Stretching Demi-Bits and Nondeterministic-Secure Pseudorandomness

Iddo Tzameret
Department of Computing, Imperial College London, UK

Lu-Ming Zhang
Department of Mathematics, London School of Economic and Political Science, UK

Abstract

We develop the theory of cryptographic nondeterministic-secure pseudorandomness beyond the point reached by Rudich's original work [25], and apply it to draw new consequences in average-case complexity and proof complexity. Specifically, we show the following:

Demi-bit stretch: Super-bits and demi-bits are variants of cryptographic pseudorandom generators which are secure against nondeterministic statistical tests [25]. They were introduced to rule out certain approaches to proving strong complexity lower bounds beyond the limitations set out by the Natural Proofs barrier of Razborov and Rudich [23]. Whether demi-bits are stretchable at all had been an open problem since their introduction. We answer this question affirmatively by showing that: every demi-bit $b : \{0,1\}^n \to \{0,1\}^{n+1}$ can be stretched into sublinear many demi-bits $b' : \{0,1\}^n \to \{0,1\}^{n+n^c}$, for every constant $0 < c < 1$.

Average-case hardness: Using work by Santhanam [26], we apply our results to obtain new average-case Kolmogorov complexity results: we show that $\mathrm{K}^{\mathsf{poly}}[n - O(1)]$ is zero-error average-case hard against $\mathsf{NP}/\mathsf{poly}$ machines iff $\mathrm{K}^{\mathsf{poly}}[n - o(n)]$ is, where for a function $s(n) : \mathbb{N} \to \mathbb{N}$, $\mathrm{K}^{\mathsf{poly}}[s(n)]$ denotes the languages of all strings $x \in \{0,1\}^n$ for which there are (fixed) polytime Turing machines of description-length at most $s(n)$ that output x.

Characterising super-bits by nondeterministic unpredictability: In the deterministic setting, Yao [31] proved that super-polynomial hardness of pseudorandom generators is equivalent to ("next-bit") *unpredictability*. Unpredictability roughly means that given any strict prefix of a random string, it is infeasible to predict the next bit. We initiate the study of unpredictability beyond the deterministic setting (in the cryptographic regime), and characterise the nondeterministic hardness of generators from an unpredictability perspective. Specifically, we propose four stronger notions of unpredictability: $\mathsf{NP}/\mathsf{poly}$-unpredictability, $\mathsf{coNP}/\mathsf{poly}$-unpredictability, $\cap$-unpredictability and $\cup$-unpredictability, and show that super-polynomial nondeterministic hardness of generators lies between $\cap$-unpredictability and $\cup$-unpredictability.

Characterising super-bits by nondeterministic hard-core predicates: We introduce a nondeterministic variant of hard-core predicates, called *super-core predicates*. We show that the existence of a super-bit is equivalent to the existence of a super-core of some non-shrinking function. This serves as an analogue of the equivalence between the existence of a strong pseudorandom generator and the existence of a hard-core of some one-way function [8, 12], and provides a first alternative characterisation of super-bits. We also prove that a certain class of functions, which may have hard-cores, cannot possess any super-core.

2012 ACM Subject Classification Theory of computation → Pseudorandomness and derandomization; Theory of computation → Computational complexity and cryptography

Keywords and phrases Pseudorandomness, Cryptography, Natural Proofs, Nondeterminism, Lower bounds

Digital Object Identifier 10.4230/LIPIcs.ITCS.2024.95

Related Version *Full Version*: `https://arxiv.org/abs/2304.14700`

Funding Part of this project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No. 101002742).

Iddo Tzameret: Part of this work was done at Simons Institute for the Theory of Computing, UC Berkeley.
Lu-Ming Zhang: Part of this work was done while at Imperial College London.

Acknowledgements We are indebted to Jan Pich who suggested looking at demi-bits and provided many clarifications regarding his work [21]. We are grateful to Rahul Santhanam for very useful discussions and specifically mentioning the potential application appearing in Theorem 28. Finally, we wish to thank Hanlin Ren for very useful comments on the manuscript as well as Oliver Korten and Yufeng Li for further discussions.

1 Introduction

Pseudorandomness is a natural concept allowing to measure the extent to which a resource-bounded computational machine can identify true random sources. It is an important notion in algorithms, enabling to derandomize efficient probabilistic algorithms by simulating many true random bits using fewer random bits. Another important aspect of pseudorandomness lies in computational complexity and cryptography, and specifically computational lower bounds, where it serves as the foundation of many results in cryptography, hardness vs. randomness trade-offs, and several barriers to proving strong computational lower bounds. Here, we will mostly be interested in the latter aspect of barrier results.

1.1 The theory of nondeterministic-secure pseudorandomness

Razborov and Rudich established a connection between pseudorandomness and the ability to prove boolean circuit lower bounds in their Natural Proofs paper [23]. They showed that most lower bounds arguments in circuit complexity contain (possibly implicitly) an efficient algorithm for deciding hardness, in the following sense: given the truth table of a boolean function, the algorithm determines if the function possesses some (combinatorial) properties that imply it is hard for a certain given circuit class (they called this "constructivity" and "usefulness" of a lower bound argument). They moreover showed that this algorithm identifies correctly the hardness of a non-negligible fraction of functions (which is called "largeness" in [23]). On the other hand, such an efficient algorithm for determining the hardness of boolean functions (for general circuits) would contradict reasonable assumptions in pseudorandomness, namely the existence of strong pseudorandom generators. This puts a *barrier*, so to speak, against the ability to prove lower bounds using natural proofs.

The notion of natural proofs has had a great influence on computational complexity theory. However, the fact that the plausible nonexistence of certain classes of natural proofs provides an obstacle against *very constructive* lower bound arguments (namely, those arguments that implicitly contain an efficient algorithm to determine when a function is hard) is somewhat less desirable. One would hope to extend the obstacle to less constructive proofs, for instance, proofs whose arguments contain implicitly only short *witnesses* for the hardness of functions.

Indeed, the notion of natural proofs comes to explain the difficulty in *proving* lower bounds, not in efficiently *deciding* hardness of boolean functions. For these reasons, among others, Rudich [25] set to extend the natural proofs barrier so that they encompass non-constructive arguments, namely, arguments implicitly using efficient *witnesses* of the hardness of boolean functions, in contrast to deterministic algorithms. This was done by extending the notion of pseudorandom generators so that they are secure against *nondeterministic* adversaries.

