects if it rejects. The only thing left to check is that C'_o is actually equivalent to C_o.

For checking the consistency of C'_o, $\mathcal{A}''$ first guesses C circuit C'_g, for each gate g. It then simulates $\mathcal{A}$ on each C'_g and rejects if $\mathcal{A}$ rejects on any of them. Finally it simulates $\mathcal{A}'$ on C''_g for each g, where C''_g is the circuit that captures the tautology "$C'_g = op(C'_{g_1}, \ldots, C'_{g_l})$" for $g = op(g_1, \ldots, g_l)$. It accepts if and only if $\mathcal{A}$ accepts on all of them.

Proof of 3: For input B, with the same strategy as in the proof of 2, we first unambiguously construct an equivalent C circuit C. Then, on this C we simulate a $\mathsf{UTIME}(t'')$ algorithm for SAT_C.

Proof of 4: In an exact proper learning algorithm, if we have access to the circuit that we are learning, then we can get a canonization algorithm for C (because the learning algorithm only cares about the truth-table of the circuit that it is learning, and outputs the same hypothesis for all the circuits that have same truth-tables). The membership queries can be handled directly since we have the circuit with us. For the equivalence queries, in the proof of Theorem 6.6 we saw that we need TAUT and SAT algorithms. Since we have the circuit with us, and the hypothesis belongs to C, these queries can be converted into TAUT_C and SAT_C queries. So we get a UTIME algorithm for $\mathsf{CAN}_{(C,n^k)}$. □

The point (4) of Theorem 6.7 shows that canonization is implied by proper learning, tautology and satisfiability algorithms. Using that and the Corollary 6.5, we can get the following new corollary.

Corollary 6.8. $\mathsf{USUBE}/\omega(n \log n) \not\subset C$ *if* TAUT_C, SAT_C, *and* $\mathsf{P\text{-}Learn}_C$ *belong to* $\mathsf{USUBEXP}$.

Note that, one can get a similar corollary from the framework of Theorem 6.6, by making some modifications to it. But our translations use an existing framework (from the Corollary 6.5), and yield a better result.

7 GENERAL CIRCUIT CLASSES: KLT & LOWER BOUNDS

For $i \geq 0$, we use $\Delta_i(s)$ to represent any circuit class from the set $\{\mathsf{SV}\Sigma_i\mathsf{SIZE}(s), \Sigma_i\mathsf{SIZE}(s), \Pi_i\mathsf{SIZE}(s), \mathsf{SIZE}_{||}^{\Sigma_i}(s)\}$. We do this because all the results in this, and the next section, treat these circuit classes equally. We first give this collapse theorem for these classes in the Section 7.1. We show how any non-adaptive Σ_i-oracle circuit family deciding a non-adaptively random-self reducible language has an equivalent $\mathsf{SV}\Sigma_i$ circuit family. Using this equivalence, and generalized Arthur-Merlin classes, we establish high-end KLTs for variety of general circuit classes in the Section 7.2. We use these KLTs and the equivalence relation to give a wide spectrum of fix-polynomial circuit lower bounds in the Section 7.3. In the Section 7.4 we discuss super-polynomial lower bounds.

7.1 General downward collapse theorem

We first formally define the term: non-adaptively random-self-reducible (na-RSR). Then, using the framework from [22] we give our general collapse theorem for na-RSR languages. This framework was also used in [64] in similar manner, to give a collapse theorem for complexity classes with multi-linear extensions (ml-Ext). This result is more general in two ways: (i) any class that supports ml-Ext also supports na-RSR [11, 50], (ii) this collapse also works for non-adaptive Σ_i-oracle circuits, and not just NP oracles. It was also noted in [23], that the collapse theorem of [64] works for PSPACE and $\mathsf{P}^{\#\mathsf{P}}$.

Definition 7.1. A function f is $k(n)$-na-RSR if there are two $k(n)$-time computable functions σ and ϕ that satisfies:

(1) $\forall n \in \mathbb{N}\ \forall x \in \{0,1\}^n,\ Pr_{r\in\{0,1\}^{k(n)}}[f(x) = \phi(x, r, f(\sigma(1, x, r)), \ldots, f(\sigma(k(n), x, r)))] \geq 2/3$;
(2) $\forall n \in \mathbb{N}\ \forall \{x_1, x_2\} \subset \{0,1\}^n,\ \forall i \in [1, k(n)]:\ \sigma(i, x_1, r)\ \&\ \sigma(i, x_2, r)$ are identical distributions over $\{0,1\}^n$.

Theorem 7.2. *The following is true for integer $i \geq 1$ and poly-na-RSR language L:*

$$\exists d\ L \in \mathsf{SIZE}_{||}^{\Sigma_i}(s(n)) \implies L \in \mathsf{SV}\Sigma_i\mathsf{SIZE}(s(n)^d)$$

Proof. L is n^c-na-RSR for some constant c. For input length n, let (C_{pre}, C_{post}) be a $\mathsf{SIZE}_{||}^{\Sigma_i}(s(n))$ circuit that decides L. Let σ and ϕ be the two n^c-time computable functions for L from the definition of non-adaptive random self-reduciblity (see Definition 7.1).

Let $\{r_1, \ldots, r_t\}$ be a set of uniformly random (independent from each other) strings, where size of each r_j for $j \in [1, t]$ is n^c. We decide the value of t later.

Idea: The Σ_i circuit we construct, does the following on any n-length input x:

(1) for $j \in [1, t]$, for $i \in [1, n^c]$, computes the values $f(\sigma(i, x, r_j))$ using (C_{pre}, C_{post});
(it non-deterministically guesses which $\Sigma_i\mathsf{SAT}$ queries in C_{pre}'s output are positive, verifies all of them, and feed them to C_{post})
(2) for $j \in [1, t]$, computes ϕ on (x, r_j);
(3) finally outputs the majority of all these ϕ computations.

We define a $(t \times n^c \times s(n))$ 3-D Boolean matrix M that encapsulates the above computation: for $l \in [1, s(n)]$, $M(j, i, l)$ contains the satisfiability of the l^{th} $\Sigma_i\mathsf{SAT}$ instance that C_{pre} outputs on the input $\sigma(i, x, r_j)$.

Hard-coding the randomness: Each of the t 2-D matrix $M(j, *, *)$ represents the computation of ϕ on (x, r_j). The circuit also has hard-coded in it, the expected number of 1s for any such 2-D matrix. Let this value be exp. It uses it in step (1) to know the number of queries that are positive. The circuit outputs correct if, for majority of these 2-D matrices, it's able to guess all the 1s correctly and the value of ϕ is correct too (i.e., correctly indicates if $x \in L$ or not).

Since $\{r_1, \ldots, r_t\}$ are independent from each other, the number of 1s in the entire 3-D matrix M is concentrated in the range $(t \times exp - \sqrt{t \times exp}, t \times exp + \sqrt{t \times exp})$ (using any appropriate concentration bound) for any n-size input x (see condition (2) of Definition 7.1). We can hard-code $\{r_1, \ldots, r_t\}$ in such a way that, when our Σ_i circuit guesses $t \times exp - \sqrt{t \times exp}$ many $\Sigma_i\mathsf{SAT}$ queries in step (1) to be positive, all the 2-D matrices except at most $2 \times \sqrt{t \times exp}$, will be guessed correctly (entirely). Setting $t = s(n)^{d/3}$ for a large enough constant d, we make sure that $t >> 2 \times \sqrt{t \times exp}$.

Moreover, there is also a possible hard-coding that makes sure that most of these correctly guessed 2-D matrices yield the correct answer for all n-size inputs (by repeating the computation of ϕ and reducing the failure probability in condition (1) of Definition 7.1). Thus our Σ_i circuit outputs correctly on all n-size inputs, and is of size $s(n)^{d/3} \times n^c \times s(n) \times s(n) \times s(n)^{d/3+c'}$ for some constant c'. This is bounded by $s(n)^d$ for appropriate choice of d. □

The Theorem 7.2 yields the following corollary.

Corollary 7.3. *For* $\Gamma \in \{\oplus\mathsf{P}, \mathsf{P}^{\mathsf{PP}}, \mathsf{P}^{\#\mathsf{P}}, \mathsf{PSPACE}, \mathsf{EXP}, \mathsf{ZUEXP}, \mathsf{UEXP}, \Sigma_i\mathsf{EXP} \cap \Pi_i\mathsf{EXP}, \Sigma_i\mathsf{EXP}, \mathsf{E}_{||}^{\Sigma_i}, \mathsf{E}^{\Sigma_i}\}$ *for integer* $i \geq 1$*:* $(\Gamma \subset \mathsf{P}_{||}^{\Sigma_j}/poly \iff \Gamma \subset \Sigma_j/poly \cap \Pi_j/poly)$ *for integer* $j \geq 1$.

Trevisan and Vadhan [68] constructed a PSPACE-complete language L^{TV} that was polynomially random self-reducible. The language was collection of multivariate polynomials of polynomial degree over a field of size 2^n, and thus the self-reducibility they get is non-adaptive. This proves the corollary for PSPACE. For similar reasons it follow for $\oplus\mathsf{P}$ [30] and $\mathsf{P}^{\mathsf{PP}} = \mathsf{P}^{\#\mathsf{P}}$ [22, 23]. The result for EXP follows from the fact that $\mathsf{EXP} \subset \mathsf{PH}/poly \implies \mathsf{EXP} = \mathsf{PSPACE}$. For ZUEXP and UEXP it follows from the fact that the ZUTIME and UTIME EWLs (Theorems 3.4 and 3.3) also work for general circuit classes. For higher classes, except Σ_iEXP, it follows from the fact that they support multi-linear extensions. For Σ_iEXP it follows because of $\mathsf{EXP}_{||}^{\Sigma_i} \subset \Sigma_i\mathsf{EXP}/poly$ (attributed to Buhrman in [23]).

7.2 General High-end KLT

Before we give the general high-end KLT, we prove the following containment to increase the expressiblility of our results.

Lemma 7.4. *For integer* $i \geq 1$*:* $\mathsf{MA}^{\Sigma_i \cap \Pi_i}/a \subseteq \mathsf{AM}^{\Sigma_i \cap \Pi_i}/a = \mathsf{AM}^{\Sigma_{i-1}}/a = \mathsf{AM}_i/a$

Proof. The first inclusion follows from the fact that $\mathsf{MA} \subseteq \mathsf{MAM} = \mathsf{AM}$ relativizes. The last equality follows from the definitions.

Let L be a language with AM protocol, where Arthur has access to some oracle $O \in \Sigma_i \cap \Pi_i$ and a amount of advice. We give an AM protocol for L, where Arthur has access to some oracle $O' \in \Pi_{i-1}$ and same a amount of advice (as in the original AM protocol). Along with the reply to Arthur's query, Merlin sends the answers to the queries Arthur would make to the oracle O in the original protocol. Merlin also sends the certificates corresponding to the first quantifiers of some Σ_i predicates for O and $\overline{O}$. Due to these certificates, Arthur only needs an oracle O' for any Π_{i-1}-complete language, to verify Merlin's reply. When required, Arthur uses the advice it was using in the original protocol. □

We now give our general high-end KLT for PSPACE and some higher classes. We convert IP protocols for PSPACE languages into variety of Arthur-Merlin protocols. We use the general downward collapse theorem (Corollary 7.3) and the generalized Arthur-Merlin protocols to improve/generalize the previously known results. One can also show similar KLTs for the lower classes $\oplus\mathsf{P}, \mathsf{P}^{\mathsf{PP}}, \mathsf{P}^{\#\mathsf{P}}$, using their special complete languages. But we only need the PSPACE or EXP KLTs for our results in the future sections, so we stick to PSPACE and higher classes to keep the proofs simple.

Theorem 7.5. *For integer* $i \geq 1$ *and* $\Gamma \in \{\mathsf{PSPACE}, \mathsf{ZUEXP}, \mathsf{UEXP}, \mathsf{EXP}, \mathsf{EXP}^{\mathsf{NP}}\}$:

(1) $\Gamma \subset (\Sigma_i \cap \Pi_i)/poly \implies \Gamma = \mathsf{MA}^{\Sigma_i \cap \Pi_i}$
(2) $\Gamma \subset (\Sigma_i \cap \Pi_i)/poly \implies \Gamma = \mathsf{AM}_i$
(3) $\Gamma \subset \Delta_i(poly) \implies \Gamma = \mathsf{MA}_{i+1}$
(4) $\Gamma \subset \mathsf{P}^{\Sigma_i}/poly \implies \Gamma = \mathsf{MA}^{\Sigma_i}$

Proof. We only prove the results for PSPACE, for all the other classes it follows: $\Gamma \subset \mathsf{PH}/poly \implies \Gamma = \mathsf{PSPACE}$ follows from [43] for EXP, from [15] for $\mathsf{EXP}^{\mathsf{NP}}$, from ZUTIME and UTIME KLTs (Theorems 3.4 and 3.3) for ZUEXP and UEXP.

For any $L \in \mathsf{PSPACE}$, any honest prover P for L with an unique strategy,

$$L_P = \{(x, y, b) \mid x \in L \text{ and } (y, b) \text{ is a prefix of an accepting strategy/transcript of prover } P\}$$

is also in PSPACE. So $\mathsf{PSPACE} \subset \Lambda$ implies a Λ algorithm for L_P as well. We use this algorithm to design protocols for L.

Proof of (2): It follows from (1) and the Lemma 7.4.

Proof of (1): Here $\Lambda = (\Sigma_i \cap \Pi_i)/poly$. There are Σ_i and Π_i algorithms, $\mathcal{A}$ and $\mathcal{A}'$ respectively, that accept complementary sets on any advice. Moreover, $\mathcal{A}$ accepts L_P on any correct advice. Including the advice into the input parameter, the modified language corresponding to $\mathcal{A}$ is a $\Sigma_i \cap \Pi_i$ language. The verifier Arthur will have oracle access to this modified language.

Merlin sends the advice adv to Arthur. Arthur guesses its random bits $R = \{r_1, r_2, \ldots, r_m\}$. At each iteration $j \in [1, m]$, it creates two inputs for L_P: $(x, r_1 n_1 \ldots r_j, 0)$ and $(x, r_1 n_1 \ldots r_j, 1)$, where $n_1, \ldots, n_{j-1}$ are the results from the previous iteration. If the oracle returns complementary results, then Arthur fixes n_j to be the last bit of the query that results in a positive answer, else it rejects (by a standard padding argument we can make the length of all the queries equal, so that a single advice from Merlin suffices). This way Arthur fills the transcript (and simultaneously checks whether the strategy encoded by $\mathcal{A}/adv$ is unique, where it matters). It finally accepts if the transcript is an accepting one, else it rejects.

Completeness: If $x \in L$, then Merlin can send the correct advice. The oracle then helps in generating accepting transcripts on high fraction of the random bits, and thus Arthur accepts with high probability.

Soundness: If $x \notin L$, and if Merlin sends an advice that makes the strategy unique (encoded by $\mathcal{A}$ on that advice) on high fraction of random bits, then Arthur must reject with high probability (due to the soundness of the original IP).

Proof of (3): Due to Corollary 7.3, here we only need to consider the case $\Lambda = \Sigma_i/poly$. There is an Σ_i algorithm $\mathcal{A}$ that accepts L_P given the correct advice. Including the advice into the input parameter, the modified language corresponding to $\mathcal{A}$ is an Σ_i language. If we include the first existential quantifier into the input parameter as well, then the modified language belongs to Π_{i-1}. Let's call this language $\mathcal{B}$. The verifier Arthur will have oracle access to $\mathcal{B}$.

Merlin sends the advice adv to two verifiers, Arthur and Henry (who are not allowed to communicate as per the definition of AM_{i+1}).

$\mathsf{AM}_i = \mathsf{AM}^{\Pi_{i-1}}$ *part:* Arthur first guesses its random bits $R = \{r_1, r_2, \ldots, r_m\}$. Merlin then sends the non-deterministic replies $N = \{n_1, n_2, \ldots, n_m\}$. Merlin also sends, for all j, positive certificates (corresponding to the first existential quantifier) for inputs $(x, r_1 n_1 \ldots r_j, n_j)$ of $\mathcal{A}/adv$. Then for all j, Arthur uses its oracle $\mathcal{B}$ to check if $\mathcal{A}/adv$ accepts $(x, r_1 n_1 \ldots r_j, n_j)$. Finally, Arthur checks whether the transcript $(x, r_1 n_1 \ldots r_m, n_m)$ is an accepting one.

Henry's Π_i part: Henry checks $\neg\exists(x, y)[\mathcal{A}(x, y, 0)/adv = \mathcal{A}(x, y, 1)/adv = 1]$, to confirm that $\mathcal{A}/adv$ encodes L_P for some honest prover P with an unique strategy (for the input length under consideration).

Completeness: If $x \in L$, then Merlin can send the correct advice corresponding to L_P. This advice passes both, Arthur's test (with high probability) and Henry's test.

Soundness: If $x \notin L$, then either Henry's test fails, or the unique strategy encoded by $\mathcal{A}/adv$ is rejected with high probability by Arthur (due to the soundness of the original IP).

Proof of (4): Here $\Lambda = \mathsf{P}^{\Sigma_i}/poly$. There is a P^{Σ_i} algorithm $\mathcal{A}$ that accepts L_P given the correct advice. Including the advice into the input parameter, the modified language corresponding to $\mathcal{A}$ is a P^{Σ_i} language. The verifier Arthur will have oracle access to this modified language. Now, the roles of Merlin and Arthur, and the arguments of completeness and soundness of the protocol, are similar to those given in the proof of (1). □

7.3 General fix-polynomial lower bounds

Santhanam [63] modified L^{TV} to give L^S which satisfies the following lemma, which was a crucial technical step in his celebrated MA lower-bound.