While empirically most known lower bound proofs were shown, at least implicitly, to fall within the scope of P/poly-constructive natural proofs, it is definitely conceivable and natural to assume that some lower bound approaches will necessitate $\mathsf{NP}/\mathsf{poly}$-constructive natural proofs. In fact, it is a very interesting open problem in itself to find such an NP/poly-constructive lower bound proof method. Furthermore, in works in proof complexity the role of nondeterministic-secure pseudorandomness is important (cf. recent work by Pich and Santhanam [22], as well as Krajíček [16]). Also, note that when dealing with the notion of barriers we refer to impossibility results and so it cannot always be expected to come up with good examples of proof methods we wish to rule out.

Accordingly, to extend the natural proofs barrier, Rudich introduced two primitives: *super-bits,* and its weaker variant *demi-bits.* Super-bits and demi-bits are (non-uniform) nondeterministic variants of strong pseudorandom generators (PRGs). They are secure against nondeterministic adversaries (i.e., adversaries in NP/poly). Demi-bits require their nondeterministic adversaries to break them in a *stronger* sense than super-bits, and hence their existence constitutes a better (i.e., *weaker*) assumption than the existence of super-bits on which to base barrier results; and this is one reason why the concept of demi-bits is important.

More specifically, super-bits and demi-bits both require a nondeterministic adversary to meaningfully distinguish truly random strings from pseudorandom ones by certifying truly random ones (i.e., an adversary outputs 1 if it thinks a given string is an output of a truly random process and 0 if it is the output of a pseudorandom generator). Thus, a nondeterministic distinguisher cannot break super-bits nor demi-bits by simply guessing a seed of the generator. Precisely, this is guaranteed as follows: for demi-bits we insist that strings in the image of the pseudorandom generator are always *rejected*, while for super-bits we allow some such pseudorandom strings to be accepted but we insist that many more strings outside the image of the pseudorandom generator are accepted (than strings in the image of the pseudorandom generator).

Formally, we have the following (all the distributions we consider in this work are, by default, uniform, unless otherwise stated, and U_n denotes the uniform distribution over $\{0,1\}^n$):

▶ **Definition 1** (Nondeterministic hardness [25]). *Let $g_n : \{0,1\}^n \to \{0,1\}^{l(n)}$, with $l(n) > n$, be a function in* P/poly*. We call such a function a* ***generator****. The* ***nondeterministic hardness*** *$H_{\mathrm{nh}}(g_n)$ (also called* ***super-hardness****) of g_n is the minimal s for which there exists a nondeterministic circuit D of size at most s such that*

$$\underset{y\in\{0,1\}^{l(n)}}{\mathbb{P}}[D(y)=1] - \underset{x\in\{0,1\}^{n}}{\mathbb{P}}[D(g_n(x))=1] \geq \frac{1}{s}. \tag{1}$$

In contrast to the standard definition of (deterministic) hardness (Definition 12), the order of the two possibilities on the left-hand side is crucial. This order forces a nondeterministic distinguisher to certify the randomness of a given input. Reversing the order, or adding an absolute value to left-hand side, trivializes (as in standard PRGs) the task of breaking g: a distinguisher D can simply guess a seed x and check if $g(x)$ equals the given input. For such a D, we have $\mathbb{P}[D(g(x))=1]=1$ and $\mathbb{P}[D(y)=1] \leq 1/2$.

Super-bits are exponentially super-hard generators (note that we call a growth rate of 2^{n^ε}, for a constant ε, *exponential*, and $\bigcap_{0<\varepsilon<1} 2^{n^\varepsilon}$ *sub-exponential*, while in cryptography the former is sometimes called a *sub-exponential* growth rate):

▶ **Definition 2** (Super-bits [25]). *A generator $g : \{0,1\}^n \to \{0,1\}^{n+c}$ (computable in* P/poly*), for some $c : \mathbb{N} \to \mathbb{N}$, is called c* ***super-bit(s)*** *(or a c-super-bit(s)) if $H_{\mathrm{nh}}(g) \geq 2^{n^\varepsilon}$ for some constant $\varepsilon > 0$ and all sufficiently large n's. In particular, if $c = 1$, we call g a super-bit.*

Many candidates of strong PRGs (against deterministic machines) were constructed by exploiting functions conjectured to be one-way and/or their hard-cores (see for example [2, 14, 13]). The work [2] presented PRGs based on the assumption that factoring is hard, while [14] presented PRGs based on the conjectured hardness of the discrete-logarithm problem. [13] presented PRGs based on the subset sum problem. Similarly, for nondeterministic-secure PRGs (namely, super-bits), Rudich conjectured the existence of a super-bit generator based on the hardness of the subset sum problem [13]. We are unaware of any additional conjectured construction of super-bits.

As mentioned above, another hardness measure of generators was introduced by Rudich:

▶ **Definition 3** (Demi-hardness [25]). *Let $g_n : \{0,1\}^n \to \{0,1\}^{l(n)}$ be a generator (computable in* P/poly*). Then the **demi-hardness** $H_{\mathrm{dh}}(g_n)$ of g_n is the minimal s for which there exists a nondeterministic circuit D of size at most s such that*

$$\mathop{\mathbb{P}}_{y\in\{0,1\}^{l(n)}}[D(y)=1] \geq \frac{1}{s} \quad \textit{and} \quad \mathop{\mathbb{P}}_{x\in\{0,1\}^{n}}[D(g_n(x))=1]=0. \tag{2}$$

We note that (2), which requires a distinguisher to make no mistake on generated strings, is a stronger requirement than (1). Thus, $H_{\mathrm{nh}}(g) \leq H_{\mathrm{dh}}(g)$ for every generator g.

Demi-bits are exponentially demi-hard generators, where "demi" here stands for "half":

▶ **Definition 4** (Demi-bits [25]). *A generator (computable in* P/poly*) $g : \{0,1\}^n \to \{0,1\}^{n+c}$ for some $c : \mathbb{N} \to \mathbb{N}$ is called c **demi-bit(s)** (or a c-demi-bit(s)) if $H_{\mathrm{dh}}(g) \geq 2^{n^{\varepsilon}}$ for some $\varepsilon > 0$ and all sufficiently large n's. In particular, if $c = 1$, we call g a demi-bit.*

It is worth mentioning that demi-bits g as in Definition 4 can be viewed as a *hitting set generator* against NP/poly (see below Section 2.1.2 and Santhanam [26]).