Lemma 7.6. *There is a* PSPACE*-complete language L^S and probabilistic polynomial-time oracle Turing machines M and M' such that the following holds for any n-length input x:*

(1) *M and M' only query their oracle on strings of length n.*
(2) *If M (resp. M') is given L^S as its oracle and $x \in L^S$ (resp. $x \notin L^S$), then M (resp. M') accepts with probability 1.*
(3) *If $x \notin L^S$ (resp. $x \in L^S$), then irrespective of the oracle, M (resp. M') rejects with probability at least 2/3.*

The modified language L^S retained the poly-na-RSR property of L^{TV}. We use this very fact, combined with the general downward collapse theorem (Theorem 7.2), to yield fix-polynomial lower bounds against non-adapative Σ_i-oracle circuits.

Our proof of the lower bounds for general Arthur-Merlin protocols (with 1 bit of advice) against fix-polynomial Λ circuits is split into two cases: (i) The easier case where PSPACE has poly-size Λ circuits we use the general high-end KLT (Theorem 7.5). (ii) The difficult case where PSPACE doesn't have poly-size Λ circuits, we design protocols for a padded version of L^S that doesn't have fix-polynomial Λ circuits. The protocols start with Merlin sending Λ circuits for L^S.

For the case of non-adaptive Σ_i-oracle circuits, we use the padded version of L^S that doesn't have fix-polynomial size non-adaptive Σ_i-oracle circuits. But to design the desired protocol, we need to start with Merlin sending Σ_i circuits instead. Here we use the following fact derived as an result of our general downward collapse theorem (Theorem 7.2): there is a constant c such that, any $\mathtt{SIZE}^{\Sigma_i}_{||}(s(n))$ circuit sequence for L^S has an equivalent $\Sigma_i\mathtt{SIZE}(s(n)^c)$ circuit sequence.

We first prove an auxiliary lemma that we use for the second (difficult) case.

Lemma 7.7. *For $k \geq 1$ and non-promise circuit class Λ, using L^S from Lemma* **??** *we define:*

$$L^k_\Lambda = \{x1^y \mid x \in L^S \wedge \exists(z \in \mathbb{N})\ y = 2^z \geq |x| > 0, (2y + |x|)^{k+1} \geq ckt_\Lambda(L^S_{|x|}) > (y + |x|)^{k+1}\}.$$

If PSPACE *doesn't have poly-size Λ circuit sequence, then L^k_Λ doesn't have n^k-size Λ circuit sequence.*

Proof. For the sake of contradiction, let's assume that there is a sequence of n^k-size Λ circuits $\{C_n\}_{n\in\mathbb{N}}$ that decides L^k_Λ. We modify this sequence to yield a sequence for L^S (used in the definition of L^k_Λ). Any input length n can be broken into unique n_1 and $y = 2^z$ such that $y \geq n_1$ and $n_1 + y = n$. If y satisfies $(2y + n_1)^{k+1} \geq ckt_\Lambda(L^S_{n_1}) \geq (y + n_1)^{k+1}$, then a circuit for the n^{th}-slice of L^k_Λ can be used to yield a circuit for the n_1^{th}-slice of L^S (by fixing the last y input bits to 1). Moreover for any n_1, there is a unique y that satisfies $(2y + n_1)^{k+1} \geq ckt_\Lambda(L^S_{n_1}) \geq (y + n_1)^{k+1}$ (since y is a power of 2). So for any input length n_1, we get an n^k-size circuit for L^S. This leads to the contradiction

$n^k \geq ckt_\Lambda(L^S_{n_1}) > (y+n_1)^{k+1} = n^{k+1}$ infinitely often. The first inequality follows from the definition of the measure ckt_Λ, and second inequality follows from the definition of L^k_Λ. The assumption that PSPACE doesn't have poly-size Λ circuit sequence is essential for this contradiction: for each $k \geq 1$, $ckt_\Lambda(L^S_{n_1}) > (2n_1)^{k+1}$ holds infinitely often, and thus infinitely many (y, n_1) valid pairs exists that satisfy $y \geq n_1$. □

For the special case of promise $\mathsf{SV}\Sigma_i$ circuits we prove a separate auxiliary lemma. The proof of this lemma: (i) either deals with each of the underlying promise algorithms $\mathcal{A}$ separately using the measure $ckt_{\mathsf{prSV}_i(\mathcal{A})}$; (ii) or combine them using the stronger measure ckt_{prSV_i} (see Section 2).

Lemma 7.8. *Let $\mathcal{A}$ be a prSV_i algorithm, then for $k \geq 1$, using L^S from Lemma ?? we define:*

(1) $L^k_{\mathcal{A}} = \{x1^y \mid x \in L^S \wedge \exists (z \in \mathbb{N})\ y = 2^z \geq |x| > 0, (2y+|x|)^{k+1} \geq ckt_{\mathsf{SV}_i(\mathcal{A})}(L^S_{|x|}) > (y+|x|)^{k+1}\}$

(2) $L^k_{\mathsf{prSV}_i} = \{x1^y \mid x \in L^S \wedge \exists (z \in \mathbb{N})\ y = 2^z \geq |x| > 0, (2y+|x|)^{k+1} \geq ckt_{\mathsf{prSV}_i}(L^S_{|x|}) > (y+|x|)^{k+1}\}$

If $\mathsf{PSPACE} \not\subset (\Sigma_i \cap \Pi_i)/poly$, *then for all* $k \geq 1$, $L^k_{\mathcal{A}} \notin \mathsf{prSV}\Sigma_i^{\mathcal{A}}\mathsf{SIZE}(n^k)$ *and* $L^k_{\mathsf{prSV}_i} \notin \mathsf{prSV}\Sigma_i\mathsf{SIZE}(n^k)$.

Proof. *Proof of (1):* It is the same as the proof of the Lemma 7.7: any n^k-size $\mathsf{SV}\Sigma_i$ circuit sequence that is produced by $\mathcal{A}$, and that decides $L^k_{\mathcal{A}}$, can be modified to give a circuit sequence for L^S that violates the bounds in the definition of $L^k_{\mathcal{A}}$.

Proof of (2): This proof too is same except few changes: for the sake of contradiction, let's assume that $L^k_{\mathsf{prSV}_i} \in \mathsf{prSV}\Sigma_i(n^k)$. That means, there is a prSV_i algorithm $\mathcal{A}$, that produces an n^k-size $\mathsf{SV}\Sigma_i$ circuit sequence, that decides $L^k_{\mathsf{prSV}_i}$. We modify this sequence as in the proof of Lemma 7.7 to give an $\mathsf{SV}\Sigma_i$ circuit sequence, that decides L^S. For any input length n_1, the size of the circuit from this sequence will be $(n_1+y)^k$ for the unique y that is paired with n_1. This leads to the contradiction $(n_1+y)^k \geq ckt_{\mathsf{prSV}_i(\mathcal{A})}(L^S_{n_1}) \geq ckt_{\mathsf{prSV}_i}(L^S_{n_1}) > (n_1+y)^{k+1}$ infinitely often. The first inequality follows from the fact that the circuit sequence is produced by $\mathcal{A}$. The second inequality uses the fact that the stronger measure ckt_{prSV_i}, beats the measure $ckt_{\mathsf{prSV}_i(\mathcal{A})}$ for any prSV_i algorithm $\mathcal{A}$, after a certain input length (because $\mathcal{A}$'s description is only of constant length, i.e. less than $\log n_1$). □

Now we prove one of the two main results of this section. We prove a variety of lower bounds for a collection of special Arthur-Merlin protocols that use 1 bit of advice (except one case, where we need extra $\log n$ amount of advice).

Theorem 7.9. *For integer $i \geq 1$:*

(1) $\forall k\ \mathsf{MA}^{\Sigma_i \cap \Pi_i}/1 \not\subset w\text{-}\mathsf{prSV}\Sigma_i\mathsf{SIZE}(n^k)$

(2) $\forall k\ \mathsf{AM}_i/1 + \log n \not\subset \mathsf{prSV}\Sigma_i\mathsf{SIZE}(n^k)$

(3) $\forall k\ \mathsf{MA}_{i+1}/1 \not\subset \Delta_i(n^k)$

(4) $\forall k\ \mathsf{MA}^{\Sigma_i}/1 \not\subset \mathsf{SIZE}^{\Sigma_i}(n^k)$

Proof. *General idea:* If PSPACE has poly-size Λ circuits, then PSPACE has the desired protocol (Theorem 7.5), and we get the desired fix-polynomial circuit lower-bounds against these protocols (without any advice) because in PSPACE we can diagonalize against any fix-polynomial size general circuit class.

If PSPACE doesn't have poly-size Λ circuits. From the Lemmas 7.7 and 7.8 we get languages with the desired lower bounds. We design the desired protocols for these languages. Arthur rejects everything if the first advice bit is 0. The first advice bit is 1 exactly for the input lengths n that split into valid (n_1, y) pairs (validity is based on the measure used in the definition of the language: see the proofs of the lemmas). Arthur checks if the input is of the format $x1^y$, and then simulates the machine M from the Lemma 7.6 to check if $x \in L^S$ or not. It uses the circuit C, sent by Merlin (or computed from Merlin's reply), as an oracle to M.

Proof of (1): To show $\mathtt{MA}^{\Sigma_i \cap \Pi_i}/1 \not\subset \text{w-}\mathtt{prSV}\Sigma_i\mathtt{SIZE}(n^k)$, for each $\mathtt{prSV}_i$ algorithm $\mathcal{A}$, we need to find one language in $\mathtt{MA}^{\Sigma_i \cap \Pi_i}/1$, that is not in $\mathtt{prSV}\Sigma_i^{\mathcal{A}}\mathtt{SIZE}(n^k)$. From Lemma 7.8 we have $L^k_{\mathcal{A}}$ that is not in $\mathtt{prSV}\Sigma_i^{\mathcal{A}}\mathtt{SIZE}(n^k)$. We give an $\mathtt{MA}^{\Sigma_i \cap \Pi_i}/1$ protocol for $L^k_{\mathcal{A}}$.

For n-length input $x1^y$ with $|x| = n_1$, Merlin sends an $(2y+n_1)^k$-length input z for $\mathcal{A}$. Arthur then generates its random bits and simulates M using the $\mathtt{SV}\Sigma_i$ circuit $\mathcal{A}(z)$ as the oracle. For any oracle query a that M makes, Arthur uses the $\Sigma_i \cap \Pi_i$ oracle $L_{\mathcal{A}}$ defined by: $L_{\mathcal{A}} = \{(z,a) \mid \exists y_1\ Flag_{\mathcal{A}(z)}(a,y_1) \wedge Value_{\mathcal{A}(z)}(a,y_1)\}$ and $\overline{L_{\mathcal{A}}} = \{(z,a) \mid \exists y_1\ Flag_{\mathcal{A}(z)}(a,y_1) \wedge \neg Value_{\mathcal{A}(z)}(a,y_1)\}$, where $Flag_{\mathcal{A}(z)}$ and $Value_{\mathcal{A}(z)}$ both are Π_{i-1} predicates (see the definition of $\mathtt{SV}\Sigma_i$ circuits from Section 2).

Completeness follows easily. If $x \in L^S$, Merlin can send the input on which the algorithm $\mathcal{A}$ outputs the correct $\mathtt{SV}\Sigma_i$ circuit for L^S. If $x \notin L^S$, soundness follows from the fact that the algorithm $\mathcal{A}$ always generates $\mathtt{SV}\Sigma_i$ circuits, and thus the oracle used by M is consistent (to some language), and thus M rejects with probability at least 2/3.

Proof of (2): Its same as (1) except few changes. We give an $\mathtt{MAM}^{\Pi_{i-1}}/1 + \log n$ protocol for the $L^k_{\mathtt{prSV}_i}$ language. The extra $\log n$ bits of advice encodes one of the most efficient $\mathtt{prSV}_i$ algorithms for that input length, i.e. an algorithm $\mathcal{A}$ such that $ckt_{\mathtt{SV}_i(\mathcal{A})}(L^S_{|x|}) = ckt_{\mathtt{prSV}_i}(L^S_{|x|})$. After Merlin sends his reply z, Arthur guesses its random bits R to simulate M. Instead of using an $(\Sigma_i \cap \Pi_i)$ oracle, it sends R to Merlin. For each oracle query a that M would make on R: Merlin replies with the certificate y_1 that, (i) satisfies $(Flag_{\mathcal{A}(z)}(a,y_1) \wedge Value_{\mathcal{A}(z)}(a,y_1))$ for a positive query a, (ii) satisfies $(Flag_{\mathcal{A}(z)}(a,y_1) \wedge \neg Value_{\mathcal{A}(z)}(a,y_1))$ for a negative query a. Arthur uses advice to compute $\mathcal{A}(z)$, and then reduces $Flag_{\mathcal{A}(z)}$ and $Value_{\mathcal{A}(z)}$ to some Π_{i-1}-complete language that it is using as the oracle. Completeness and soundness follow because of similar reasons given in the proof of (1).

Proof of (3): We prove the result for non-adaptive Σ_i-oracle circuits. Similar arguments yield the results for Σ_i, Π_i and $\mathtt{SV}\Sigma_i$ circuit classes.

First we use the Theorem 7.2 to get the implication: $\exists d\ L^S \in \mathtt{SIZE}^{\Sigma_i}_{||}\mathtt{SIZE}(s(n)) \implies L^S \in \mathtt{SV}\Sigma_i\mathtt{SIZE}(s(n)^d)$. Now Merlin sends the $\mathtt{SV}\Sigma_i$ circuit instead of the non-adaptive Σ_i-oracle circuit (which is also of size polynomial in the input, no matter what $s(n)$ is).

Now the "$\mathtt{AM}_i = \mathtt{AM}^{\Pi_{i-1}}$" part, "Henry's Π_i" part, completeness, and soundness are similar to that given in the proof of the point (2) of Theorem 7.5. Merlin sends a Σ_i circuit C to Arthur and Henry. Henry verifiers if C is consistent with some language, and Arthur interacts with Merlin to simulate machine M with oracle C (hoping that C encodes L^S). Due to the interaction with Merlin, Arthur only needs an Π_{i-1} oracle to compute C.

Proof of (4): Here, Merlin sends the Σ_i-oracle circuit C. Arthur generates its random bits, and simulates M as in the above proofs. Whenever M makes an oracle query, Arthur uses C, and for computing the output values of the $\Sigma_i\mathtt{SAT}$ oracle gates of C, it makes oracle queries to the $\Sigma_i\mathtt{SAT}$ oracle. Rest of the proof is same as the above proofs. □

Including the advice into the input, and making the inputs with the correct advice as the promise inputs, we get the following theorem. Also note that, for any class Γ, ip-$(\Gamma \cap \text{co-}\Gamma)$ is a special case of the class $\mathtt{pr}(\Gamma \cap \text{co-}\Gamma)$, which is a special case of $\mathtt{pr}\Gamma \cap \mathtt{prco}\text{-}\Gamma$.

Theorem 7.10. *For integer $i \geq 1$ and constant $k \geq 1$:*

(1) $ip\text{-}(\mathtt{MA}^{\Sigma_i \cap \Pi_i} \cap \text{Co-}\mathtt{MA}^{\Sigma_i \cap \Pi_i}) \not\subset w\text{-}\mathtt{prSV}\Sigma_i\mathtt{SIZE}(n^k)$

(2) $ip\text{-}(\mathtt{AM}_i \cap \text{Co-}\mathtt{AM}_i) \not\subset \mathtt{prSV}\Sigma_i\mathtt{SIZE}(n^k)$

(3) $ip\text{-}(\mathtt{MA}_{i+1} \cap \text{Co-}\mathtt{MA}_{i+1}) \not\subset \Delta_i(n^k)$

(4) $ip\text{-}(\mathtt{MA}^{\Sigma_i} \cap \text{Co-}\mathtt{MA}^{\Sigma_i}) \not\subset \mathtt{SIZE}^{\Sigma_i}(n^k)$

Proof. Here we just give the proof of (2). All the other proofs follow from similar arguments. In the proof of the Theorem 7.9, if we use the machine M' from the Lemma 7.6, we can get a

$\mathsf{Co\text{-}AM}_i/1+\log n$ protocol for the language $L^k_{\mathsf{prSV}_i}$ that uses the same advice that the $\mathsf{AM}_i/1+\log n$ protocol used. This protocol accepts if the input length n doesn't split into a valid n_1 and y pair (using the first advice bit), and if the input is not in the $x1^y$ format. It both these tests are passed by the input, then it accepts if M' accepts x (i.e. $x \notin L^S$). Only change is that Arthur simulates M' instead of M (using the same advice). Let's denote these $\mathsf{AM}_i/1+\log n$ and $\mathsf{Co\text{-}AM}_i/1+\log n$ protocols for $L^k_{\mathsf{prSV}_i}$, by $\mathcal{A}$ and $\mathcal{A}$ respectively.

Now, define a new modified language $T^k_{\mathsf{prSV}_i} = \{g1^h \mid \mathcal{A} \text{ accepts } g \text{ on } h^{th} \text{ advice}\}$. Any input length $n+j$ in the range $[n+1, 3n]$ is dedicated to the simulation of $\mathcal{A}$ on the j^{th} advice (lexicographically j^{th} among all the $1+\log n$ bits long advice strings). Now we create a promise problem prT^k using the language $T^k_{\mathsf{prSV}_i}$, whose promise input lengths are the ones, that correspond to the correct advice for $\mathcal{A}$ and $\mathcal{A}'$.

The protocols $\mathcal{A}$ and $\mathcal{A}'$ decide $T^k_{\mathsf{prSV}_i}$ and $\overline{T^k_{\mathsf{prSV}_i}}$ correctly on the promise inputs of prT^k. $\mathcal{A}$ and $\mathcal{A}'$ also satisfy the semantic promises on these input lengths. Thus $prT^k \subset \text{ip-}(\mathsf{AM}_i \cap \mathsf{Co\text{-}AM}_i)$, and from the same arguments as in the proof of the Lemma 7.8, $prT^k \notin \mathsf{prSV}\Sigma_i\mathsf{SIZE}(n^{k-1})$. □

7.4 General super-polynomial lower bounds

In this section we show super-polynomial lower bounds against the exponential version of the same protocols. We also improve these lower bounds in two ways: (i) extending them to sub-half-exponential circuit sizes; (ii) extending them to super-half-exponential versions of the protocols. Also, for the special case of $\mathsf{MA}^{\Sigma_i\cap\Pi_i}$, unlike the case of fix-polynomial lower bounds, we get strong lower bounds against $\mathsf{prSV}\Sigma_i$ circuits. That is, only one language in $\mathsf{MAE}^{\Sigma_i\cap\Pi_i}$ is enough to beat the circuits produced by each of the prSV_i algorithms (infinitely often).