The difference between super-bits and demi-bits is that demi-bits require their distinguishers to break them in a stronger sense: a demi-bit distinguisher must always be correct on the pseudorandom strings (i.e., always output 0 for strings in the image of the generator). Thus, if g is a super-bit(s) (the plural here denotes that g may have a stretching-length greater than 1; if the stretching length is exactly 1, we say g is *a* super-bit), no algorithm in NP/poly can break g in the weaker sense (1), and hence no algorithm in NP/poly can break g in the stronger sense (2), which means g is also a demi-bit(s). In other words, the existence of super-bits implies the existence of demi-bits (although it is open if any of these two exists).

▶ Remark (Cryptographic vs. complexity-theoretic regime). In this work, we are interested only in the *cryptographic* regime of pseudorandomness. In this regime, the adversary whom the generator tries to fool is allowed to be stronger than the generator and specifically has sufficient computational resources to run the generator. In the *complexity-theoretic regime*, in which the adversary cannot simulate the generator, the notion of nondeterministic secure pseudorandomness was developed in works by, e.g., Klivans and van Melkebeek [15] as well as Shaltiel and Umans [29] (see also the recent work by Sdroievski and van Melkebeek [28] and references therein). These complexity-theoretic ideas have also found several applications in cryptography (originating from the work of Barak, Ong and Vadhan [4]). It is also worth mentioning that in the complexity-theoretic regime, one can use the original definition of the hardness of PRGs (Definition 12) even against nondeterministic adversaries; while this is not the case in the cryptographic regime, in which a PRG as in Definition 12 can never be safe against a nondeterministic adversary who guesses the seed.

1.2 Relations to barrier results

Recall that natural proofs [23] for proving circuit lower bounds are proofs that use a natural combinatorial property of boolean functions. A combinatorial property (or *a property*, for short) C of boolean functions is a set of boolean functions. We say a function f has property

C if f is in C. Let Γ and Λ be complexity classes, and F_n be the set of all $f : \{0,1\}^n \to \{0,1\}$. We say C is Γ-natural if a subset $C' \subseteq C$ satisfies *constructivity*, that is, it is Γ-decidable whether f is in C', and *largeness*, that is, C' constitutes a non-negligible portion of F_n. We say C is *useful* against Λ if every function family f that has property C infinitely often is not computable in Λ. The idea of natural proofs is that, if we want to prove some function family f (e.g., the boolean satisfiability problem SAT) is not in P/poly (or in general, some other complexity class Λ), we identify some natural combinatorial property C of f and show all function families that have property C are not in P/poly (i.e., the property is useful against P/poly). If f is NP-complete (e.g., SAT), then such a proof concludes $\mathsf{P} \neq \mathsf{NP}$.

Razborov and Rudich argued that, based on the existence of strong PRGs, no P/poly-natural proofs can be useful against P/poly. They showed that many known proofs of lower bounds against (non-monotone) boolean circuits are natural or can be presented as natural in some way.

In this context, the theory of nondeterministic-secure generators allows one to rule out a larger (arguably more natural) class of lower bound arguments:

▶ **Theorem 5** ([25]). *If super-bits exist, then there are no $N\tilde{P}/qpoly$-natural properties useful against* P/poly*, where $N\tilde{P}/qpoly$ is the class of languages recognised by non-uniform, quasi-polynomial-size circuit families.*

This theorem is proved based on the ability to *stretch* super-bits, namely, taking a generator that maps n bits to $n+1$ bits, which we refer to as stretching length 1, to a generator that maps n bits to $n+N$ bits, with $N > 1$, which we call stretching length N. In the standard theory of pseudorandomness, a hard-bit (i.e., a strong PRG with stretching length 1) is shown to be stretchable to polynomially many hard-bits (i.e., a polynomial stretching-length) [5, 31] and can be exploited to construct hard-to-break pseudorandom *function* generators (loosely speaking, generators that generate pseudorandom functions indistinguishable from truly random ones) [7]. As a hard-bit, a super-bit can also be stretched, using similar stretching algorithms, to polynomially many super-bits and to pseudorandom function generators secure against nondeterministic adversaries [25]. The proofs of the correctness of such stretching algorithms are based on a technique called the *hybrid argument* [10] reviewed below. In contrast, whether a demi-bit can be stretched even to two demi-bits was unknown before the current work, since this cannot be concluded with a direct application of a standard hybrid argument.

2 Contributions, significance and context

We develop the foundations of nondeterministic-secure pseudorandomness. This is the first systematic investigation into nondeterministic pseudorandomness (in the cryptographic regime) we are aware of, building on the primitives proposed by Rudich [25]. We provide new understanding of the primitives of the theory, namely, super and demi-bits, as well as introducing new notions and showing how they relate to established ones. We draw several conclusions from these results in average-case and proof complexity. We also achieve some modest progress on establishing sounder foundations for barrier results: by showing, for instance, that demi-bits can be (moderately) stretched, we provide some hope to strengthen the connection between demi-bits and unprovability results (as of now, it is only known that the existence of a super-bit yields barrier results, while we hope to show that the weaker assumption of the existence of demi-bits suffices for that matter).

2.1 Stretching demi-bits

In Demi-Bit Stretching Algorithm 24, we provide an algorithm that achieves a sublinear stretch for any given demi-bit. This solves the open problem of whether a demi-bit can be stretched to 2-bits [25, Open Problem 2] (see also Santhanam [26, Question 4]).

▶ **Theorem** (Informal; Theorem 25). *Every demi-bit $b : \{0,1\}^n \to \{0,1\}^{n+1}$ can be efficiently converted (stretched) into demi-bits $g : \{0,1\}^n \to \{0,1\}^{n+n^c}$, for every constant $0 < c < 1$.*

2.1.1 Discussion and significance of stretching demi-bits to barrier results

Stretching demi-bits can be viewed as a first step towards showing that the existence of a demi-bit rules out $N\tilde{P}/qpoly$-natural properties useful against $\mathsf{P/poly}$, as we explain below. Providing such a barrier for $N\tilde{P}/qpoly$-natural properties based on the existence of a demi-bit is important, since assuming the existence of a demi-bit is a weaker assumption than assuming the existence of a super-bit.