Theorem 7.11. *[lower bounds against poly-size] For integer $i \geq 1$:*

(1) $\mathsf{MAE}^{\Sigma_i\cap\Pi_i} \not\subset (\Sigma_i \cap \Pi_i)/poly$
(2) $\mathsf{AM}_i\mathsf{E} \not\subset (\Sigma_i \cap \Pi_i)/poly$
(3) $\mathsf{MA}_{i+1}\mathsf{E} \not\subset \Delta_i(poly)$
(4) $\mathsf{MAE}^{\Sigma_i} \not\subset \mathsf{P}^{\Sigma_i}/poly$

Proof. *Proof of (1):* We prove it by contradiction. Suppose $\mathsf{MAE}^{\Sigma_i\cap\Pi_i} \subset (\Sigma_i \cap \Pi_i)/poly$, then by Theorem 7.5 we get $\mathsf{EXP} = \mathsf{MA}^{\Sigma_i\cap\Pi_i}$. By a simple padding argument we get $\mathsf{DTIME}(2^{2^{O(n)}})$ is contained in $\mathsf{MAE}^{\Sigma_i\cap\Pi_i}$, and since in $\mathsf{DTIME}(2^{2^{O(n)}})$ we can diagonalize against any poly-size general circuit class, we get the desired lower bound.

Proof of (2): It follows from (1) and a padded version of the Lemma 7.4.

Proofs of (3) & (4): They are analogous to the proof of (1). □

Theorem 7.12. *[lower bounds against sub-half-exponential size] Let $f : \mathbb{N} \to \mathbb{N}$ be a function that satisfies $f(f(n)^{O(1)}) \in 2^{O(n)}$ then for integer $i \geq 1$:*

(1) $\mathsf{MAE}^{\Sigma_i\cap\Pi_i} \not\subset \mathsf{prSVSIZE}(f(n))$
(2) $\mathsf{AM}_i\mathsf{E} \not\subset \mathsf{prSVSIZE}(f(n))$
(3) $\mathsf{MA}_{i+1}\mathsf{E} \not\subset \Delta_i(f(n))$
(4) $\mathsf{MAE}^{\Sigma_i} \not\subset \mathsf{SIZE}^{\Sigma_i}(f(n))$

Proof. *Proof of (1):* We prove it by contradiction. Suppose $\mathsf{MAE}^{\Sigma_i\cap\Pi_i} \subset \mathsf{prSV}\Sigma_i\mathsf{SIZE}(f(n))$, then $\mathsf{DSPACE}(n) \subset \mathsf{prSV}\Sigma_i\mathsf{SIZE}(f(n))$ and by the same arguments as in the proof of Theorem 7.5 we get $\exists k\ \mathsf{DSPACE}(n) \subset \mathsf{MA}^{\Sigma_i\cap\Pi_i}\mathsf{TIME}(f(n^k)^k)$. We can diagonalize against $\mathsf{prSV}\Sigma_i\mathsf{SIZE}(f(n))$ circuits in the class $\mathsf{DSPACE}(f(n)^2)$, which is a subset of $\mathsf{MA}^{\Sigma\cap\Pi_i}\mathsf{TIME}(f(f(n)^{2k})^k) \subset \mathsf{MAE}^{\Sigma_i\cap\Pi_i}$ due to padding.

Proof of (2): It follows from (1) and a padded version of the Lemma 7.4.

Proofs of (3) & (4): They are analogous to the proof of (1). □

Theorem 7.13. *[lower bounds for super-half-exponential time] Let $f : \mathbb{N} \to \mathbb{N}$ be a function that satisfies $f(f(n)^{\omega(1)}) \in 2^{\Omega(n)}$, then for integer $i \geq 1$:*

(1) $\mathtt{MA}^{\Sigma_i \cap \Pi_i}\mathtt{TIME}(f(n^{\omega(1)})^{O(1)}) \not\subset (\Sigma_i \cap \Pi_i)/poly$
(2) $\mathtt{AM}_i\mathtt{TIME}(f(n^{\omega(1)})^{O(1)}) \not\subset (\Sigma_i \cap \Pi_i)/poly$
(3) $\mathtt{MA}_{i+1}\mathtt{TIME}(f(n^{\omega(1)})^{O(1)}) \not\subset \Delta_i(poly)$
(4) $\mathtt{MA}^{\Sigma_i}\mathtt{TIME}(f(n^{\omega(1)})^{O(1)}) \not\subset \mathtt{P}^{\Sigma_i}/poly$

Proof. *Proof of (1):* We prove it by contradiction. Suppose $\mathtt{MATIME}^{\Sigma_i \cap \Pi_i}(f(n^{\omega(1)}))$ can be decided by $\mathtt{prSV}\Sigma_i\mathtt{SIZE}(poly(n))$ circuits, then by padding we get $\mathtt{MA}^{\Sigma_i \cap \Pi_i}\mathtt{E} \subset \mathtt{prSV}\Sigma_i\mathtt{SIZE}(poly(f(n)))$. Thus $\exists k\ \mathtt{DSPACE}(n) \subset \mathtt{prSV}\Sigma_i\mathtt{SIZE}(f(n)^k)$, and by the same arguments as in the proof of Theorem 7.5 we get $\exists k\ \mathtt{DSPACE}(n) \subset \mathtt{MA}^{\Sigma_i \cap \Pi_i}\mathtt{TIME}(f(n^k)^k)$. Padding gives us that $\mathtt{DSPACE}(n^{\omega(1)})$ is contained in $\mathtt{MA}^{\Sigma_i \cap \Pi_i}\mathtt{TIME}(f(n^{\omega(1)})^k)$, but $\mathtt{DSPACE}(n^{\omega(1)}) \not\subset \mathtt{prSV}\Sigma_i\mathtt{SIZE}(poly(n))$ due to diagonalization.

Proof of (2): It follows from (1) and a padded version of the Lemma 7.4.

Proofs of (3) & (4): They are analogous to the proof of (1). □

8 DERANDOMIZATION VS LOWER BOUNDS

In this section we extend the lower bounds vs derandomization connection to: (i) UEXP and ZUEXP (Section 8.1); and (ii) general circuit classes (Section 8.2). Combining the extension to general circuit classes, with the fix-polynomial lower bounds of Section 7, we show how lower bounds translate to sub-exponential versions of the corresponding exponential time classes (Section 8.3)

8.1 Variety of derandomization from variety of lower bounds

In this section we extend the "lower-bounds to derandomization" frame-work of [75] to get unified results for: (i) the three one-sided error classes: NEXP, REXP and UEXP; and their zero-error versions: ZNEXP, ZREXP, and ZUEXP. We use the "hardness to randomness" connection from Theorem 2.4.

As a first step towards unification, we generalize the connection between "lower bounds for ZRE seeds (*resp.* RE hitting-sets for witnesses) and P-R useful properties (*resp.* P/log n-R useful properties)" [75], to ZNE (*resp.* NE) and ZUE (*resp.* UE). The results for ZNE (*resp.* NE) and ZUE (*resp.* UE) follow from the results in the Table 1.

Lemma 8.1. *For* C = N, R, U*:*

(1) $\mathtt{ZCE} \not\subset_s C \iff \mathtt{P\text{-}C} \not\subset_{tt} C$
(2) $\mathtt{CE} \not\subset_{hw} C \implies \mathtt{P}/\log n\text{-}\mathtt{C} \not\subset_{tt} C$

The original connection was used to show that $\mathtt{REXP} \not\subset \mathtt{SIZE}(poly)$ or $\mathtt{REXP} \neq \mathtt{EXP}$ implies $\mathtt{BPP} \subset \cap_{\epsilon>0}$ io-$\mathtt{ZPTIME}(2^{n^\epsilon})/n^\epsilon$. The lower bounds implied the existence of useful properties (or hard functions), which were used to derandomize BPP. Using Lemma 8.1, we get a variety of properties from a variety of lower bounds, and thus get a variety of derandomization results.

Theorem 8.2. *For* C = N, R, U*:*

(1) $\mathtt{ZCEXP} \neq \mathtt{EXP} \implies \mathtt{BPP} \subset \cap_{\epsilon>0}$ *io*-$\mathtt{ZCTIME}(2^{n^\epsilon})$
(2) $\mathtt{ZCEXP} \not\subset \mathtt{SIZE}(poly) \implies \mathtt{BPP} \subset \cap_{\epsilon>0}$ *io*-$\mathtt{ZCTIME}(2^{n^\epsilon})$
(3) $\mathtt{ZCEXP} \neq \mathtt{MA} \implies \mathtt{BPP} \subset \cap_{\epsilon>0}$ *io*-$\mathtt{ZCTIME}(2^{n^\epsilon})$
(4) $\mathtt{ZCEXP} \neq \mathtt{BPP} \implies \mathtt{BPP} \subset \cap_{\epsilon>0}$ *io-Heur*-$\mathtt{ZCTIME}(2^{n^\epsilon})$
(5) $\mathtt{CEXP} \neq \mathtt{EXP} \implies \mathtt{BPP} \cap_{\epsilon>0} \subset$ *io*-$\mathtt{ZCTIME}(2^{n^\epsilon})/n^\epsilon$
(6) $\mathtt{CEXP} \not\subset \mathtt{SIZE}(poly) \implies \mathtt{BPP} \subset \cap_{\epsilon>0}$ *io*-$\mathtt{ZCTIME}(2^{n^\epsilon})/n^\epsilon$
(7) $\mathtt{CEXP} \neq \mathtt{MA} \implies \mathtt{BPP} \subset \cap_{\epsilon>0}$ *io*-$\mathtt{ZCTIME}(2^{n^\epsilon})/n^\epsilon$
(8) $\mathtt{CEXP} \neq \mathtt{BPP} \implies \mathtt{BPP} \subset \cap_{\epsilon>0}$ *io-Heur*-$\mathtt{ZCTIME}(2^{n^\epsilon})/n^\epsilon$

Proof. *Proof of (1):* Let's assume that ZCEXP ≠ EXP. Then ZCE can't have seeds in SIZE($poly$), because brute-forcing through the seeds will prove ZCEXP = EXP. Thus, there exists a P-C property $\mathcal{P}$ useful against SIZE($poly$) (from the Lemma 8.1). For each c, let S_c be the infinite set of input lengths where $\mathcal{P}$ only accept strings str satisfying $ckt(str) \geq n^c$. These strings are truth-tables of hard functions, and can be computed in CE using the constructivity of $\mathcal{P}$.

For $k, \epsilon > 0, \epsilon > \epsilon'/2$ and $L \in$ BPTIME($n^{k/2}$), set $c = gk/\epsilon'$ (where g is the constant from Theorem 2.4). We give a ZCTIME(2^{n^ϵ}) algorithm for L that works for any input length n with $2^n \in S_c$. For n-length input x of L, let C_x be the SIZE(n^k) circuit capturing the BP computation of L.

For C = N : Non-deterministically guess a string Y of length $m = 2^{n^{\epsilon'}}$. Output '?' if $\mathcal{P}(Y) = 0$. Once we have access to Y with $\mathcal{P}(Y) = 1$ (or $ckt(Y) \geq n^k$), we can construct PRG $G : n^\epsilon \to n^k$ from Y (using the Theorem 2.4) that is computable in E. We brute-force through all the $n^{\epsilon'}$-length seeds, and on each of the output strings of length n^k, compute the circuit C_x to calculate its acceptance probability in time 2^{n^ϵ} (within $\pm 1/n^k$ error). Output 1 if this value is 1/2 or more, else output 0.

For C = U : The same process as above works, because $\mathcal{P}(Y) = 1$ holds for unique Y.

For C = R : Instead of one, non-deterministically guess a collection of strings $\{Y_1, \ldots, Y_c\}$ for some constant c. Output '?' if $\forall i\ \mathcal{P}(Y_i) = 0$. Else proceed with any Y_i with $\mathcal{P}(Y_i) = 1$ in the same manner as above. For large enough c, this is a ZRTIME algorithm: as $\mathcal{P}$ is large, with high probability we find Y_i satisfying $\mathcal{P}(Y_i) = 1$.

Proof of (2): We prove the contrapositive. Assume that $\exists \epsilon > 0$ such that BPP $\not\subset$ io-ZCTIME(2^{n^ϵ}). This gives us EXP $\subset$ SIZE($poly$) [9, 55, 56], and ZCEXP = EXP from (1). Thus, ZCEXP $\subset$ SIZE($poly$).

Proof of (3): Using (1), (2) and EXP KLT we get a series of implications that conclude the proof.

$$\begin{aligned}
\text{ZCEXP} \neq \text{MA} &\implies \text{ZCEXP} \neq \text{EXP or EXP} \neq \text{MA} \\
&\implies \text{BPP} \subset \cap_{\epsilon>0} \text{io-ZCTIME}(2^{n^\epsilon}) \text{ or EXP} \not\subset \text{SIZE}(poly) \\
&\implies \text{BPP} \subset \cap_{\epsilon>0} \text{io-ZCTIME}(2^{n^\epsilon}) \text{ or ZCEXP} \not\subset \text{SIZE}(poly) \\
&\implies \text{BPP} \subset \cap_{\epsilon>0} \text{io-ZCTIME}(2^{n^\epsilon})
\end{aligned}$$

Proof of (4): Its the same as above, except EXP ≠ MA is replaced with EXP ≠ BPP and the Equation (1) is used.

Proof of (5): It's analogous to the proof of (1), except that the properties we get are P/log n and not P. The log n-bit advice for this property is precisely the n^ϵ-bit advice for the ZCTIME(2^{n^ϵ}) algorithm we get.

Proofs of (6), (7) & (8): They are analogous to the proofs of (2), (3) and (4). The advice from the proof of (5) travels to them as well. □

In [75] they got: $\exists c \geq 1$ RP $\subseteq$ RE$\cap$BPP $\subset \cap_{\epsilon>0}$ io-ZPTIME(2^{n^ϵ})/n^c. We get the following corollary that is non-trivial, but not equally impressive.

Corollary 8.3. *For* C = N, R, U : $\exists c \geq 1$ CE $\cap$ BPP $\subset \cap_{\epsilon>0}$ *io-*ZCTIME(2^{n^ϵ})/n^c

8.2 Lower bounds against general circuit classes vs derandomization of higher Arthur-Merlin classes

In this section we extend the "lower bounds vs derandomization" framework of [7] to get results for all general circuit classes. We also extend the tighter connection of [18] to all general circuit classes. We use our general high-end KLT (Theorem 7.5) in all these extensions. Summary of all the results from this section can be found in the Table 2. Each row corresponds to an equivalence between lower bounds and derandomization of randomized classes. The derandomization is: (i) of the classes that have a checkmark in their column; (ii) works in $\cap_{\epsilon>0}$ io-Σ_{i+1}TIME(2^{n^ϵ}) (and uses

sub-polynomial advice if the checkmark has a super-script n^ϵ). The lower bounds are: (i) for the class(es) in the first column; (ii) against the fix-polynomial size of the circuit class (set of classes) that has a checkmark in its column (for columns 4-6 the fix-polynomial lower bounds are also equivalent to super-polynomial lower bounds); (iii) the star in the first row indicates that the lower bound is only against the non-adaptive Σ_i-oracle circuits (and not against Σ_i / Π_i / $\mathsf{SV}\Sigma_i$ circuits).

Table 2. Derandomization vs Lower Bounds

Complexity Class	$\Delta_i(poly)$	$\mathsf{P}^{\Sigma_i}/poly$	$(\Sigma_{i+1} \cap \Pi_{i+1})/poly$	AM_i / MA_{i+1}	BPP^{Σ_i} / MA^{Σ_i}	$\mathsf{BPP}^{\Sigma_{i+1}\cap\Pi_{i+1}}$ / $\mathsf{MA}^{\Sigma_{i+1}\cap\Pi_{i+1}}$
ip-$(\Sigma_{i+1}\mathsf{E} \cap \Pi_{i+1}\mathsf{E})$	$\checkmark^\star$			$\checkmark_{io}$		
ip-$(\Sigma_{i+1}\mathsf{E} \cap \Pi_{i+1}\mathsf{E})$		$\checkmark$			$\checkmark_{io}$	
ip-$(\Sigma_{i+1}\mathsf{E} \cap \Pi_{i+1}\mathsf{E})$			$\checkmark$			$\checkmark_{io}$
$\Pi_{i+1}\mathsf{E}$ / $\Sigma_{i+1}\mathsf{E}$ / $\mathsf{E}_{\|\|}^{\Sigma_{i+1}}$	$\checkmark$			$\checkmark_{io}^{n^\epsilon}$		
$\Pi_{i+1}\mathsf{E}$ / $\Sigma_{i+1}\mathsf{E}$ / $\mathsf{E}_{\|\|}^{\Sigma_{i+1}}$		$\checkmark$			$\checkmark_{io}^{n^\epsilon}$	
$\Pi_{i+1}\mathsf{E}$ / $\Sigma_{i+1}\mathsf{E}$ / $\mathsf{E}_{\|\|}^{\Sigma_{i+1}}$			$\checkmark$			$\checkmark_{io}^{n^\epsilon}$
$\mathsf{E}^{\Sigma_{i+1}}$	$?_1$	$?_2$	$?_3$	$?_1$	$?_2$	$?_3$

Note that, in the rows 1-3 one can use similar arguments as give in the Section 5 to replace the class ip-$(\Sigma_{i+1}\mathsf{E} \cap \Pi_{i+1}\mathsf{E})$ with $(\Sigma_{i+1}\mathsf{E} \cap \Pi_{i+1}\mathsf{E})/O(1)$, and the 'derandomization without advice' with 'derandomization with $O(1)$ advice'.