Why is stretching demi-bits a step towards showing that the existence of a demi-bit would rule out $N\tilde{P}/qpoly$-natural properties useful against $\mathsf{P/poly}$? The reason is that stretching is the first step in the argument to base barrier results on the existence of a super-bit, in the following sense: the existence of a super-bit implies barrier results because one can stretch super-bits to obtain pseudorandom function generators, from which one gets the barrier result as noted in Theorem 5 above (and the text that follows it). More precisely, stretching demi-bits is a first (and necessary) step towards Rudich's Open Problem 3, and this problem also implies Rudich's Open Problem 4:

▶ **Open problem** (Rudich's Open Problem 3 [25]). *Given a demi-bit, is it possible to build a pseudorandom function generator with exponential (2^{n^ε}) demi-hardness?*

▶ **Open problem** (Rudich's Open Problem 4 [25]). *Does the existence of demi-bits rule out $N\tilde{P}/qpoly$-natural properties useful against $\mathsf{P/poly}$?*

Moreover, the study of the stretchability of demi-bit(s) provides a perspective towards resolving Rudich's Open Problem 1 (a positive answer of which would also resolve positively Open Problem 4):

▶ **Open problem** (Rudich's Open Problem 1 [25]). *Does the existence of a demi-bit imply the existence of a super-bit?*

This is because the stretchability of super-bits is well understood, while previously we did not know anything about the stretchability of demi-bits. We expect that understanding better basic properties of demi-bits, such as stretchability, would shed light on the relation between the existence of demi-bits and the existence of super-bits (Open problem 1).

2.1.2 Applications in average-case complexity

Here we describe an application of Theorem 25 to the average-case hardness of time-bounded Kolmogorov complexity (cf. [11] for related recent progress).

As observed by Santhanam [26], a hitting set generator $g : \{0,1\}^n \to \{0,1\}^{n+1}$ exists iff there exists a demi-bit $b : \{0,1\}^n \to \{0,1\}^{n+1}$. To recall, a ***hitting set generator against a class of decision problems*** $\mathcal{C} \subseteq 2^{\{0,1\}^N}$ is a function $g : \{0,1\}^n \to \{0,1\}^N$, for $n < N$, such that the image of g *hits* (namely, intersects) every dense enough set A in $\mathcal{C}$ (that is, $|A| \geq \frac{2^N}{N^{O(1)}}$). And we have:

▶ **Proposition** (Proposition 26; [26]). *Let $n < N$. A hitting set generator $g : \{0,1\}^n \to \{0,1\}^N$ computable in the class $\mathcal{D}$ against* NP/poly *exists iff there exists a demi-bit $b : \{0,1\}^n \to \{0,1\}^N$ computable in $\mathcal{D}$ (against* NP/poly*).*

Santhanam [26, Proposition 3] established an equivalence between (succinct) hitting set generators and average-case hardness of MCSP, where MCSP stands for the *minimal circuit size problem*. However, as mentioned to us by Santhanam [27], similar arguments can show an equivalence between hitting set generators (not-necessarily succinct ones) and polytime bounded Kolmogorov complexity zero-error average-case hardness against NP/poly machines, as we show in this work.

We define the ***t-bounded Kolmogorov complexity of string x***, denoted $\mathrm{K}^t(x)$, to be the minimal length of a string D such that the universal Turing machine $U(D)$ (we fix some such universal machine) runs in time at most t and outputs x. See [3] for more details about time-bounded Kolmogorov complexity and Definition 9 there for the definition of time-bounded Kolmogorov complexity of strings (that definition actually produces the ith bit of the string x given an index i and D as inputs to U, but this does not change our result).

▶ **Definition 6** (The language $\mathrm{K}^t[s]$ and $\mathrm{K}^{\mathsf{poly}}[s(n)]$). *For a time function $t(n) : \mathbb{N} \to \mathbb{N}$ and a size function $s(n) : \mathbb{N} \to \mathbb{N}$, such that $s(n) \leq n$, let $\mathrm{K}^{t(n)}[s(n)]$ be the language $\{x \in \{0,1\}^* \; : \; |X| = n \;\wedge\; \mathrm{K}^{t(n)}(x) \leq s(n)\}$. We define $\mathrm{K}^{\mathsf{poly}}[s(n)]$ to be the language $\bigcup_{c\in\mathbb{N}} \mathrm{K}^{n^c}[s(n)]$.*

We also need to define the concept of zero-error average-case hardness against the class NP/poly (see Definition 27). Informally, for a language L to be zero-error average-case *easy* for NP/poly, there should be a nondeterministic polytime machine with advice such that given an input x the machine guesses a witness for $x \in L$ or a witness for $x \notin L$, and when the witness is found it answers accordingly; and moreover we assume that for a polynomial-small fraction of inputs there are such witnesses (for membership or non-membership in L). If a witness is not found the machine outputs "Don't-Know". (We also assume that there are no pairs of contradicting witnesses for both $x \in L$ and $x \notin L$.) A language is said to be zero-error average-case *hard* against the class NP/poly if it is not zero-error average-case easy against the class NP/poly.

In Section 4.1 we show the following:

▶ **Theorem** (Equivalence for average-case time-bounded Kolmogorov Complexity; Theorem 28). *$\mathrm{K}^{\mathsf{poly}}[n - O(1)]$ is zero-error average-case hard against* NP/poly *machines iff $\mathrm{K}^{\mathsf{poly}}[n - o(n)]$ is zero-error average-case hard against* NP/poly *machines.*

2.1.3 Applications in proof complexity

In proof complexity, Krajíček [16, 17] and independently Alekhnovich, Ben-Sasson, Razborov and Wigderson [1] introduced the notion of *proof complexity generators*. Given a P/poly mapping $g : \{0,1\}^n \to \{0,1\}^\ell$, with $n < \ell$, and a fixed vector $r \in \{0,1\}^\ell$, we denote by $\tau(g)_r$ the $\mathsf{poly}(\ell)$-size propositional formula that encodes naturally the statement $r \notin \mathrm{Im}(g)$, so that if r is not in the image of g then $\tau(g)_r$ is a propositional tautology. For r not in the image of g, the tautology $\tau(g)_r$ is called a *proof complexity generator*, and the hope is that for strong propositional proof systems one can establish (at least conditionally) that there are no $\mathsf{poly}(\ell)$-size proofs of $\tau(g)_r$, under the assumption that the mapping g is sufficiently pseudorandom (see also [24]). Krajíček observed the connection between proof complexity generators and demi-bits (see [16, Corollary 1.3] and the discussion that follows there).

An immediate corollary of Theorem 25 is the following.

▶ **Corollary** (Stretching proof complexity generators). *Let $b : \{0,1\}^n \to \{0,1\}^{n+1}$ be a demi-bit computable in* P/poly*. Let $0 < c < 1$ be a constant and $\ell = n + n^c$. Then, there is a proof complexity generator $g : \{0,1\}^n \to \{0,1\}^\ell$ in* P/poly*, such that for every propositional proof system, with probability at least $1 - \frac{1}{\ell^{\omega(1)}}$ over the choice of $r \in \{0,1\}^\ell$, there are no* poly(n)*-size proofs of the tautology $\tau(g)_r$.* [1]

2.1.4 Technique overview

We prove the stretchability of demi-bits by a novel and more flexible use of the hybrid argument combined with other ideas.

The *hybrid argument* (a.k.a. the hybrid method, the hybrid technique, etc.) is a common proof technique originating from the work of Goldwasser and Micali [10]. It was named by Leonid Levin. (See Section 3.1 for notations used below.) When we have a generator $g : \{0,1\}^n \to \{0,1\}^{m(n)}$, a distinguisher D, and a function p (usually a polynomial) such that

$$\mathbb{P}[D(U_m) = 1] - \mathbb{P}[D(g(U_n)) = 1] \geq 1/p(n), \tag{$*$}$$

where U_m stands for the truly random strings and $g(U_n)$ stands for the pseudorandom ones, the standard hybrid argument defines a spectrum (i.e. an ordered set) of random variables H_i's, called hybrids, traversing from one extreme, U_m, to another, $g(U_n)$. A concrete example is $H_i := g(U_n)[1...i] \cdot U_{m-i}, 0 \leq i \leq m$ (where $\cdot$ here means concatenation, and for a binary vector X we denote by $X[1 \dots, i]$ the i leftmost bits of X). In this example, indeed $H_0 = U_m$ and $H_m = g(U_n)$. Then the inequality $(*)$ can be written as:

$$\begin{aligned} 1/p(n) &\leq \mathbb{P}[D(U_m) = 1] - \mathbb{P}[D(g(U_n)) = 1] \\ &= \sum_i (\mathbb{P}[D(H_i) = 1] - \mathbb{P}[D(H_{i+1}) = 1]). \end{aligned}$$

Thus, a usual next step is to claim there exists some i such that

$$\mathbb{P}[D(H_i) = 1] - \mathbb{P}[D(H_{i+1}) = 1] \geq \frac{1}{k \cdot p(n)},$$

where k is the total number of hybrids (in the above example, $k = m$). In a nutshell, the hybrid argument now shows that if we can distinguish U_m from $g(U_n)$ by a $1/p(n)$ portion, then we can distinguish some neighbouring pair of hybrids H_i from H_{i+1} by a $1/(k \cdot p(n))$ portion. See [30] for a simple demonstration of the hybrid argument.

As mentioned above, a standard hybrid argument cannot be applied to prove that stretching a single demi-bit b by some stretching algorithm still constitutes demi-bit(s). We now intuitively explain the reason for this. A usual proof goes like this: we assume, for a contradiction, g are not demi-bits. Then there are some distinguisher D of g and a function p such that $\mathbb{P}[D(U_m) = 1] - \mathbb{P}[D(g(U_n)) = 1] \geq 1/p(n)$, and in particular $\mathbb{P}[D(g(U_n)) = 1] = 0$ as D breaks demi-bits, and we hope to construct a new appropriate distinguisher C of b based on D. However, as we saw above, a standard hybrid argument only yields that $\mathbb{P}[D(H_i) = 1] - \mathbb{P}[D(H_{i+1}) = 1] \geq 1/p'(n)$ for some function p' and cannot deduce that $\mathbb{P}[D(H_{i+1}) = 1] = 0$. Hence, it is unclear how to continue this construction.

[1] The points r are taken uniformly from $\{0,1\}^\ell$, and with probability $1 - 1/2^{\ell-n}$ the formula $\tau(g)_r$ is a tautology, because for all $r \in \{0,1\}^\ell \setminus \mathrm{Im}(g)$ the formula $\tau(g)_r$ is a tautology. While in some works, proof complexity generators are supposed to be hard for *every* r outside the image of the generator g, in our formulation the hardness is only with high probability over the r's.

Our argument for proving Theorem 25 proceeds by the contrapositive. We assume there is a distinguisher D which breaks demi-bits g (stretched from a single demi-bit b by Demi-Bit Stretching Algorithm 24) in the desired sense, and we want to construct a distinguisher C which breaks b. Rather than applying the hybrid argument directly to D, we apply the hybrid argument to a new distinguisher D' defined based on D: the new distinguisher D' can use nondeterminism to change the pseudorandom part of the hybrids and thus "amplifies" the probability of certificating randomness (intuitively, this can be viewed as changing the average-case analysis in the standard hybrid argument to a worst-case or existence analysis). By applying the hybrid argument to D', we are able to identify a non-empty class, denoted by S_2 in the proof, of random strings $y_{i+1} \dots y_m$, that are not random witnesses (in the sense that, for each $y_{i+1} \dots y_m$ in this class, there are no seeds $x_1, \dots, x_i$ such that $D(b(x_1) \dots b(x_i)\, y_{i+1} \dots y_m) = 1$). Thus, for $y_{i+1} \dots y_m$ in S_2, $y_i y_{i+1} \dots y_m$ can become a random witness (i.e., there are seeds $x_1, \dots, x_{i-1}$ such that $D(b(x_1) \dots b(x_{i-1})\, y_i \dots y_m) = 1$) only if y_i is truly random (i.e., not equal to $b(x)$ for some seed x). The hybrid argument also implies a "good" such $y_{i+1} \dots y_m$ in S_2, which can identify a sufficient portion of truly random y_i. We can thereby build a new distinguisher C to distinguish truly random strings from pseudorandom ones.