Recall that, CAPP (circuit approximation probability problem) is the problem where the input is an n-input $s(n)$-size circuit C, and the output is the fraction of the 2^n inputs that C accepts (within $\pm 1/s$ error). $s(n)$ is $poly(n)$ by default, and we measure the complexity of a CAPP problem in terms of the input length of the input circuit, i.e., in terms of n and not $s(n)$. For $s(n) = poly(n)$, these problems turn into classes of problems: one problem for each polynomial. We abuse the notation and treat these classes as single problems.

We use CAPP_{Σ_i}, CAPP_{Π_i}, $\mathsf{CAPP}^{\Sigma_i\cap\Pi_i}$, CAPP^{Σ_i} and $\mathsf{CAPP}_{||}^{\Sigma_i}$ to denote the CAPP problem for Σ_i, Π_i, $(\Sigma_i \cap \Pi_i)$-oracle, $\Sigma_i\mathsf{SAT}$-oracle, and non-adaptive $\Sigma_i\mathsf{SAT}$-oracle circuits, respectively. Note that, $\mathsf{CAPP}^{\Sigma_i\cap\Pi_i}$ is the class of all CAPP problems for A-oracle circuits, for all $A \in \Sigma_i \cap \Pi_i$. A non-deterministic algorithm for CAPP, on any non-deterministic branch, either outputs the correct value (within proper range, on at least one branch), or outputs '?' / nothing.

Faster CAPP algorithms are known to be equivalent to the derandomization of the associated randomized classes. We also use them as an intermediate step in proving our equivalences.

First we prove the results from the row 4 of Table 2. We then extend that to rows 6 and 5. Finally we prove the results from the rows 1-3.

Theorem 8.4. *[Row 4 of Table 2] The following are equivalent for integer $i \geq 0$:*

(1) $\mathsf{prAM}_i \subset \cap_{\epsilon>0}$ *io*-$\Sigma_{i+1}\mathsf{TIME}(2^{n^\epsilon})/n^\epsilon$
(2) $\mathsf{prMA}_{i+1} \subset \cap_{\epsilon>0}$ *io*-$\Sigma_{i+1}\mathsf{TIME}(2^{n^\epsilon})/n^\epsilon$
(3) $(\Pi_{i+1}\mathsf{E} / \Sigma_{i+1}\mathsf{E} / \mathsf{E}_{||}^{\Sigma_{i+1}}) \not\subset \Delta_i(poly)$
(4) $\forall k \geq 1$ $(\Pi_{i+1}\mathsf{E} / \Sigma_{i+1}\mathsf{E} / \mathsf{E}_{||}^{\Sigma_{i+1}}) \not\subset \Delta_i(n^k)$
(5) $\mathsf{CAPP}_{\Sigma_i} \in \cap_{\epsilon>0}$ *io*-$\Sigma_{i+1}\mathsf{TIME}(2^{n^\epsilon})/n^\epsilon$

Proof. (1) $\implies$ (2) Let $\Pi = (\Pi_Y, \Pi_N)$ be a promise problem in prMA_{i+1}. Let $\Psi = (\Psi_Y, \Psi_N)$ be the corresponding prAM_i problem, and Γ be the corresponding Π_i verifier. Let c be a constant such that:

$$x \in \Pi_Y \implies \exists (y \in \{0,1\}^{n^c})[(x,y) \in \Psi_Y \wedge (x,y) \in \Gamma]$$
$$x \in \Pi_N \implies \forall (y \in \{0,1\}^{n^c})[(x,y) \in \Psi_N \vee (x,y) \notin \Gamma]$$

Fix some $\epsilon > 0$. We give a $\Sigma_{i+1}\mathsf{TIME}(2^{n^\epsilon})/n^\epsilon$ algorithm $\mathcal{A}$ that decides Π on infinitely many input lengths. $\mathcal{A}$ first non-deterministically guesses an n^c-length string y, then simulates some Π_i-algorithm for Γ on the input (x,y). It rejects if the Π_i-algorithm rejects. Else, it simulates a $\Sigma_{i+1}\mathsf{TIME}(2^{n^{\epsilon/c}})/n^{\epsilon/c}$ algorithm $\mathcal{A}'$ for Ψ on the input (x,y) that is correct on infinitely many input lengths (which exists due to our assumption). It rejects if $\mathcal{A}'$ rejects, else it accepts. The quantifiers of the Π_i-algorithm and the algorithm $\mathcal{A}'$ can be easily merged to give the desired algorithm.

(2) $\implies$ (3, 4) First note that all the assumptions in (3) are equivalent because of the general downward collapse theorem (Corollary 7.3) and the fact that $\mathsf{E}_{||}^{\Sigma_{i+1}} \subset \Sigma_{i+1}\mathsf{E}/O(n)$, where the advice encodes the number of positive Σ_i oracle queries for all the inputs of that length. The assumptions in (3) imply assumptions in (4). The reverse is true because $\mathsf{E}_{||}^{\Sigma_{i+1}}/O(n) = \Sigma_{i+1}\mathsf{E}/O(n)$, and $\Sigma_{i+1}\mathsf{E}$ has complete problems under linear-time reductions.

Now we show that the assumptions $\mathsf{prMA}_{i+1} \subset$ io-$\Sigma_{i+1}\mathsf{E}/n$ and $\Sigma_{i+1}\mathsf{E}/n \subset \Sigma_i\mathsf{SIZE}(n^k)$ (for some constant k) lead to a contradiction. The latter assumption implies $\mathsf{EXP} \subset \Sigma_i/poly$, which in turn implies $\mathsf{EXP} = \mathsf{MA}_{i+1}$ by the general KLT (Theorem 7.5). Now the former assumption implies $\mathsf{EXP} \subset$ io-$\Sigma_i\mathsf{SIZE}(n^k)$. This is a contradiction because in EXP we can diagonalize against any fix-polynomial size general circuit class.

(3) $\implies$ (5) Let C be an input to CAPP_{Σ_i}, i.e. a Σ_i circuit of size n^d, for some constant d. Fix an $\epsilon > 0$ and $\epsilon' < \epsilon/2$. We give a $\Sigma_{i+1}\mathsf{TIME}(2^{n^\epsilon})/n^\epsilon$ algorithm that correctly approximates the fraction of inputs that C accepts (within $\pm 1/n^d$ error). Our algorithm works for infinitely many values of n.

We start with the hardness assumption $\mathsf{E}_{||}^{\Sigma_{i+1}} \not\subset \Sigma_i/poly$. This assumption gives us a language $L \in \mathsf{E}_{||}^{\Sigma_{i+1}}$ that doesn't have $\mathsf{SIZE}_{||}^{\Sigma_i}(n^{gd/\epsilon'})$ (where g is the constant from Theorem 2.4) circuits for infinitely many lengths n. Using Theorem 2.4 we first construct a PRG $G : n^{\epsilon'} \to n^d$ that fools $\mathsf{SIZE}_{||}^{\Sigma_i}(n^d)$ circuits (and thus $\Sigma_i\mathsf{SIZE}(n^d)$ circuits) for infinitely many n. Then apply this PRG G to solve CAPP for C: by brute-forcing through G's seeds.

Carefully brute-forcing through seeds: $C \in \mathsf{CAPP}_{\Sigma_i}$ if and only if we are able to guess $2^{n^{\epsilon'}}/2$ seeds y where $C(G(y)) = 1$. So to solve CAPP for C in $\Sigma_{i+1}\mathsf{TIME}(2^{n^\epsilon})/n^\epsilon$ we do the following: guess $2^{n^{\epsilon'}}/2$ seeds and what G would output on them, and for each guessed seed y check if $G(y)$ was guessed correctly, and that $C(G(y)) = 1$.

Note that, after the initial non-deterministic guessing, we just want positive answers from some queries. Query where $G(y)$ value is being verified can be computed in $\Sigma_{i+1}\mathsf{TIME}(2^{O(n^{2\epsilon'})})/O(n^{2\epsilon'})$. Query where C needs to be computed can be done in Σ_i. The total number of queries is bounded by $O(2^{n^{2\epsilon'}})$, thus all the queries can be combined to give a $\Sigma_{i+1}\mathsf{TIME}(2^{n^\epsilon})/n^\epsilon$ algorithm.

(5) $\implies$ (1) Let L be a problem in prAM_i. Then L has a $\mathsf{BP} \cdot \Sigma_i$ algorithm (that works for all promise inputs) where the length of the BP quantifier and all the other quantifiers, and the running-time of the algorithm is upper-bounded by $n^{d/2-1}$, for some constant d. For any n-length input x, fixing x and including the BP quantifier into the input, we get an n^d-size Σ_i circuit C_x. We use the CAPP_{Σ_i} algorithm on these circuits C_x (from the assumption) to give the desired derandomization of prAM_i. □

Theorem 8.5. *[Row 6 of Table 2] The following are equivalent for integer $i \geq 0$:*

(1) $\mathsf{prBPP}^{\Sigma_{i+1}\cap\Pi_{i+1}} \subset \cap_{\epsilon>0}\ io\text{-}\Sigma_{i+1}\mathsf{TIME}(2^{n^\epsilon})/n^\epsilon$
(2) $\mathsf{prMA}^{\Sigma_{i+1}\cap\Pi_{i+1}} \subset \cap_{\epsilon>0}\ io\text{-}\Sigma_{i+1}\mathsf{TIME}(2^{n^\epsilon})/n^\epsilon$
(3) $(\Pi_{i+1}\mathsf{E}\ /\ \Sigma_{i+1}\mathsf{E}\ /\ \mathsf{E}_{||}^{\Sigma_{i+1}}) \not\subset (\Sigma_{i+1}\cap\Pi_{i+1})/poly$
(4) $\forall k \geq 1\ (\Pi_{i+1}\mathsf{E}\ /\ \Sigma_{i+1}\mathsf{E}\ /\ \mathsf{E}_{||}^{\Sigma_{i+1}}) \not\subset \mathsf{prSV}\Sigma_{i+1}\mathsf{SIZE}(n^k)$
(5) $\forall k \geq 1\ (\Pi_{i+1}\mathsf{E}\ /\ \Sigma_{i+1}\mathsf{E}\ /\ \mathsf{E}_{||}^{\Sigma_{i+1}}) \not\subset w\text{-}\mathsf{prSV}\Sigma_{i+1}\mathsf{SIZE}(n^k)$
(6) $\mathsf{CAPP}^{\Sigma_{i+1}\cap\Pi_{i+1}} \subset \cap_{\epsilon>0}\ io\text{-}\Sigma_{i+1}\mathsf{TIME}(2^{n^\epsilon})/n^\epsilon$

Proof. (1 $\implies$ 2) Any $L \in \mathsf{prMA}^{\Sigma_{i+1}\cap\Pi_{i+1}}$ can be converted to some $L' \in \mathsf{prBPP}^{\mathsf{NP}\cap\mathsf{Co\text{-}NP}}$ if Merlin's non-determinism is included in the input. If Merlin's non-determinism is bounded by n^c for some constant c, then for any ϵ we use the $\Sigma_{i+1}\mathsf{TIME}(2^{n^{\epsilon/c}})/n^{\epsilon/c}$ algorithm for L' to give a $\Sigma_{i+1}\mathsf{TIME}(2^{n^\epsilon})/n^\epsilon$ algorithm for L.

(2) $\implies$ (3, 4, 5) Like Theorem 8.4, we get the equivalence of all the assumptions in (3,4,5) due to complete problems (under linear-time reduction) in $\Sigma_{i+1}\mathsf{E}$, and the fact $\mathsf{E}_{||}^{\Sigma_{i+1}}/O(n) = \Sigma_{i+1}/O(n)$.

Now we show that the assumptions $\mathsf{prMA}^{\Sigma_{i+1}\cap\Pi_{i+1}} \subset$ i.o. $\Sigma_{i+1}\mathsf{E}/n$ and $\Sigma_{i+1}\mathsf{E} \subset \mathsf{prSV}\Sigma_{i+1}\mathsf{SIZE}(n^k)$ (for some constant k) lead to a contradiction. Again like Theorem 8.4, we use Theorem 7.5 to get the contradiction $\mathsf{EXP} = \mathsf{MA}^{\Sigma_{i+1}\cap\Pi_{i+1}} \subset \text{io-}\mathsf{prSV}\Sigma_i\mathsf{SIZE}(n^k)$.

(3) $\implies$ (6) Let C be an input to $\mathsf{CAPP}^{\Sigma_{i+1}\cap\Pi_{i+1}}$, i.e. a $\mathsf{SIZE}^A(n^d)$ circuit for some constant d and oracle $A \in \Sigma_{i+1}\cap\Pi_{i+1}$. Fix an $\epsilon > 0$ and $\epsilon' < \epsilon/2$. The assumption $\mathsf{E}_{||}^{\Sigma_{i+1}} \not\subset (\Sigma_{i+1}\cap\Pi_{i+1})/poly$ gives us a language $L \in \mathsf{E}_{||}^{\Sigma_i}$ that doesn't have $\mathsf{SIZE}^A(n^{gd/\epsilon'})$ circuits (since $\Sigma_{i+1}\cap\Pi_{i+1} = \mathsf{P}^{\Sigma_{i+1}\cap\Pi_{i+1}}$), infinitely often. Now same as in the proof of Theorem 8.4, we again use the Theorem 2.4 to construct a $\Sigma_{i+1}\mathsf{TIME}(2^{n^{\epsilon'}})/n^{\epsilon'}$ computable PRG $G : n^{\epsilon'} \to n^d$ that fools n^d-size A-oracle circuits, infinitely often. We use this G to fool C.

Carefully brute-forcing through seeds: It's the same as in the proof of Theorem 8.4, except: The queries where C needs to be computed, requires some extra non-deterministic guessing to be done at the initial guessing step. The answers to all the oracle queries, that C would make during the computation of $C(G(y))$ for each guessed seed y, is also guessed. These guesses are verified in Σ_{i+1} using A and $\overline{A}$, and thus $C(G(y))$ is computed in Σ_{i+1}. This way, after merging the quantifiers of the queries, we get an $\Sigma_{i+1}\mathsf{TIME}(2^{n^\epsilon})/n^\epsilon$ computation.

(6) $\implies$ (1) Same as in the proof of Theorem 8.4, we can show that any $L \in \mathsf{prBPP}^{\Sigma_{i+1}\cap\Pi_{i+1}}$ can be reduced to $\mathsf{CAPP}^{\Sigma_{i+1}\cap\Pi_{i+1}}$. The proof then follows. □

Theorem 8.6. *[Row 5 of Table 2] The following are equivalent for integer $i \geq 0$:*

(1) $\mathsf{prBPP}^{\Sigma_i} \subset \cap_{\epsilon>0}\ io\text{-}\Sigma_{i+1}\mathsf{TIME}(2^{n^\epsilon})/n^\epsilon$
(2) $\mathsf{prMA}^{\Sigma_i} \subset \cap_{\epsilon>0}\ io\text{-}\Sigma_{i+1}\mathsf{TIME}(2^{n^\epsilon})/n^\epsilon$
(3) $(\Pi_{i+1}\mathsf{E}\ /\ \Sigma_{i+1}\mathsf{E}\ /\ \mathsf{E}_{||}^{\Sigma_{i+1}}) \not\subset \mathsf{P}^{\Sigma_i}/poly$
(4) $\forall k \geq 1\ (\Pi_{i+1}\mathsf{E}\ /\ \Sigma_{i+1}\mathsf{E}\ /\ \mathsf{E}_{||}^{\Sigma_{i+1}}) \not\subset \mathsf{SIZE}^{\Sigma_i}(n^k)$
(5) $\mathsf{CAPP}^{\Sigma_i} \in \cap_{\epsilon>0}\ io\text{-}\Sigma_{i+1}\mathsf{TIME}(2^{n^\epsilon})/n^\epsilon$

Proof. Except the computation of $C(G(y))$, all the steps either follow from the proof of Theorem 8.4 or the proof of Theorem 8.6. The circuit C has an equivalent $\Sigma_{i+1}(|C|^c)$ circuit that can be obtained uniformly. Here the constant c doesn't depend on the input length. We use this circuit as in the proofs of Theorem 8.4 and 8.5 to make the final computation $\Sigma_{i+1}\mathsf{TIME}(2^{n^\epsilon})/n^\epsilon$. □

The first three rows of the Table 2 follow the same proof pattern, except few minor technical challenges that make the results not so clean: (i) Due to the lack of a complete language in $(\Sigma_{i+1}\mathsf{E}\cap\Pi_{i+1}\mathsf{E})$, the equivalence of fix-polynomial and super-polynomial lower bounds can't be

established. (ii) Due to the lack of a downward collapse theorem that works for fix-polynomial lower-bounds (for the entire class), the equivalence between lower bounds against non-adaptive Σ_i-oracle circuits and $\mathtt{SV}\Sigma_i$ circuits can't be established (one can overcome this challenge using the stronger notion of super-polynomial and sub-exponential from [18], but we stick to the fix-polynomial setting) (iii) The stronger io-$(\Sigma_{i+1}\mathtt{SUBEXP} \cap \Pi_{i+1}\mathtt{SUBEXP})$ derandomization fails to transfer from $\mathtt{prAM}_i$ to $\mathtt{prMA}_{i+1}$ (same issue with the other two pairs of randomized classes). So to keep the derandomization assumption same (without advice), we need to make the lower bound assumption weaker (the 'ip' version).