A key step that makes this proof work is that nondeterministically guessing seeds $x_1, \dots, x_{i-1}$ in $b(x_1) \dots b(x_{i-1}) z_i$ preserves the "randomness-structure" of $b(x_1) \dots b(x_{i-1}) z_i$, in the sense that: when $z_i = b(\cdot)$ is pseudorandom, the nondeterministic guess preserves the form $b(\cdot) \dots b(\cdot) b(\cdot)$ (i.e., i equal-length pseudorandom chunks); and when $z_i = y$ is truly random, it preserves the form $b(\cdot) \dots b(\cdot) y$ (i.e., $i - 1$ equal-length pseudorandom chunks followed by a truly random chunk y of the same length).

For common stretching algorithms that produce exponentially many new bits (e.g., recursively applying a one-bit generator), it is unclear how to use nondeterminism in a way that respects the "randomness-structure" of a given string. Nevertheless, the new proof technique should hopefully inspire researchers to further explore the stretchability of demi-bits. On the other hand, the fact could also be that there is a specific demi-bit which cannot be stretched to exponentially many demi-bits by the standard stretching algorithms which are applied to super-bits and strong PRGs.

2.2 Fine-grained characterisation of nondeterministic security based on unpredictability

Yao [31] defined PRGs as producing sequences that are computationally indistinguishable, by deterministic adversaries, from uniform sequences and proved that this definition of indistinguishability is equivalent to deterministic unpredictability, which was used in an earlier definition of PRGs suggested by Blum and Micali [5]. Loosely speaking, unpredictability means, given any strict prefix of a random string, it is infeasible to predict the next bit.

We provide a more fine-grained picture of nondeterministic hardness (Definition 1), by introducing the concept of nondeterministic unpredictability. This allows us to establish new lower and upper bounds to nondeterministic hardness, in the sense that we sandwich nondeterministic hardness between two unpredictability properties.

Specifically, we propose four notions of unpredictability for probability ensembles:

1. NP/poly-unpredictability: the capacity of being unpredictable by NP/poly predictors.
2. coNP/poly-unpredictability: the capacity of being unpredictable by coNP/poly predictors.
3. $\cup$-unpredictability: the capacity of being unpredictable by predictors in the union of NP/poly and coNP/poly.
4. $\cap$-unpredictability: the capacity of being unpredictable by nondeterministic function-computing predictors.

The names NP/poly-unpredictability, coNP/poly-unpredictability, and ∪-unpredictability (a shorthand for NP/poly ∪ coNP/poly-unpredictability) are self-explanatory, while the use of "∩-unpredictability" is somewhat less intuitive. We show in [30, Sec. 5.2] (where we use nondeterministic function-computing machines[2]), that a decision problem is in NP/poly ∩ coNP/poly if and only if it is decidable by a nondeterministic polynomial-size function-computing algorithm.

We establish the following characterisation of the nondeterministic hardness of generators from an unpredictability perspective:

▶ **Theorem.** *Here, $A \leq B$ means that if a generator has property B, then it also has property A:*

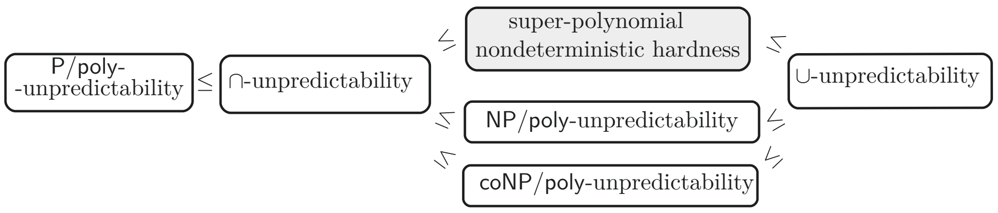
P/poly- -unpredictability
≤
∩-unpredictability
super-polynomial nondeterministic hardness
NP/poly-unpredictability
coNP/poly-unpredictability
∪-unpredictability

Figure 1 Super-polynomial nondeterministic hardness here refers to Definition 1. Note that ∪-unpredictability is at least as strong as NP/poly-unpredictability, because it rules out predictors in *both* NP/poly and coNP/poly. And similarly, ∪-unpredictability is at least as strong as coNP/poly-unpredictability.

2.3 Super-cores: hard-core predicates in the nondeterministic setting

In the deterministic context, the existence of strong PRGs is known to be equivalent to the existence of central cryptographic primitives such as one-way functions, secure private-key encryption schemes, digital signatures, etc. Liu and Pass [18] recently showed that a meta-complexity assumption about mild average-case hardness of the time-bounded Kolmogorov Complexity is also equivalent to the existence of strong PRGs. On the other hand, in the nondeterministic case we have no known such equivalent characterisations (of nondeterministic-secure PRGs, namely, super-bits). We introduce a definition of *super-cores* serving as a nondeterministic variant of hard-cores. We then use this concept to draw the first equivalent characterisation of super-bits.

We start by reviewing the concepts of one-way functions and hard-core predicates. Loosely speaking, a one-way function (family) f is a one that is easy to compute but hard to invert on average (with the probability taken over the domain of f). More precisely, "easy to compute" means f is in P or P/poly, and "hard to invert" means any efficient deterministic algorithm can only invert a negligible portion of $y = f(x)$ when x is unseen. By "efficient", in the uniform setting, we mean an algorithm in bounded-error probabilistic polynomial time (BPP), and in the nonuniform setting, an algorithm in P/poly, and by "invert", we mean finding an x' for a given y in $range(f)$ such that $f(x') = y$. A negligible portion for us means a portion that is less than $1/p(n)$ for any polynomial p and all large n's. We say $b : \{0,1\}^n \to \{0,1\}$ in P or P/poly is *a hard-core of a function f* if it is impossible to efficiently predict $b(x)$ with probability at least $1/2 + 1/\mathsf{poly}(n)$ given $f(x)$.

[2] We say a nondeterministic algorithm $\mathcal{A}$ is a **function-computing** algorithm, if for every input $x \in \{0,1\}^n$, every computation branch yields one of $\{0,1,\bot\}$, in which $\bot$ indicates a failure, and there is always a computation branch yielding 0 or 1.