Theorem 8.7. *[Rows 1-3 of Table 2] For integer $i \geq 0$:*

(1) $\mathtt{pr}(\mathtt{AM}_i/\mathtt{MA}_{i+1}) \subset$ *io*-$\Sigma_{i+1}\mathtt{SUBEXP} \iff \forall k \geq 1$ *ip*-$(\Sigma_{i+1}\mathtt{E} \cap \Pi_{i+1}\mathtt{E}) \not\subset \mathtt{SIZE}^{\Sigma_i}_{||}(n^k)$

(2) $\mathtt{pr}(\mathtt{BPP}/\mathtt{MA})^{\Sigma_i} \subset$ *io*-$\Sigma_{i+1}\mathtt{SUBEXP} \iff \forall k \geq 1$ *ip*-$(\Sigma_{i+1}\mathtt{E} \cap \Pi_{i+1}\mathtt{E}) \not\subset \mathtt{SIZE}^{\Sigma_i}(n^k)$

(3) $\mathtt{pr}(\mathtt{BPP}/\mathtt{MA})^{\Sigma_{i+1}\cap\Pi_{i+1}} \subset$ *io*-$\Sigma_{i+1}\mathtt{SUBEXP} \iff \forall k \geq 1$ *ip*-$(\Sigma_{i+1}\mathtt{E} \cap \Pi_{i+1}\mathtt{E}) \not\subset \mathtt{prSV}\Sigma_{i+1}\mathtt{SIZE}(n^k)$

Proof. The equivalence of the derandomization of the two randomized classes follows from the same arguments that were used in the proofs of Theorem 8.4, 8.6 and 8.5.

Derandomization to lower bound: We first give a proof for (2). ip-$(\Sigma_{i+1}\mathtt{E} \cap \Pi_{i+1}\mathtt{E}) \subset \mathtt{SIZE}^{\Sigma_i}(n^k)$ yields the contradiction $\mathtt{EXP} = \mathtt{MA}^{\Sigma_i} \cap \mathtt{Co\text{-}MA}^{\Sigma_i} \subset$ ip-$(\Sigma_{i+1}\mathtt{E} \cap \Pi_{i+1}\mathtt{E}) \subset \mathtt{SIZE}^{\Sigma_i}(n^k)$ using the general $\mathtt{KLT}$ (Theorem 7.5). Note that, ip-$(\Sigma_{i+1}\mathtt{E} \cap \Pi_{i+1}\mathtt{E}) \subset \mathtt{SIZE}^{\Sigma_i}(n^k)$ gives $\mathtt{EXP} \subset$ io-$\mathtt{SIZE}(n^k)$, but even this is a contradiction.

The only non-trivial inclusion is $\mathtt{MA}^{\Sigma_i} \cap \mathtt{Co\text{-}MA}^{\Sigma_i} \subset$ ip-$(\Sigma_{i+1}\mathtt{E} \cap \Pi_{i+1}\mathtt{E})$. We achieve this using $\mathtt{MA}^{\Sigma_i} \subset$ io-$\Sigma_{i+1}\mathtt{SUBEXP}$. For $L \in \mathtt{MA}^{\Sigma_i} \cap \mathtt{Co\text{-}MA}^{\Sigma_i}$ we construct a new $\mathtt{MA}^{\Sigma_i} \cap \mathtt{Co\text{-}MA}^{\Sigma_i}$ language $L' = \{bx \mid (b = 1 \wedge x \in L) \vee (b = 0 \wedge x \in \overline{L})\}$. Now the assumption gives $L' \in$ io-$\Sigma_{i+1}\mathtt{SUBEXP}$. A $\Sigma_{i+1}\mathtt{SUBEXP}$ algorithm for L' that works infinitely often, yields $\Sigma_{i+1}\mathtt{SUBEXP}$ algorithms for L and $\overline{L}$ that work fine on infinitely many input lengths (for the other input lengths the algorithms might not accept complimentary inputs). Thus $L \in$ ip-$(\Sigma_{i+1} \cap \Pi_{i+1})$.

Similar proof works for (1) and (3). For (3), we need that the $(\Sigma_{i+1} \cap \Pi_{i+1})$-oracles used by the $\mathtt{MA}$-protocols for L and $\overline{L}$ must be the same. We actually get the same oracle from the Theorem 7.5. Even if they are different, we can merge them into one oracle, using the same trick that we used to merge L and $\overline{L}$ into L'.

Lower bound to derandomization: We first give a proof for (2). ip-$(\Sigma_{i+1}\mathtt{E} \cap \Pi_{i+1}\mathtt{E}) \not\subset \mathtt{SIZE}^{\Sigma_i}(n^k)$ yields, for infinitely many $n \in \mathbb{N}$, functions that can be computed in $\Sigma_{i+1}\mathtt{E} \cap \Pi_{i+1}\mathtt{E}$ and don't have $\mathtt{SIZE}^{\Sigma_i}(n^k)$ circuits. From the lower bound assumption this is true for every $k \geq 1$. Now for any $L \in \mathtt{prBPP}^{\Sigma_i}$, let the corresponding $\mathtt{BP}^{\Sigma_i}$ computation for the promise inputs be represented by $\mathtt{SIZE}^{\Sigma_i}(n^c)$ circuits for some constant c. Fix $\epsilon > 0$. We choose the appropriate k, and construct PRG G similar to the one constructed in Theorem 8.6 that fools $\mathtt{SIZE}^{\Sigma_i}(n^c)$ circuits, infinitely often. The entire fooling process can be done in $\Sigma_{i+1}\mathtt{TIME}(2^{n^\epsilon})$ without advice (because now the $G(y)$-type of queries don't need advice).

Similar proofs work for (1) and (3). For (3), we use the fact: For any oracle $A \in (\Sigma_{i+1} \cap \Pi_{i+1})$, for any $k \geq 1$, the assumption gives us an $L \in$ ip-$(\Sigma_{i+1} \cap \Pi_{i+1})$ that doesn't have $\mathtt{SIZE}^A(n^k)$ circuits. This is true, else ip-$(\Sigma_{i+1} \cap \Pi_{i+1}) \subset \mathtt{prSV}\Sigma_{i+1}\mathtt{SIZE}(n^{k'})$ for some k' that depends on k and A. □

8.3 General downward separation

In this section we use the equivalences from Table 2, and the unconditional lower bounds from Theorem 7.10, to transfer lower bounds from exponential time to sub-exponential time of certain classes.

Theorem 8.8. *For integer $i \geq 0, \Gamma = \Sigma_{i+1}\mathtt{SUBEXP} \cap \Pi_{i+1}\mathtt{SUBEXP}$ and $\Gamma' = \cap_{\epsilon>0}(\Sigma_{i+1}\mathtt{TIME}(2^{n^\epsilon})/n^\epsilon \cap \Pi_{i+1}\mathtt{TIME}(2^{n^\epsilon})/n^\epsilon)$:*

(1) $\forall k \geq 1\ \mathsf{E}_{||}^{\Sigma_{i+1}} \not\subset io\text{-}\mathsf{SV}\Sigma_i\mathsf{SIZE}(n^k) \implies \forall k \geq 1\ \Gamma' \not\subset \mathsf{SIZE}_{||}^{\Sigma_i}(n^k)$
(2) $\forall k \geq 1\ \mathsf{E}_{||}^{\Sigma_{i+1}} \not\subset io\text{-}\mathsf{SIZE}^{\Sigma_i}(n^k) \implies \forall k \geq 1\ \Gamma' \not\subset \mathsf{SIZE}^{\Sigma_i}(n^k)$
(3) $\forall k \geq 1\ \mathsf{E}_{||}^{\Sigma_{i+1}} \not\subset w\text{-}io\text{-}\mathsf{prSV}\Sigma_{i+1}\mathsf{SIZE}(n^k) \implies \forall k \geq 1\ \Gamma' \not\subset w\text{-}\mathsf{prSV}\Sigma_{i+1}\mathsf{SIZE}(n^k)$
(4) $\forall k \geq 1\ \Sigma_{i+1}\mathsf{E} \cap \Pi_{i+1}\mathsf{E} \not\subset io\text{-}\mathsf{SIZE}_{||}^{\Sigma_i}(n^k) \implies \forall k \geq 1\ \Gamma \not\subset \mathsf{SIZE}_{||}^{\Sigma_i}(n^k)$
(5) $\forall k \geq 1\ \Sigma_{i+1}\mathsf{E} \cap \Pi_{i+1}\mathsf{E} \not\subset io\text{-}\mathsf{SIZE}^{\Sigma_i}(n^k) \implies \forall k \geq 1\ \Gamma \not\subset \mathsf{SIZE}^{\Sigma_i}(n^k)$
(6) $\forall k \geq 1\ \Sigma_{i+1}\mathsf{E} \cap \Pi_{i+1}\mathsf{E} \not\subset w\text{-}io\text{-}\mathsf{prSV}\Sigma_{i+1}\mathsf{SIZE}(n^k) \implies \forall k \geq 1\ \Gamma \not\subset w\text{-}\mathsf{prSV}\Sigma_{i+1}\mathsf{SIZE}(n^k)$

Proof. If these stronger lower bounds (against infinitely often circuit classes) are plugged into the proofs from the previous section, then the PRGs they yield, fool the corresponding circuits on all input lengths, and the derandomization of the corresponding randomized classes works on all input lengths. From Theorem 7.10 we have weaker lower bounds for the corresponding randomized classes that are being derandomized. These lower bounds thus transfer to Γ and Γ'. □

9 THE SPECIAL CASE OF $(\mathsf{NP} \cap \mathsf{Co\text{-}NP})/poly$

In this section we discuss the consequences of lower bound $\mathsf{NEXP} \not\subset (\mathsf{NP} \cap \mathsf{Co\text{-}NP})/poly$. In Section 9.1 we discuss its equivalence with a number of other lower bounds and derandomization results. In Section 9.2 we extend some of the above equivalences to ZNE, for the fix-polynomial lower bound case. In Sections 9.3, 9.4 and 9.5: we show how one can obtain the lower bound $\mathsf{NEXP} \not\subset (\mathsf{NP} \cap \mathsf{Co\text{-}NP})/poly$ from fast TAUT and CAPP algorithms. In Section 9.6 we show gap theorems for MA, $\mathsf{MA}^{\mathsf{NP}\cap\mathsf{Co\text{-}NP}}$ and $\mathsf{CAPP}^{\mathsf{NP}\cap\mathsf{Co\text{-}NP}}$. In Section 9.7 we prove an unconditional lower bound for NEXP against the ACC analogue of $(\mathsf{NP} \cap \mathsf{Co\text{-}NP})/poly$ (with sub-polynomial non-determinism).

9.1 NEXP vs $(\mathsf{NP} \cap \mathsf{Co\text{-}NP})/poly$

In this section we give NEXP EWL and KLT for $(\mathsf{NP} \cap \mathsf{Co\text{-}NP})/poly$, and the converses. We also extend the results to $\mathsf{E}_{||}^{\mathsf{NP}}$. All these results work even if replace $\mathsf{NP} \cap \mathsf{Co\text{-}NP}$ with P (in circuit classes and as oracles). That is, similar proof also extends the Theorem 4.10 to $\mathsf{E}_{||}^{\mathsf{NP}}$, to give its KLT for $\mathsf{P}/poly$: $\mathsf{E}_{||}^{\mathsf{NP}} \subset \mathsf{P}/poly \iff \mathsf{E}_{||}^{\mathsf{NP}} = \mathsf{MA} \iff \mathsf{prMA} \subset \cap_{\epsilon>0}\ \text{io-}\mathsf{NTIME}(2^{n^\epsilon})/n^\epsilon$.

Theorem 9.1. *The following statements are equivalent:*
(1) $(\mathsf{NE}/\mathsf{E}_{||}^{\mathsf{NP}}) \not\subset \mathsf{MA}^{\mathsf{NP}\cap\mathsf{Co\text{-}NP}}$
(2) $(\mathsf{NE}/\mathsf{E}_{||}^{\mathsf{NP}}) \not\subset (\mathsf{NP} \cap \mathsf{Co\text{-}NP})/poly$
(3) $\mathsf{NE} \not\subset_{ow} (\mathsf{NP} \cap \mathsf{Co\text{-}NP})/poly$
(4) $\mathsf{NE} \not\subset_{hw} (\mathsf{NP} \cap \mathsf{Co\text{-}NP})/poly$
(5) $\mathsf{NE} \not\subset_{w} (\mathsf{NP} \cap \mathsf{Co\text{-}NP})/poly$
(6) $\mathsf{NP}/\log n\text{-U} \not\subset_{tt} (\mathsf{NP} \cap \mathsf{Co\text{-}NP})/poly$
(7) $\mathsf{NP}/\log n\text{-N} \not\subset_{tt} (\mathsf{NP} \cap \mathsf{Co\text{-}NP})/poly$
(8) $\mathsf{P}/\log n\text{-N} \not\subset_{tt} (\mathsf{NP} \cap \mathsf{Co\text{-}NP})/poly$
(9) $\mathsf{CAPP}^{\mathsf{NP}\cap\mathsf{Co\text{-}NP}} \subset \cap_{\epsilon>0}\ io\text{-}\mathsf{NTIME}(2^{n^\epsilon})/n^\epsilon$
(10) $\mathsf{pr}(\mathsf{BPP}/\mathsf{MA})^{\mathsf{NP}\cap\mathsf{Co\text{-}NP}} \subset \cap_{\epsilon>0}\ io\text{-}\mathsf{NTIME}(2^{n^\epsilon})/n^\epsilon$

Proof. (1) $\implies$ (2) First note that both the assumptions in (2) are equivalent from the arguments used in the previous section. Now we prove the contrapositive. $\mathsf{NE} \subset (\mathsf{NP} \cap \mathsf{Co\text{-}NP})/poly$ gives us $\mathsf{EXP} = \mathsf{MA}^{\mathsf{NP}\cap\mathsf{Co\text{-}NP}}$ from the general KLT (Theorem 7.5), and $\exists k\ \mathsf{NE}/O(n) \subset \mathsf{NSIZE}(n^k)$ using a complete problem for NE under liner time reductions.

If $\mathsf{NE} \not\subset \mathsf{MA}^{\mathsf{NP}\cap\mathsf{Co\text{-}NP}} = \mathsf{EXP}$, then from [31] we get $\mathsf{AM} \subset \text{io-}\mathsf{NE}/O(n)$. Since $\mathsf{MA}^{\mathsf{NP}\cap\mathsf{Co\text{-}NP}} \subseteq \mathsf{AM}$ from Lemma 7.4, we get $\exists k\ \mathsf{EXP} = \mathsf{MA}^{\mathsf{NP}\cap\mathsf{Co\text{-}NP}} \subset \text{io-}\mathsf{NSIZE}(n^k)$. This is a contradiction since in EXP we can diagonalize against any fix-polynomial size general circuit class.

If $\mathsf{E}^{\mathsf{NP}}_{||} \not\subset \mathsf{MA}^{\mathsf{NP}\cap\mathsf{Co\text{-}NP}} = \mathsf{EXP}$, then we give a similar argument to prove $\exists k\ \mathsf{EXP} \subset \text{io-}\mathsf{NSIZE}(n^k)$. We show $\mathsf{AM} \subset \text{io-}\mathsf{NE}/O(n)$ by using a language $L \in \mathsf{E}^{\mathsf{NP}}_{||}$, that has a $\mathsf{E}^{\mathsf{NP}}_{||}$ algorithm $\mathcal{A}$ deciding it such that: there can't be any $\mathsf{SIZE}^{\mathsf{NP}}_{||}(n^k)$ circuits for any constant k, encoding the witnesses for all the positive oracle queries that $\mathcal{A}$ makes on all n-length input, for infinitely many n (else brute-forcing through these circuits will prove $\mathsf{E}^{\mathsf{NP}}_{||} \subset \mathsf{EXP}$). Now using $\mathcal{A}$ we get an $\mathsf{NE}/O(n)$ algorithm $\mathcal{B}$ that produces strings tt with $ckt^{\mathsf{NP}}_{||}(tt) > n^{k-1}$ for all k. The advice encodes the number of positive oracle queries that $\mathcal{A}$ makes on that input length. For any n, $\mathcal{B}$ simulates $\mathcal{A}$ on all n-length inputs and using advice guesses that many queries to be positive. It verifiers its guesses by non-deterministically guessing certificates for the positive oracle queries. After all the verification steps, it outputs the concatenation of all its non-deterministic certificates. This concatenated string can't have $\mathsf{SIZE}^{\mathsf{NP}}_{||}(n^{k-1})$ circuits for any constant k (because its sub-strings doesn't have $\mathsf{SIZE}^{\mathsf{NP}}_{||}(n^k)$ circuits). Now we use $\mathcal{B}$ to get $\mathsf{AM} \subset \text{io-}\mathsf{NE}/O(n)$ (similar to the proof of Theorem 8.4).

(2) $\implies$ (6) The proof of ($\mathsf{NE} \not\subset C \implies \mathsf{NP}/\log n\text{-}\mathsf{U} \not\subset_{tt} C$) from Theorem 4.10 also works here. Any language $L \in \mathsf{NE} \setminus (\mathsf{NP} \cap \mathsf{Co\text{-}NP})/poly$, if provided with $|L_n|$ (size of n^{th}-slice) as advice, converts into an $\mathsf{NP}/\log n$-U property against $(\mathsf{NP} \cap \mathsf{Co\text{-}NP})/poly$.

(6) $\implies$ (7) This follows from the definitions.

(7) $\implies$ (8) The proof of ($\mathsf{NP}/\log n\text{-}\mathsf{N} \not\subset_{tt} C \implies \mathsf{P}/\log n\text{-}\mathsf{N} \not\subset_{tt} C$) from [57] also works here. The non-deterministic certificates are included into the input, inducing a non-significant change in the circuit complexity of the strings accepted by the property.

(8) $\implies$ (5) The proof of ($\mathsf{P}/\log n\text{-}\mathsf{N} \not\subset_{tt} C \implies \mathsf{NE} \not\subset_{w} C$) from [57, 75] also works here. The advice becomes input, and input becomes certificates. Thus the useful advice sequence converts into useful input sequence, i.e. inputs that have non-easy witnesses / witnesses of high circuit complexity.

(5) $\implies$ (4) This follows from the definitions.

(4) $\implies$ (3) This follows from the definitions.

(3) $\implies$ (9) This follows from the arguments used in Theorem 8.5. The NE verifier that doesn't have oblivious-witnesses in $(\mathsf{NP} \cap \mathsf{Co\text{-}NP})/poly$, yields a function sequence computable in NE, that infinitely often has high $(\mathsf{NP} \cap \mathsf{Co\text{-}NP})$-oracle circuit complexity (w.r.t. any $(\mathsf{NP} \cap \mathsf{Co\text{-}NP})$-oracle). So we use the Theorem 2.4 to construct a PRG and solve $\mathsf{CAPP}^{\mathsf{NP}\cap\mathsf{Co\text{-}NP}}$ in the desired time.

(9) $\implies$ (10) This implication, and the equivalence of the assumptions in (10), follows from the Theorem 8.5.

(10) $\implies$ (1) If $\mathsf{NE} \subset \mathsf{MA}^{\mathsf{NP}\cap\mathsf{Co\text{-}NP}}$ (or $\mathsf{E}^{\mathsf{NP}}_{||} \subset \mathsf{MA}^{\mathsf{NP}\cap\mathsf{Co\text{-}NP}}$) and $\mathsf{MA}^{\mathsf{NP}\cap\mathsf{Co\text{-}NP}} \subset \cap_{\epsilon>0}\ \text{io-}\mathsf{NTIME}(2^{n^\epsilon})/n^\epsilon$, then we get $\mathsf{EXP} = \mathsf{NEXP} \subset \cap_{\epsilon>0}\ \text{io-}\mathsf{NTIME}(2^{n^\epsilon})/n^\epsilon$. This gives us $\mathsf{EXP} \subset \cap_{\epsilon>0}\ \text{io-}\mathsf{TIME}(2^{n^c})/n$ for some constant c. This is false due to the diagonalization result given in [31]. □

9.2 ZNE vs $(\mathsf{NP} \cap \mathsf{Co\text{-}NP})/poly$

All the results for ZNE from Sections 4 and 5 also work for promise SV non-deterministic circuits, if we replace: MA with $\mathsf{MA}^{\mathsf{NP}\cap\mathsf{Co\text{-}NP}}$; and C with prSVN. We summarize the important connections in the following theorems. As in the Theorem 9.1, here again we use the generalized connections from Section 8.

Theorem 9.2. *For constant $k \geq 1$:*

(1) $\mathsf{ZNE}/O(1) \subset \mathsf{prSVNSIZE}(n^k) \implies \mathsf{ZNE}/O(1) \subset \mathsf{MA}^{\mathsf{NP}\cap\mathsf{Co\text{-}NP}}/O(1)$
(2) $\mathsf{ZNE} \subset \mathsf{prSVNSIZE}(n^k) \implies \mathsf{ZNE} \subset io\text{-}\mathsf{MA}^{\mathsf{NP}\cap\mathsf{Co\text{-}NP}}$
(3) $ip\text{-}\mathsf{ZNE} \subset \mathsf{prSVNSIZE}(n^k) \implies ip\text{-}\mathsf{ZNE} \subset \mathsf{MA}^{\mathsf{NP}\cap\mathsf{Co\text{-}NP}}$

Theorem 9.3. *The following statements are equivalent:*

(1) $\forall k \geq 1$ $\mathsf{ZNE}/O(1) \not\subset \mathsf{prSV}\Sigma_i\mathsf{SIZE}(n^k)$ *(resp.* $\forall k \geq 1$ *ip-*$\mathsf{ZNE} \not\subset \mathsf{prSV}\Sigma_i\mathsf{SIZE}(n^k)$*)*
(2) $\forall k \geq 1$ $\mathsf{ZNE}/O(1) \not\subset_{os} \mathsf{prSV}\Sigma_i\mathsf{SIZE}(n^k)$ *(resp.* $\forall k \geq 1$ *ip-*$\mathsf{ZNE} \not\subset_{os} \mathsf{prSV}\Sigma_i\mathsf{SIZE}(n^k)$*)*
(3) $\forall k \geq 1$ $\mathsf{ZNE}/O(1) \not\subset_{hs} \mathsf{prSV}\Sigma_i\mathsf{SIZE}(n^k)$ *(resp.* $\forall k \geq 1$ *ip-*$\mathsf{ZNE} \not\subset_{hs} \mathsf{prSV}\Sigma_i\mathsf{SIZE}(n^k)$*)*
(4) $\forall k \geq 1$ $\mathsf{ZNE}/O(1) \not\subset_{s} \mathsf{prSV}\Sigma_i\mathsf{SIZE}(n^k)$ *(resp.* $\forall k \geq 1$ *ip-*$\mathsf{ZNE} \not\subset_{s} \mathsf{prSV}\Sigma_i\mathsf{SIZE}(n^k)$*)*
(5) $\forall k \geq 1$ $\mathsf{NP\text{-}U}/O(1) \not\subset_{tt} \mathsf{prSV}\Sigma_i\mathsf{SIZE}(n^k)$ *(resp.* $\forall k \geq 1$ $\mathsf{NP\text{-}prU} \not\subset_{tt} \mathsf{prSV}\Sigma_i\mathsf{SIZE}(n^k)$*)*
(6) $\forall k \geq 1$ $\mathsf{NP\text{-}N}/O(1) \not\subset_{tt} \mathsf{prSV}\Sigma_i\mathsf{SIZE}(n^k)$ *(resp.* $\forall k \geq 1$ $\mathsf{NP\text{-}prN} \not\subset_{tt} \mathsf{prSV}\Sigma_i\mathsf{SIZE}(n^k)$*)*
(7) $\forall k \geq 1$ $\mathsf{P\text{-}N}/O(1) \not\subset_{tt} \mathsf{prSV}\Sigma_i\mathsf{SIZE}(n^k)$ *(resp.* $\forall k \geq 1$ $\mathsf{P\text{-}prN} \not\subset_{tt} \mathsf{prSV}\Sigma_i\mathsf{SIZE}(n^k)$*)*

As in the Theorem 9.1, the ip-ZNE lower bounds from the Theorems 5.5 (for unrestricted Boolean circuits) and 9.3, are also equivalent to "$\mathsf{pr}(\mathsf{BPP}/\mathsf{MA}) \subset \cap_{\epsilon>0}$ io-$\mathsf{NTIME}(2^{n^\epsilon})$" and "$\mathsf{pr}(\mathsf{BPP}/\mathsf{MA})^{\mathsf{NP}\cap\mathsf{Co\text{-}NP}} \subset \cap_{\epsilon>0}$ io-$\mathsf{NTIME}(2^{n^\epsilon})$", respectively.

9.3 Improving exhaustive search for $\mathsf{TAUT}^{\mathsf{NP}\cap\mathsf{Co\text{-}NP}}$ implies $\mathsf{NEXP} \not\subset (\mathsf{NP} \cap \mathsf{Co\text{-}NP})/poly$

The class $\mathsf{TAUT}^{\mathsf{NP}\cap\mathsf{Co\text{-}NP}}$, is defined similar to $\mathsf{CAPP}^{\mathsf{NP}\cap\mathsf{Co\text{-}NP}}$ (see Section 8.2). The inputs are A-oracle circuits, for some $A \in \mathsf{NP} \cap \mathsf{Co\text{-}NP}$, and the output indicates whether the circuit is a tautology or not. The complexity is measured in terms of the input length of the input circuit.

In this section we show: super-polynomial savings in non-deterministic algorithms for the class $\mathsf{TAUT}^{\mathsf{NP}\cap\mathsf{Co\text{-}NP}}$, implies $\mathsf{NEXP} \not\subset (\mathsf{NP} \cap \mathsf{Co\text{-}NP})/poly$. We first state two theorems that we would need in our result.

Theorem 9.4 (NTIME Hierarchy [39]). *Let t_1 and t_2 be time constructible functions that satisfy $t_1(n+1) \in o(t_2(n))$. There is a unary language in $\mathsf{NTIME}(t_2(n))$ that is not in $\mathsf{NTIME}(t_1(n))$.*

Theorem 9.5 (Efficient local reductions [25, 37, 67]). *Every language $L \in \mathsf{NTIME}(2^n)$ can be reduced to 3-SAT instances of $2^n n^c$-size, for some constant c. Moreover, given an instance of L there is an n^c-size P-uniform deterministic circuit that, on an integer $i \in [2^n n^c]$ in binary as input, output the i^{th}-clause of the resulting 3-SAT formula.*

Now we give our result.

Theorem 9.6. *For any super-polynomial function sp, an $\mathsf{NTIME}(2^n/sp(n))$ tautology algorithm for n-input $poly(n)$-size A-oracle circuits, for every $A \in (\mathsf{NP}\cap\mathsf{Co\text{-}NP})$, implies $\mathsf{NEXP} \not\subset (\mathsf{NP}\cap\mathsf{Co\text{-}NP})/poly$.*

Proof. We prove it by contradiction. $\mathsf{NEXP} \subset (\mathsf{NP} \cap \mathsf{Co\text{-}NP})/poly$ along with the faster tautology algorithm will conclude $\mathsf{NTIME}(2^n) \subset \mathsf{NTIME}(2^n/sp(n))$, and thus contradict the non-deterministic time hierarchy from Theorem 9.4.

Reduction circuit: For $L \in \mathsf{NTIME}(2^n)$, we give an $\mathsf{NTIME}(2^n/sp(n))$ algorithm. From the Theorem 9.5 we get: any input x for L uniformly reduces to a 3-SAT instance ϕ_x, where the number of variables and clauses in ϕ_x are bounded by $n^d 2^n$ for some constant d. Moreover the reduction is local in the sense that: it can be uniformly converted to a deterministic circuit C_x that on $(n + d\log n)$-bits input i outputs the three variables x_{i1}, x_{i2}, x_{i3} ($3n + 3d\log n$ bits) from the i^{th}-clause of ϕ_x, along with three extra bits z_1, z_2, z_3 that indicate for each of these three variables, whether they appear as positive literals or a negative literals.

Special verifier: Let V be a non-deterministic verifier for L, that first reduces L to 3-SAT, and then non-deterministically guesses a satisfying assignment for the 3-SAT formula.

Easy-witness circuit: Since $\mathsf{NEXP} \subset (\mathsf{NP} \cap \mathsf{Co\text{-}NP})/poly \implies \mathsf{NEXP} = \mathsf{AM}$, from [31] we get that the search problem for V is in EXP. Thus, there is an algorithm $\mathcal{A}$ that: on any input $x \in L$ outputs y such that $V(x, y) = 1$; on any input $x \notin L$ outputs an all zeros string. Now define a new language $L' = \{(x, i) \mid i^{th}$ output bit of $\mathcal{A}$ on input x is $1\}$. $L' \in \mathsf{EXP}$ and thus $L' \in \mathsf{P}^A/poly$ for some $A \in \mathsf{NP} \cap \mathsf{Co\text{-}NP}$. Let B_x be the A-oracle circuit whose truth-table is a witness for V on input x.

Final Circuit F_x: $(n + d \log n)$-bits long input i is given to C_x. We plug the output variables x_{i1}, x_{i2}, x_{i3} to three different copies of the witness circuit B_x. Let the three output bits be b_1, b_2, b_3. The final output is $(b_1 \oplus \overline{z_1}) \vee (b_2 \oplus \overline{z_2}) \vee (b_3 \oplus \overline{z_3})$.

Final algorithm: On input x, we get C_x, non-deterministically guess B_x, construct F_x and run the fast tautology algorithm on F_x.

Correctness: $x \in L \iff F_x$ is a tautology. The tautology algorithm on F_x checks if the non-deterministic guess B_x satisfies: $V(x, tt(B_x)) = 1$. If $x \notin L$, this is not possible for any B_x. If $x \in L$, this is true for a poly-size A-oracle circuit B_x, which exists due the easy-witness lemma for $(\mathsf{NP} \cap \mathsf{Co\text{-}NP})/poly$. □

9.4 Improving exhaustive search for $\mathsf{CAPP}^{\mathsf{NP} \cap \mathsf{Co\text{-}NP}}$ implies $\mathsf{NEXP} \not\subset (\mathsf{NP} \cap \mathsf{Co\text{-}NP})/poly$

In this section we show: super-polynomial savings in non-deterministic algorithms for the class $\mathsf{CAPP}^{\mathsf{NP} \cap \mathsf{Co\text{-}NP}}$, implies $\mathsf{NEXP} \not\subset (\mathsf{NP} \cap \mathsf{Co\text{-}NP})/poly$. We first state the following PCP verifier for NEXP, that we would need in our result.

Theorem 9.7 (see [12, 73]). *For any $L \in \mathsf{NTIME}(2^n)$, there exists a PCP verifier $V(x, y, r)$ with soundness $1/2$, perfect completeness, and randomness complexity $n + c \log n$, query complexity n^c, and verification time n^c, for some constant c. That means:*

- *V has random access to x and y, uses at most $|r| = n + c \log n$ random bits in any execution, makes n^c queries to the candidate proof y, and runs in at most n^c steps.*
- *if $x \in L$, $\exists y : |y| = n^c$ $Pr_r[V(x, y, r) = 1] = 1$.*
- *if $x \notin L$, $\forall y : |y| = n^c$ $Pr_r[V(x, y, r) = 1] \leq 1/2$.*

Now we give our result. The proof uses the same structure as the proof of Theorem 9.6. Their the 'easy-witness circuit' was queried by the 'reduction-circuit' three times. Here, the 'easy-witness circuit' will be queried polynomially-many times by the circuit that will capture the randomized verification procedure of the PCP verifier, that we get from Theorem 9.7.

Theorem 9.8. *For any super-polynomial function sp, an $\mathsf{NTIME}(2^n/sp(n))$ CAPP algorithm for n-input $poly(n)$-size A-oracle circuits, for every $A \in (\mathsf{NP} \cap \mathsf{Co\text{-}NP})$, implies $\mathsf{NEXP} \not\subset (\mathsf{NP} \cap \mathsf{Co\text{-}NP})/poly$.*

Proof. For $L \in \mathsf{NTIME}(2^n)$ we design an $\mathsf{NTIME}(2^n/sp(n))$ algorithm, under the assumption $\mathsf{NEXP} \subset (\mathsf{NP} \cap \mathsf{Co\text{-}NP})/poly$. This will contradict the non-deterministic time hierarchy from Theorem 9.4.

Reduction circuit: Let V be a PCP verifier for L from the Theorem 9.7. On any input x, $V(x, y, r)$ receives $|r| = n + c \log n$ random bits, makes oracle queries to the proof y of size $2^n n^c$, and runs for n^c-time. Let C_x be an oracle circuit capturing this computation. For the oracle gates, we will use copies of the following described 'easy-witness circuit' B_x.

Easy-witness circuit for an special verifier: Due to the same reasoning given in the proof of Theorem 9.6, we have an A-oracle circuit B_x whose truth-table is the witness for the non-deterministic verifier V' on input x: $V'(x, y)$ computes $V(x, y, r)$ on each value of r and outputs 1 if and only if $\forall r\ V(x, y, r) = 1$.

Final circuit F_x: $(n + c \log n)$-bits long input r is given to C_x. The oracle gates are replaced by the circuit B_x. The final output is the output of C_x.

Final algorithm: On input x, we get C_x, non-deterministically guess B_x, construct F_x and run the fast CAPP algorithm on F_x.

Correctness: $x \in L \iff F_x$ outputs 1 on more than 0.9 of its inputs. The CAPP algorithm on F_x checks if the non-deterministic guess B_x satisfies: $Pr_r\ [V(x, tt(B_x), r) = 1] \geq 0.9$ or $V'(x, tt(B_x)) = 1$. If $x \notin L$, this is not possible for any B_x. If $x \in L$, this is true for a poly-size A-oracle circuit B_x, which exists due the easy-witness lemma for $(\mathsf{NP} \cap \mathsf{Co\text{-}NP})/poly$. □

9.5 NSUBEXP TAUT algorithm for deterministic circuits implies NEXP $\not\subset$ (NP $\cap$ Co-NP)/*poly*

In this section, we extend the results from the previous sections, to deterministic circuits: fast TAUT algorithm for deterministic circuits implies NEXP $\not\subset$ (NP $\cap$ Co-NP)/*poly*.

Theorem 9.9. *$A \cap_{\epsilon>0}$ io-Heur-NTIME$(2^{n^\epsilon})/n^\epsilon$ tautology algorithm for n-input poly(n)-size deterministic circuits implies* NEXP $\not\subset$ (NP $\cap$ Co-NP)/*poly*.

Proof. The proof is very similar to the proof of Theorem 9.8.

We assume NEXP $\subset$ (NP $\cap$ Co-NP)/*poly*. Now for any $L \in$ NEXP we give a $\cap_{\epsilon>0}$ io-Heur-NTIME$(2^{n^\epsilon})/n^\epsilon$ algorithm. Since NEXP = EXP, NEXP $\subset \cap_{\epsilon>0}$ io-Heur-NTIME$(2^{n^\epsilon})/n^\epsilon$ is a contradiction from [75].

We use the same reduction circuit C_x and the same verifier V. Instead of using the (NP $\cap$ Co-NP)-oracle witness circuit B_x, we construct two witness circuits (after guessing the advice of the (NP $\cap$ Co-NP)/*poly* algorithm $\mathcal{A}$ that has V's oblivious-witnesses): one non-deterministic B^1_x, and one co-non-deterministic B^2_x.

Now for constructing the final circuit F_x: Take the reduction circuit C_x. C_x outputs three literals. Plug any positive literal into a copy of the co-non-deterministic circuit B^2_x, and any negative literal into a copy of the co-non-deterministic circuit $\overline{B^1_x}$. Output is the logical-or of the three copies used. To make the circuit deterministic, include the non-deterministic inputs of the copies of B^2_x and $\overline{B^1_x}$ into the actual input.

Final algorithm: Get C_x. Non-deterministically guess the advice for $\mathcal{A}$, and get B^1_x and B^2_x (that are guaranteed to have complementary truth-tables). Construct the deterministic circuit F_x as described above. Run the the fast TAUT algorithm on F_x.