The existence of hard-core predicates is known (e.g., $b(x) = x[-1]$, the last bit of a string x, is a hard-core of the function $f(x) = x[1]$, the first bit of x), but the existence of a hard-core for a one-way function and the existence of any one-way function to begin with are unknown. Goldreich and Levin [8] proved that inner product *mod* 2 is a hard-core for any function of the form $g(x, r) = (f(x), r)$, where f is any one-way function and $|x| = |r|$. Subsequently, Håstad, Impagliazzo, Levin, and Luby [12] showed that strong PRGs exist if and only if one-way functions exist. This theorem can be stated equivalently as: a strong PRG exists if and only if a hard-core of some one-way function exists.

Since a strong PRG exists if and only if a hard-core of some one-way function exists, and a super-bit is the nondeterministic analogue of strong PRGs, a meaningful question to ask is:

What are the nondeterministic analogues of one-way functions and hard-cores?

To come up with a reasonable definition of super one-way functions is not an easy task because, for any function f, a nondeterministic algorithm can always invert a range-element y by guessing some x and checking if $f(x) = y$. Similarly, a reasonable definition of super-core predicates is non-trivial as well: for any function f and predicate b, a nondeterministic algorithm can predict $b(x)$ when given $f(x)$ as input by guessing x and then applying b.

We propose a definition of super-cores of a function f, which are secure against both NP/poly and coNP/poly predictors in the sense of [30, Def. 6.7], when $f(x)$ is presented as the input. With this definition, we can establish the following equivalence (we say a function $f : \{0, 1\}^n \to \{0, 1\}^{m(n)}$ is *non-shrinking* if $m(n) \geq n$ for every n):

▶ **Theorem** (Informal). *There is a super-core of some non-shrinking function if and only if there is a super-bit.*

This result is analogous to the known equivalence between the existence of a hard-core of some one-way function and the existence of a strong PRG. We also show that a certain class of functions, which may have hard-cores, cannot possess any super-core. This also suggests that a one-way function could possibly possess no super-cores. (See the text before [30, Thm. 6.16] for the definition of "predominantly one-to-one".)

▶ **Theorem** (Informal). *If a non-shrinking function f is "predominantly" one-to-one, then f does not have a super-core.*

We thus provide a step forward to better understand the nondeterministic hardness of PRGs and develop a sensible definition of one-way functions in the nondeterministic setting.

➛***Below we shall provide the full proofs involving demi-bit stretching and its applications in average-case complexity. For all other results we refer the reader to the fuller version [30].***

3 Preliminaries and basic concepts

3.1 Notations and conventions

We follow the following conventions:

- $\mathbb{N}$ denotes the set of positive integers (excluding 0). For $n \in \mathbb{N}$, $[n]$ denotes $\{1, ..., n\}$. $[0]$ is the empty set $\emptyset$.
- The size of a Boolean circuit C, denoted as $size(C)$ or $|C|$, is the total number of gates (including the input gates). $Circuit[s]$ denotes the Boolean circuits of size at most s. If $s : \mathbb{N} \to \mathbb{N}$ is a function, $Circuit[s]$ contains all the Boolean circuit families C_n such that $|C_n| \leq s(n)$ for all large n's.

- All the distributions we consider in this work are, by default, uniform. U_n denotes the uniform distribution over $\{0,1\}^n$ unless stated otherwise.
- For functions $f, g : \{0,1\}^n \to \{0,1\}$, we say g γ-approximates f if $\mathbb{P}_x[f(x) = g(x)] \geq \gamma$.
- For a string x, where its bits are indexed from left to right by 1 to $|x|$, $x[i]$ denotes its i-th bit, and $x[i...j]$ denotes the sub-string indexed from i to j of x (if $i > j$, $x[i...j] = \varepsilon$, the empty string). $x[-i]$ denotes its i-th last bit.
- For strings x, y, we may use any of the following to denote the concatenation of x and y: xy, (x, y), $x \cdot y$.
- We may not verbally distinguish a function with its string representation (this can be a truth table, or a string encoding a circuit representation of this function, etc.) when there is no ambiguity.

We may also follow other common conventions used in the complexity community or literature.

3.2 Computational models

▶ **Definition 7** (Randomized circuits; equiv. probabilistic circuit). *A circuit C is a **randomized circuit** (equivalently, a* probabilistic circuit*) if, in addition to the standard input bits (similar to the input bits of a non-randomized circuit), it contains zero or more* random input bits *(i.e., bits taken from a random distribution). We call $\{C_n\}_{n=1}^{\infty}$ a **randomized circuit family** if for every n, C_n is a randomized circuit with n standard input bits.*

Note that if a randomized circuit family $\{C_n\}$ is in $Circuit[s(n)]$, it means that for every sufficiently large n, the randomized circuit C_n has size at most $s(n)$, which automatically constrains the number of random input bits that C_n is allowed to have.

▶ **Definition 8** (Nondeterministic and co-nondeterministic circuits). *A circuit $C(\overline{x}, \overline{r})$ is a **(co-)nondeterministic circuit** if, in addition to the standard input bits $\overline{x}$, it contains zero or more* nondeterministic input bits *$\overline{r}$ (namely, bits that are meant to control the nondeterministic decisions made by the circuit). A nondeterministic circuit with a single output bit is said to **accept** an input $\overline{\alpha} \in \{0,1\}^n$ to $\overline{x}$ iff there* exists *an assignment $\overline{\beta} \in \{0,1\}^{|\overline{r}|}$ to $\overline{r}$ such that $C(\overline{\alpha}, \overline{\beta}) = 1$ (and otherwise it is said to reject $\overline{x}$). A co-nondeterministic circuit with a single output bit is said to **reject** an input $\overline{\alpha} \in \{0,1\}^n$ to $\overline{x}$ iff there exists an assignment $\overline{\beta} \in \{0,1\}^{|\overline{r}|}$ to $\overline{r}$ such that $C(\overline{\alpha}, \overline{\beta}) = 0$ (and otherwise it is said to accept $\overline{x}$). We call $\{C_n\}$ a (co-)nondeterministic circuit family if for every n, C_n is a (co-)nondeterministic circuit with n standard input bits.*

▶ **Definition 9** (Oracle circuits). *C is an **oracle circuit** if it is allowed to use oracle gates. We write C as $C^{f_1,...,f_k}$ if C has oracle gates computing Boolean functions $f_1, ..., f_k$.*

We note that an oracle gate computing a Boolean function $f : \{0,1\}^n \to \{0,1\}$ has fan-in n, and in our model, an oracle gate is allowed to appear in any place in the circuit.