Correctness: Its the same. We just replaced the function of the (NP $\cap$ Co-NP)-oracle witness circuit B_x with two co-non-deterministic circuits B^2_x and $\overline{B^1_x}$. So F_x becomes a co-non-deterministic circuit. And tautology of a co-non-deterministic circuit, is same as the tautology of the deterministic circuit we get after including the non-deterministic inputs into the actual input. So we modify F_x accordingly to make it deterministic. □

Using the same proof as above, we also get NEXP lower bounds for (NP $\cap$ Co-NP)/*poly* with sub-polynomial non-determinism, with only mildly-exponential savings in the tautology algorithm. This is because, now the conversion of the co-non-deterministic circuit into the deterministic circuit doesn't blow up the input size by much.

Corollary 9.10. *For any $\epsilon > 0$, an* NTIME$(2^{(1-\epsilon)n})$ *tautology algorithm for n-input poly(n)-size deterministic circuits implies* NEXP $\not\subset$ (NP $\cap$ Co-NP)/*poly with sub-polynomial non-determinism.*

9.6 New gap theorems for CAPP and MA

In the previous three sections we saw that, fast algorithms imply NEXP $\not\subset$ (NP $\cap$ Co-NP)/*poly*. In the Section 8 we saw that this lower bound is equivalent to fast algorithm for the class CAPP$^{\text{NP}\cap\text{Co-NP}}$. This gives us the following corollary.

Corollary 9.11 (Gap theorem for CAPP$^{\text{NP}\cap\text{Co-NP}}$). *Let sp be any super-polynomial function. Then a $\cap_{\epsilon>0}$ io-*NTIME$(2^{n^\epsilon})/n^\epsilon$ *algorithm for n-input poly(n)-size A-oracle circuits, for every $A \in$* (NP $\cap$ Co-NP)*, is implied by any of the following:*

(1) *an* NTIME$(2^n/sp(n))$ CAPP *algorithm for n-input poly(n)-size A-oracle circuits, for every $A \in$* (NP $\cap$ Co-NP)*;*
(2) *an* NTIME$(2^n/sp(n))$ TAUT *algorithm for n-input poly(n)-size A-oracle circuits, for every $A \in$* (NP $\cap$ Co-NP)*;*

(3) *a* $\cap_{\epsilon>0}$ *io-Heur-*$\mathsf{NTIME}(2^{n^\epsilon})/n^\epsilon$ TAUT *algorithm for n-input* $poly(n)$*-size deterministic circuits.*

In [31] they showed a gap theorem for MA: either MA is as powerful as NEXP, or can be derandomized in NSUBEXP (infinitely often, with sub-polynomial advice). From the arguments in Section 9.1 we can get an improved gap theorem where $\mathsf{MA} = \mathsf{EXP}^{\mathsf{NP}}_{||}$ in the first case. We also get a similar gap theorem for $\mathsf{MA}^{\mathsf{NP}\cap\mathsf{Co-NP}}$: either $\mathsf{MA}^{\mathsf{NP}\cap\mathsf{Co-NP}}$ is as powerful as $\mathsf{EXP}^{\mathsf{NP}}_{||}$, or can be derandomized in NSUBEXP (infinitely often, with sub-polynomial advice).

Corollary 9.12 (Gap theorem for MA). *Exactly one of the following statements is true:*

(1) $\mathsf{MA} = \mathsf{EXP}^{\mathsf{NP}}_{||}$

(2) $\mathsf{MA} \subset \cap_{\epsilon>0}$ *io-*$\mathsf{NTIME}(2^{n^\epsilon})/n^\epsilon$

Corollary 9.13 (Gap theorem for $\mathsf{MA}^{\mathsf{NP}\cap\mathsf{Co-NP}}$). *Exactly one of the following statements is true:*

(1) $\mathsf{MA}^{\mathsf{NP}\cap\mathsf{Co-NP}} = \mathsf{EXP}^{\mathsf{NP}}_{||}$

(2) $\mathsf{MA}^{\mathsf{NP}\cap\mathsf{Co-NP}} \subset \cap_{\epsilon>0}$ *io-*$\mathsf{NTIME}(2^{n^\epsilon})/n^\epsilon$

9.7 New ACC lower bounds

Let $\mathsf{ACC}(s)$, $\mathsf{NACC}(s)^p$, $\mathsf{SVACC}(s)^p$, $\mathsf{prSVACC}(s)^p$, denote the classes of s-size deterministic, non-deterministic, SV non-deterministic, and promise SV non-deterministic, ACC circuits respectively. The superscript p indicates the amount of non-determinism (the size of the non-deterministic inputs). We omit s or p, if they are $poly(n)$. $\mathsf{prSVACC}(s)^p$ are defined similar to prSVN circuits: the only difference is that the underlying prSV_1 algorithm outputs $\mathsf{SVACC}(s)^p$ circuits on all inputs.

In this section we say Γ doesn't have ACC circuits (or ACC witnesses), if a single language in Γ beats d-depth ACC circuits that use m-mod gates, for all constants d and m. We write lower bounds for all the other ACC variants similarly.

In [74] it was proved that NEXP doesn't have ACC circuits, and witnesses in ACC. In [75] it was improved to:

(1) $\mathsf{NEXP} \not\subset_w \cap_{\epsilon>0}\mathsf{ACC}(2^{n^\epsilon})$

(2) $\mathsf{NEXP} \not\subset \mathsf{ACC}(n^{\log n})$

The point (1) also gives that $\mathsf{NEXP} \not\subset_w \cap_{\epsilon>0}\mathsf{NACC}^{n^\epsilon}$: because $\mathsf{NACC}^{n^\epsilon}$ circuit can be converted into an $\mathsf{ACC}(poly(n)2^{n^\epsilon})$ circuit by evaluating the original circuit on each non-deterministic choice, and taking a logical-or of each evaluation.

Now, different types of EWL will give different types of lower bounds for NEXP:

(1) EWL for $\cap_{\epsilon>0}\mathsf{NACC}^{n^\epsilon}$ implies $\mathsf{NEXP} \not\subset \cap_{\epsilon>0}\mathsf{NACC}^{n^\epsilon}$;

(2) EWL for $\cap_{\epsilon>0}\mathsf{SVNACC}^{n^\epsilon}$ implies $\mathsf{NEXP} \not\subset \cap_{\epsilon>0}\mathsf{SVNACC}^{n^\epsilon}$; and

(3) EWL for $\cap_{\epsilon>0}\mathsf{prSVNACC}^{n^\epsilon}$ implies $\mathsf{NEXP} \not\subset \cap_{\epsilon>0}\mathsf{prSVNACC}^{n^\epsilon}$.

We establish the lower bound in point (3), and prove that the lower bounds in points (1) and (2) are equivalent. We use the following lemma in our proofs. The proof of the this lemma is similar to the proof of Lemma 5.1.

Lemma 9.14. *If* $\mathsf{NP} \subset \cap_{\epsilon>0}\mathsf{SVNACC}^{n^\epsilon}$*, then there exists a constant c such that: for large enough n, any* $\cap_{\epsilon>0}\mathsf{NSIZE}(s)^{n^\epsilon}$ *circuit has an equivalent* $\cap_{\epsilon>0}\mathsf{SVNACC}(s^c)^{n^\epsilon}$ *circuit.*

Proof. NCkt-Eval is a problem in NP whose input is a non-deterministic Boolean circuit C and a string x, and the output is the output of C on x. If $\mathsf{NP} \subset \cap_{\epsilon>0}\mathsf{SV\text{-}NACC}^{n^\epsilon}$, then there is a constant c such that NCkt-Eval has $n^{c/2}$-size $\cap_{\epsilon>0}\mathsf{SVNACC}^{n^\epsilon}$ circuits.

Let B be a $\cap_{\epsilon>0}\mathsf{NSIZE}(s)^{n^\epsilon}$ circuit. Let E be $(n + s\log s)^{c/2}$-size circuit corresponding to the $(n + s\log s)^{th}$-slice of NCkt-Eval. Define $D(x) = E(B, x)$. It is easy to check that: (i) D is an $\cap_{\epsilon>0}\mathsf{SVNACC}(s^c)^{n^\epsilon}$ circuit; and (ii) D is equivalent to B. □

Now using the above lemma, and the result from [75] that NEXP doesn't have oblivious-witnesses in $\cap_{\epsilon>0}\mathsf{NACC}^{n^\epsilon}$, we give the following results. Note that, the oblivious-witness lower bound obtained from Corollary 9.10 is also sufficient for these results.

Theorem 9.15. $\mathsf{NP} \subset \cap_{\epsilon>0}\mathsf{SVNACC}^{n^\epsilon} \implies \mathsf{NEXP} \not\subset \cap_{\epsilon>0}\mathsf{prSVNSIZE}(poly)^{n^\epsilon}$

Proof. We prove: $\mathsf{NP} \subset \cap_{\epsilon>0}\mathsf{NACC}^{n^\epsilon}$ and $\mathsf{NEXP} \subset \cap_{\epsilon>0}\mathsf{prSVNSIZE}(poly)^{n^\epsilon}$ leads to contradiction.

First we prove: $\mathsf{NEXP} \subset \cap_{\epsilon>0}\mathsf{prSVNSIZE}(poly)^{n^\epsilon}$ implies $\mathsf{NEXP} \subset_{ow} \cap_{\epsilon>0}\mathsf{prSVNSIZE}(poly)^{n^\epsilon}$. This proof is analogous to the EWL in the previous sections. Idea is, $\mathsf{NEXP} \subset \cap_{\epsilon>0}\mathsf{prSVNSIZE}(poly)^{n^\epsilon}$ implies $\mathsf{NEXP} = \mathsf{AM}$, and from [31] we get that the search version of NEXP is in EXP. Specifically, it is in $\cap_{\epsilon>0}\mathsf{prSVNSIZE}(poly)^{n^\epsilon}$, thus oblivious-witnesses are also in $\cap_{\epsilon>0}\mathsf{prSVNSIZE}(poly)^{n^\epsilon}$.

Now $\mathsf{NP} \subset \cap_{\epsilon>0}\mathsf{SVNACC}^{n^\epsilon}$, from the Lemma 9.14 implies that: any $\cap_{\epsilon>0}\mathsf{prSVNSIZE}(poly)^{n^\epsilon}$ circuit has an equivalent $\cap_{\epsilon>0}\mathsf{SVNACC}^{n^\epsilon}$ circuit.

Thus NEXP has oblivious-witnesses in $\cap_{\epsilon>0}\mathsf{SVNACC}^{n^\epsilon}$. We know this is not true from [75]. □

Corollary 9.16. $\mathsf{NEXP} \not\subset \cap_{\epsilon>0}\mathsf{prSVNACC}^{n^\epsilon}$

Theorem 9.17. $\mathsf{NEXP} \not\subset \cap_{\epsilon>0}\mathsf{SVNACC}^{n^\epsilon} \iff \mathsf{NEXP} \not\subset \cap_{\epsilon>0}\mathsf{NACC}^{n^\epsilon}$

Proof. Backward direction is trivial. For the forward direction, first we observe that: $\Sigma_2 \cap \Pi_2 \not\subset \cap_{\epsilon>0}\mathsf{SVNACC}^{n^\epsilon} \iff \Sigma_2 \cap \Pi_2 \not\subset \cap_{\epsilon>0}\mathsf{NACC}^{n^\epsilon}$. Now, if $\Sigma_2 \cap \Pi_2 \not\subset \cap_{\epsilon>0}\mathsf{NACC}^{n^\epsilon}$, we are done. Else, $\mathsf{NP} \subset \cap_{\epsilon>0}\mathsf{SVNACC}^{n^\epsilon}$. Now, if $\mathsf{NEXP} \subset \cap_{\epsilon>0}\mathsf{NACC}^{n^\epsilon}$, then from the Lemma 9.14 we contradict our assumption and get: $\mathsf{NEXP} \subset \cap_{\epsilon>0}\mathsf{SVNACC}^{n^\epsilon}$. □

10 CONCLUSIONS AND OPEN PROBLEMS

The main open problem is whether there are any connections between fast algorithms and non-uniform lower bounds possible within deterministic classes such as EXP. In almost all of the prior connections, non-uniformity is simulated with non-determinism, by having a non-deterministic machine guess the appropriate circuit. Can we substitute a recursive argument for non-determinism here? Our results show that, while still allowing non-determinism, the form of non-determinism can be restricted. In what other ways could we get such connections for smaller classes by restricting the use of non-determinism? The circuit model combines two features: time and non-uniformity. Can we get a fine-grained version of easy-witness lemma by distinguishing these two parameters?

Our results also show that, if we are using unrestricted non-determinism to simulate non-uniformity, we can extract more out of it. That is, the guessed circuit is also allowed to use non-determinism that is promise-single-valued. In what other ways can we extend this allowance? Specifically, can we prove NEXP EWL and KLT for circuit classes above $(\mathsf{NP} \cap \mathsf{Co\text{-}NP})/poly$? Which of the inclusions, $\mathsf{MA} \subseteq \mathsf{MA}^{\mathsf{NP}\cap\mathsf{Co\text{-}NP}} \subseteq \mathsf{AM} \subseteq \mathsf{M}(\mathsf{AM}||\mathsf{Co\text{-}NP}) \subseteq \mathsf{MA}^{\mathsf{NP}}$, are equalities? Where exactly the non-deterministic circuit class lies, whose NEXP lower bound is equivalent to $\mathsf{NEXP} \neq \mathsf{AM}$?

We also show unconditional ACC lower bounds where sub-polynomial promise-single-valued non-determinism is allowed. Can we increase the amount of non-determinism allowed, to polynomial or linear? Designing fast algorithms for ACC is one direct strategy. Can we remove the 'promise' condition? Designing NEXP EWL for higher circuit classes is one direct strategy.

In the inclusion chain, $\mathsf{EXP} \subseteq \text{ip-}\mathsf{ZNEXP} \subseteq \mathsf{NEXP} \subseteq \mathsf{EXP}^{\mathsf{NP}}_{||} \subseteq \mathsf{EXP}^{\mathsf{NP}}$, we have KLTs for the all the classes. The KLT for ip-ZNEXP is only for fix-polynomial upper bounds. Can we improve it to polynomial upper bounds? Except $\mathsf{EXP}^{\mathsf{NP}}$, we have established equivalences between lower bounds and derandomization of certain probabilistic classes. Can we get such equivalences for $\mathsf{EXP}^{\mathsf{NP}}$?

ACKNOWLEDGMENTS

This work is supported by the Simons Foundation and NSF grant CCF-1909634. We want to thank Sasank Mouli, Marco Carmosino and Sam McGuire for useful discussions, and for comments and corrections on the manuscript.

REFERENCES

[1] Manindra Agrawal and Thomas Thierauf. 1996. The Boolean Isomorphism Problem. In *37th Annual Symposium on Foundations of Computer Science, FOCS '96, Burlington, Vermont, USA, 14-16 October, 1996*. IEEE Computer Society, 422–430. https://doi.org/10.1109/SFCS.1996.548501

[2] Eric Allender. 1986. The Complexity of Sparse Sets in P. In *Structure in Complexity Theory, Proceedings of the Conference hold at the University of California, Berkeley, California, USA, June 2-5, 1986*. 1–11. https://doi.org/10.1007/3-540-16486-3_85

[3] Eric Allender. 2001. When Worlds Collide: Derandomization, Lower Bounds, and Kolmogorov Complexity. In *FST TCS 2001: Foundations of Software Technology and Theoretical Computer Science*, Ramesh Hariharan, V. Vinay, and Madhavan Mukund (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 1–15.

[4] Dana Angluin. 1987. Queries and Concept Learning. *Machine Learning* 2, 4 (1987), 319–342. https://doi.org/10.1007/BF00116828

[5] Sanjeev Arora and Boaz Barak. 2009. *Computational Complexity: A Modern Approach* (1st ed.). Cambridge University Press, New York, NY, USA.

[6] Vikraman Arvind and Johannes Köbler. 1997. On Resource-Bounded Measure and Pseudorandomness. In *Foundations of Software Technology and Theoretical Computer Science, 17th Conference, Kharagpur, India, December 18-20, 1997, Proceedings (Lecture Notes in Computer Science)*, S. Ramesh and G. Sivakumar (Eds.), Vol. 1346. Springer, 235–249. https://doi.org/10.1007/BFb0058034

[7] Baris Aydinlioglu and Dieter van Melkebeek. 2017. Nondeterministic circuit lower bounds from mildly derandomizing Arthur-Merlin games. *Computational Complexity* 26, 1 (2017), 79–118. https://doi.org/10.1007/s00037-014-0095-y

[8] L Babai. 1985. Trading Group Theory for Randomness. In *Proceedings of the Seventeenth Annual ACM Symposium on Theory of Computing* (Providence, Rhode Island, USA) *(STOC '85)*. Association for Computing Machinery, New York, NY, USA, 421–429. https://doi.org/10.1145/22145.22192

[9] László Babai, Lance Fortnow, Noam Nisan, and Avi Wigderson. 1993. BPP Has Subexponential Time Simulations Unless EXPTIME has Publishable Proofs. *Computational Complexity* 3 (1993), 307–318. https://doi.org/10.1007/BF01275486

[10] László Babai and Shlomo Moran. 1988. Arthur-Merlin Games: A Randomized Proof System, and a Hierarchy of Complexity Classes. *J. Comput. Syst. Sci.* 36, 2 (1988), 254–276. https://doi.org/10.1016/0022-0000(88)90028-1

[11] Donald Beaver and Joan Feigenbaum. 1990. Hiding Instances in Multioracle Queries. In *STACS 90, 7th Annual Symposium on Theoretical Aspects of Computer Science, Rouen, France, February 22-24, 1990, Proceedings (Lecture Notes in Computer Science)*, Christian Choffrut and Thomas Lengauer (Eds.), Vol. 415. Springer, 37–48. https://doi.org/10.1007/3-540-52282-4_30

[12] Eli Ben-Sasson, Oded Goldreich, Prahladh Harsha, Madhu Sudan, and Salil P. Vadhan. 2005. Short PCPs Verifiable in Polylogarithmic Time. In *20th Annual IEEE Conference on Computational Complexity (CCC 2005), 11-15 June 2005, San Jose, CA, USA*. IEEE Computer Society, 120–134. https://doi.org/10.1109/CCC.2005.27

[13] Nader H. Bshouty, Richard Cleve, Ricard Gavaldà, Sampath Kannan, and Christino Tamon. 1996. Oracles and Queries That Are Sufficient for Exact Learning. *J. Comput. Syst. Sci.* 52, 3 (1996), 421–433. https://doi.org/10.1006/jcss.1996.0032

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

[15] Harry Buhrman and Steven Homer. 1992. Superpolynomial Circuits, Almost Sparse Oracles and the Exponential Hierarchy. In *Foundations of Software Technology and Theoretical Computer Science, 12th Conference, New Delhi, India, December 18-20, 1992, Proceedings (Lecture Notes in Computer Science)*, R. K. Shyamasundar (Ed.), Vol. 652. Springer, 116–127. https://doi.org/10.1007/3-540-56287-7_99

[16] Jin-yi Cai. 2001. S^P_2 subseteq ZPP^{NP}. In *42nd Annual Symposium on Foundations of Computer Science, FOCS 2001, 14-17 October 2001, Las Vegas, Nevada, USA*. IEEE Computer Society, 620–629. https://doi.org/10.1109/SFCS.2001.959938

[17] Jin-Yi Cai, Venkatesan T. Chakaravarthy, Lane A. Hemaspaandra, and Mitsunori Ogihara. 2005. Competing provers yield improved Karp–Lipton collapse results. *Information and Computation* 198, 1 (2005), 1 – 23. https://doi.org/10.1016/j.ic.2005.01.002

[18] Marco Carmosino, Russell Impagliazzo, Valentine Kabanets, and Antonina Kolokolova. 2015. Tighter Connections between Derandomization and Circuit Lower Bounds. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2015, August 24-26, 2015, Princeton, NJ, USA (LIPIcs)*, Naveen

Garg, Klaus Jansen, Anup Rao, and José D. P. Rolim (Eds.), Vol. 40. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 645–658. https://doi.org/10.4230/LIPIcs.APPROX-RANDOM.2015.645

[19] Marco L. Carmosino, Jiawei Gao, Russell Impagliazzo, Ivan Mihajlin, Ramamohan Paturi, and Stefan Schneider. 2016. Nondeterministic Extensions of the Strong Exponential Time Hypothesis and Consequences for Non-reducibility. In *Proceedings of the 2016 ACM Conference on Innovations in Theoretical Computer Science, Cambridge, MA, USA, January 14-16, 2016*, Madhu Sudan (Ed.). ACM, 261–270. https://doi.org/10.1145/2840728.2840746

[20] Lijie Chen, Dylan M. McKay, Cody D. Murray, and R. Ryan Williams. 2019. Relations and Equivalences Between Circuit Lower Bounds and Karp-Lipton Theorems. In *34th Computational Complexity Conference, CCC 2019, July 18-20, 2019, New Brunswick, NJ, USA (LIPIcs)*, Amir Shpilka (Ed.), Vol. 137. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 30:1–30:21. https://doi.org/10.4230/LIPIcs.CCC.2019.30

[21] Stephen A. Cook. 1971. The Complexity of Theorem-Proving Procedures. In *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing, May 3-5, 1971, Shaker Heights, Ohio, USA*. 151–158. https://doi.org/10.1145/800157.805047

[22] Joan Feigenbaum and Lance Fortnow. 1993. Random-Self-Reducibility of Complete Sets. *SIAM J. Comput.* 22, 5 (1993), 994–1005. https://doi.org/10.1137/0222061

[23] Lance Fortnow and Adam R. Klivans. 2005. NP with Small Advice. In *20th Annual IEEE Conference on Computational Complexity (CCC 2005), 11-15 June 2005, San Jose, CA, USA*. IEEE Computer Society, 228–234. https://doi.org/10.1109/CCC.2005.15

[24] Lance Fortnow and Adam R. Klivans. 2009. Efficient learning algorithms yield circuit lower bounds. *J. Comput. Syst. Sci.* 75, 1 (2009), 27–36. https://doi.org/10.1016/j.jcss.2008.07.006

[25] Lance Fortnow, Richard Lipton, Dieter van Melkebeek, and Anastasios Viglas. 2005. Time-space Lower Bounds for Satisfiability. *J. ACM* 52, 6 (Nov. 2005), 835–865. https://doi.org/10.1145/1101821.1101822

[26] Lance Fortnow, Rahul Santhanam, and Luca Trevisan. 2005. Hierarchies for Semantic Classes. In *Proceedings of the Thirty-seventh Annual ACM Symposium on Theory of Computing* (Baltimore, MD, USA) *(STOC '05)*. ACM, New York, NY, USA, 348–355. https://doi.org/10.1145/1060590.1060642

[27] L. Fortnow, R. Santhanam, and R. Williams. 2009. Fixed-Polynomial Size Circuit Bounds. In *2009 24th Annual IEEE Conference on Computational Complexity*. 19–26.

[28] Oded Goldreich and David Zuckerman. 1997. Another proof that BPP subseteq PH (and more). *Electronic Colloquium on Computational Complexity (ECCC)* 4, 45 (1997). http://eccc.hpi-web.de/eccc-reports/1997/TR97-045/index.html

[29] Ryan C. Harkins and John M. Hitchcock. 2013. Exact Learning Algorithms, Betting Games, and Circuit Lower Bounds. *TOCT* 5, 4 (2013), 18:1–18:11. https://doi.org/10.1145/2539126.2539130

[30] Russell Impagliazzo, Valentine Kabanets, and Ilya Volkovich. 2018. The Power of Natural Properties as Oracles. In *33rd Computational Complexity Conference, CCC 2018, June 22-24, 2018, San Diego, CA, USA (LIPIcs)*, Rocco A. Servedio (Ed.), Vol. 102. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 7:1–7:20. https://doi.org/10.4230/LIPIcs.CCC.2018.7

[31] Russell Impagliazzo, Valentine Kabanets, and Avi Wigderson. 2002. In search of an easy witness: exponential time vs. probabilistic polynomial time. *J. Comput. Syst. Sci.* 65, 4 (2002), 672–694. https://doi.org/10.1016/S0022-0000(02)00024-7

[32] Russell Impagliazzo and Ramamohan Paturi. 1999. Complexity of k-SAT. In *Proceedings of the 14th Annual IEEE Conference on Computational Complexity, Atlanta, Georgia, USA, May 4-6, 1999*. 237–240. https://doi.org/10.1109/CCC.1999.766282

[33] Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. 1998. Which Problems Have Strongly Exponential Complexity?. In *39th Annual Symposium on Foundations of Computer Science, FOCS '98, November 8-11, 1998, Palo Alto, California, USA*. 653–663. https://doi.org/10.1109/SFCS.1998.743516

[34] Russell Impagliazzo and Gábor Tardos. 1989. Decision Versus Search Problems in Super-Polynomial Time. In *30th Annual Symposium on Foundations of Computer Science, Research Triangle Park, North Carolina, USA, 30 October - 1 November 1989*. IEEE Computer Society, 222–227. https://doi.org/10.1109/SFCS.1989.63482

[35] Russell Impagliazzo and Avi Wigderson. 1997. *P = BPP* if *E* Requires Exponential Circuits: Derandomizing the XOR Lemma. In *Proceedings of the Twenty-Ninth Annual ACM Symposium on the Theory of Computing, El Paso, Texas, USA, May 4-6, 1997*. 220–229. https://doi.org/10.1145/258533.258590

[36] Russell Impagliazzo and Avi Wigderson. 2001. Randomness vs Time: Derandomization under a Uniform Assumption. *J. Comput. Syst. Sci.* 63, 4 (2001), 672–688. https://doi.org/10.1006/jcss.2001.1780

[37] Hamid Jahanjou, Eric Miles, and Emanuele Viola. 2015. Local Reductions. In *Automata, Languages, and Programming - 42nd International Colloquium, ICALP 2015, Kyoto, Japan, July 6-10, 2015, Proceedings, Part I*. 749–760. https://doi.org/10.1007/978-3-662-47672-7_61

[38] Jin-yicai, Denischarles, A.pavan, and Samiksengupta. 2011. ON HIGHER ARTHUR-MERLIN CLASSES. *International Journal of Foundations of Computer Science* 15 (11 2011). https://doi.org/10.1142/S0129054104002273

[39] Stanislav Žák. 1983. A Turing machine time hierarchy. *Theoretical Computer Science* 26, 3 (1983), 327 – 333. https://doi.org/10.1016/0304-3975(83)90015-4

[40] Valentine Kabanets. 2001. Easiness Assumptions and Hardness Tests: Trading Time for Zero Error. *J. Comput. Syst. Sci.* 63, 2 (2001), 236–252. https://doi.org/10.1006/jcss.2001.1763

[41] Valentine Kabanets and Russell Impagliazzo. 2004. Derandomizing Polynomial Identity Tests Means Proving Circuit Lower Bounds. *Computational Complexity* 13, 1-2 (2004), 1–46. https://doi.org/10.1007/s00037-004-0182-6

[42] Ravi Kannan. 1982. Circuit-Size Lower Bounds and Non-Reducibility to Sparse Sets. *Information and Control* 55, 1-3 (1982), 40–56. https://doi.org/10.1016/S0019-9958(82)90382-5

[43] Richard M. Karp and Richard J. Lipton. 1980. Some Connections Between Nonuniform and Uniform Complexity Classes. In *Proceedings of the Twelfth Annual ACM Symposium on Theory of Computing* (Los Angeles, California, USA) *(STOC '80)*. ACM, New York, NY, USA, 302–309. https://doi.org/10.1145/800141.804678

[44] Johannes Köbler and Osamu Watanabe. 1998. New Collapse Consequences of NP Having Small Circuits. *SIAM J. Comput.* 28, 1 (1998), 311–324. https://doi.org/10.1137/S0097539795296206 arXiv:https://doi.org/10.1137/S0097539795296206

[45] Adam R. Klivans, Pravesh Kothari, and Igor Carboni Oliveira. 2013. Constructing Hard Functions Using Learning Algorithms. In *Proceedings of the 28th Conference on Computational Complexity, CCC 2013, K.lo Alto, California, USA, 5-7 June, 2013*. 86–97. https://doi.org/10.1109/CCC.2013.18

[46] Adam R. Klivans and Dieter van Melkebeek. 2002. Graph Nonisomorphism Has Subexponential Size Proofs Unless the Polynomial-Time Hierarchy Collapses. *SIAM J. Comput.* 31, 5 (2002), 1501–1526. https://doi.org/10.1137/S0097539700389652

[47] Ker-I Ko. 1982. Some observations on the probabilistic algorithms and NP-hard problems. *Inform. Process. Lett.* 14, 1 (1982), 39 – 43. https://doi.org/10.1016/0020-0190(82)90139-9

[48] Matthias Krause and Stefan Lucks. 2002. Pseudorandom Functions in TC0 and Cryptographic Limitations to Proving Lower Bounds. *Comput. Complex.* 10, 4 (May 2002), 297–313. https://doi.org/10.1007/s000370100002

[49] Leonid A. Levin. 1973. Universal sorting problems. *Problems of Information Transmission* 9 (1973), 265–266.

[50] Richard J. Lipton. 1989. New Directions In Testing. In *Distributed Computing And Cryptography, Proceedings of a DIMACS Workshop, Princeton, New Jersey, USA, October 4-6, 1989 (DIMACS Series in Discrete Mathematics and Theoretical Computer Science)*, Joan Feigenbaum and Michael Merritt (Eds.), Vol. 2. DIMACS/AMS, 191–202. https://doi.org/10.1090/dimacs/002/13

[51] Carsten Lund, Lance Fortnow, Howard J. Karloff, and Noam Nisan. 1992. Algebraic Methods for Interactive Proof Systems. *J. ACM* 39, 4 (1992), 859–868. https://doi.org/10.1145/146585.146605

[52] Eric Miles and Emanuele Viola. 2012. Substitution-Permutation Networks, Pseudorandom Functions, and Natural Proofs. In *Advances in Cryptology – CRYPTO 2012*, Reihaneh Safavi-Naini and Ran Canetti (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 68–85.

[53] Cody Murray and R. Ryan Williams. 2018. Circuit lower bounds for nondeterministic quasi-polytime: an easy witness lemma for NP and NQP. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25-29, 2018*. 890–901. https://doi.org/10.1145/3188745.3188910

[54] Moni Naor and Omer Reingold. 2004. Number-theoretic Constructions of Efficient Pseudo-random Functions. *J. ACM* 51, 2 (March 2004), 231–262. https://doi.org/10.1145/972639.972643

[55] Noam Nisan. 1991. Pseudorandom bits for constant depth circuits. *Combinatorica* 11, 1 (1991), 63–70. https://doi.org/10.1007/BF01375474

[56] Noam Nisan and Avi Wigderson. 1994. Hardness vs Randomness. *J. Comput. Syst. Sci.* 49, 2 (1994), 149–167. https://doi.org/10.1016/S0022-0000(05)80043-1

[57] Igor Carboni Oliveira. 2013. Algorithms versus Circuit Lower Bounds. *CoRR* abs/1309.0249 (2013). arXiv:1309.0249 http://arxiv.org/abs/1309.0249

[58] Igor Carboni Oliveira and Rahul Santhanam. 2017. Conspiracies Between Learning Algorithms, Circuit Lower Bounds, and Pseudorandomness. In *32nd Computational Complexity Conference, CCC 2017, July 6-9, 2017, Riga, Latvia*. 18:1–18:49. https://doi.org/10.4230/LIPIcs.CCC.2017.18

[59] Ramamohan Paturi, Pavel Pudlák, Michael E. Saks, and Francis Zane. 2005. An improved exponential-time algorithm for *k*-SAT. *J. ACM* 52, 3 (2005), 337–364. https://doi.org/10.1145/1066100.1066101

[60] Ramamohan Paturi, Pavel Pudlák, and Francis Zane. 1999. Satisfiability Coding Lemma. *Chicago J. Theor. Comput. Sci.* 1999 (1999). http://cjtcs.cs.uchicago.edu/articles/1999/11/contents.html

[61] Alexander A Razborov and Steven Rudich. 1997. Natural Proofs. *J. Comput. System Sci.* 55, 1 (1997), 24 – 35. https://doi.org/10.1006/jcss.1997.1494

[62] Alexander Russell and Ravi Sundaram. 1998. Symmetric Alternation Captures BPP. *Comput. Complex.* 7, 2 (Nov. 1998), 152–162. https://doi.org/10.1007/s000370050007

[63] Rahul Santhanam. 2009. Circuit Lower Bounds for Merlin–Arthur Classes. *SIAM J. Comput.* 39, 3 (2009), 1038–1061. https://doi.org/10.1137/070702680

[64] Ronen Shaltiel and Christopher Umans. 2006. Pseudorandomness for Approximate Counting and Sampling. *Computational Complexity* 15, 4 (2006), 298–341. https://doi.org/10.1007/s00037-007-0218-9

[65] Donald M. Stull. 2017. Some Results on Circuit Lower Bounds and Derandomization of Arthur-Merlin Problems. *CoRR* abs/1701.04428 (2017). arXiv:1701.04428 http://arxiv.org/abs/1701.04428

[66] Madhu Sudan, Luca Trevisan, and Salil P. Vadhan. 2001. Pseudorandom Generators without the XOR Lemma. *J. Comput. Syst. Sci.* 62, 2 (2001), 236–266. https://doi.org/10.1006/jcss.2000.1730

[67] Iannis Tourlakis. 2001. Time–Space Tradeoffs for SAT on Nonuniform Machines. *J. Comput. System Sci.* 63, 2 (2001), 268 – 287. https://doi.org/10.1006/jcss.2001.1767

[68] L. Trevisan and S. Vadhan. 2002. Pseudorandomness and average-case complexity via uniform reductions. In *Proceedings 17th IEEE Annual Conference on Computational Complexity*. 129–138.

[69] Christopher Umans. 2003. Pseudo-random generators for all hardnesses. *J. Comput. Syst. Sci.* 67, 2 (2003), 419–440. https://doi.org/10.1016/S0022-0000(03)00046-1

[70] Leslie G. Valiant. 1984. A Theory of the Learnable. *Commun. ACM* 27, 11 (1984), 1134–1142. https://doi.org/10.1145/1968.1972

[71] N.V. Vinodchandran. 2005. A note on the circuit complexity of PP. *Theoretical Computer Science* 347, 1 (2005), 415 – 418. https://doi.org/10.1016/j.tcs.2005.07.032

[72] N. V. Vinodchandran. 2004. $\mathrm{AM}_{\mathrm{exp}}$[nsube](NP[cap]coNP)/poly. *Inf. Process. Lett.* 89, 1 (2004), 43–47. https://doi.org/10.1016/j.ipl.2003.09.011

[73] Ryan Williams. 2013. Improving Exhaustive Search Implies Superpolynomial Lower Bounds. *SIAM J. Comput.* 42, 3 (2013), 1218–1244. https://doi.org/10.1137/10080703X

[74] Ryan Williams. 2014. Nonuniform ACC Circuit Lower Bounds. *J. ACM* 61, 1 (2014), 2:1–2:32. https://doi.org/10.1145/2559903

[75] R. Ryan Williams. 2016. Natural Proofs versus Derandomization. *SIAM J. Comput.* 45, 2 (2016), 497–529. https://doi.org/10.1137/130938219

[76] R. Ryan Williams. 2018. New Algorithms and Lower Bounds for Circuits With Linear Threshold Gates. *Theory of Computing* 14, 1 (2018), 1–25. https://doi.org/10.4086/toc.2018.v014a017

[77] Stathis Zachos and Martin Fürer. 1987. Probabalistic Quantifiers vs. Distrustful Adversaries. In *Foundations of Software Technology and Theoretical Computer Science, Seventh Conference, Pune, India, December 17-19, 1987, Proceedings (Lecture Notes in Computer Science)*, Kesav V. Nori (Ed.), Vol. 287. Springer, 443–455. https://doi.org/10.1007/3-540-18625-5_67

ECCC ISSN 1433-8092

https://eccc.weizmann.ac.il