3.3 Natural proofs

Let F_n be the set of all functions $f : \{0,1\}^n \to \{0,1\}$ and Γ and Λ be complexity classes. We call $C = (C_n)_{n \in \mathbb{N}}$ a **combinatorial property** of boolean functions if each $C_n \subseteq F_n$.

▶ **Definition 10** (Natural properties [23]). *We say a combinatorial property $C = (C_n)_{n \in \mathbb{N}}$ is **Γ-natural** if some $C' = (C'_n)_{n \in \mathbb{N}}$ with $C'_n \subseteq C_n$ for each n satisfies:*

- ***Constructivity.** Whether $f \in C'_n$ is computable in Γ when f is encoded by its truth table as input.*

- ***Largeness.*** *$|C'_n| \geq 2^{-O(n)} \cdot |F_n|$ for all large n's.*

*We say C is **useful** against $\mathbf{\Lambda}$ if it satisfies:*

- ***Usefulness.*** *For any function family $f = (f_n)_{n\in\mathbb{N}}$, if $f_n \in C_n$ infinitely often, then $f \notin \Lambda$.*

A circuit lower bound proof (that some function family is not in Λ) is called a **Γ-natural proof** against Λ if it uses, explicitly or implicitly, some Γ-natural combinatorial property useful against Λ. Especially, a P/poly-natural proof against P/poly is a proof that uses a P/poly-natural combinatorial properties useful against P/poly.

We note that the notion of natural proofs, unlike natural combinatorial property, is not defined in a mathematically rigorous sense. Nevertheless, the use of the terminology "natural proof" in a statement more intuitively embodies our intention and also does not affect the rigorousness of the statement: whenever we say Γ-natural proofs against Λ do or do not exist, what we mean, in a mathematically rigorous sense, is Γ-natural combinatorial properties against Λ do or do not exist.

3.4 Pseudorandom generators

We recall here the basic definition of pseudorandom generators. As mentioned in the introduction, all the distributions we consider in this work are, by default, uniform, and U_n denotes the uniform distribution over $\{0,1\}^n$ unless stated otherwise.

▶ **Definition 11** (Generators). *A function family $g_n : \{0,1\}^n \to \{0,1\}^{l(n)}$ is a **generator** if $g_n \in$ P/poly and $l(n) > n$ for every n. We call such an l a* stretching function *and call $l(n) - n$ the* stretching length *of g_n (sometimes $l(n)$ is called* the stretching length*).*

We note that all the generators in this work, with the exception of Section 4.1, will be computable in P/poly, although in more general settings, it is not required that generators are P/poly-computable (cf. [20]).

▶ **Definition 12** (Standard hardness). *Let $g_n : \{0,1\}^n \to \{0,1\}^{l(n)}$ be a generator. Then the **hardness** $H(g_n)$ of g_n is the minimal s for which there exists a (deterministic) circuit D of size at most s such that*

$$\left| \underset{y\in\{0,1\}^{l(n)}}{\mathbb{P}}[D(y)=1] - \underset{x\in\{0,1\}^{n}}{\mathbb{P}}[D(g_n(x))=1] \right| \geq 1/s(n).$$

The order of the two terms in the absolute value and the absolute value itself are immaterial since in the deterministic setting, we can always flip the output bit of a distinguisher D.

▶ **Definition 13** ((Strong) pseudorandom generators (PRG)). *A generator $g : \{0,1\}^n \to \{0,1\}^{l(n)}$ is called a (strong) **PRG** if for every D in P/poly, every polynomial p, and all sufficiently large n's,*

$$\left| \underset{y\in\{0,1\}^{l(n)}}{\mathbb{P}}[D(y)=1] - \underset{x\in\{0,1\}^{n}}{\mathbb{P}}[D(g(x))=1] \right| < 1/p(n).$$

In other words, a strong PRG is defined to be a generator safe against all polynomial-size distinguishers. An alternative definition used in some texts is: a generator with hardness at least 2^{n^ε} for some $\varepsilon > 0$ and all large n's, which defines a stronger PRG.

We shall say that a function $f(n) : \mathbb{N} \to \mathbb{R}^+$ is *(at least) exponential* if there exists some $\varepsilon > 0$ such that $f(n) \geq 2^{n^\varepsilon}$ for all sufficiently large n's. A function is *not* (at least) exponential if for every $\varepsilon > 0$, there exist infinitely many n's such that $f(n) < 2^{n^\varepsilon}$; this latter condition is equivalent to: there is an infinite monotone sequence $(n_i) \subseteq \mathbb{N}$ such that $f(n_i)$ is sub-exponential in n_i (i.e., $f(n_i) = 2^{n_i^{o(1)}}$).

The existence of a strong PRG is considered quite plausible because many intractable problems (e.g., factoring) seem to provide a basis for constructing such generators (cf. [9]).

▶ **Conjecture 14.** *Strong PRGs exist.*

Razborov and Rudich showed that the existence of a strong PRG rules out the existence of P/poly-natural proofs useful against P/poly [23].

Concrete examples of strong PRGs are unknown, as the existence of such a PRG implies P $\neq$ NP in the uniform setting and P/poly $\neq$ NP/poly in the nonuniform setting. Nevertheless, generators that can fool classes of weaker distinguishers were constructed (e.g., Nisan and Wigderson [20]).[3]

3.5 Super-bits and demi-bits

Here we provide a brief review of the main results and open problems in [25] that are relevant to our work. (Some of the text is repeated from the introduction.)

▶ **Definition 15** (Nondeterministic hardness). *Let $g_n : \{0,1\}^n \to \{0,1\}^{l(n)}$ be a generator. Then the* ***nondeterministic hardness*** *$H_{\mathrm{nh}}(g_n)$ (also called* ***super-hardness****) of g_n is the minimal s for which there exists a nondeterministic circuit D of size at most s such that*

$$\mathop{\mathbb{P}}_{y\in\{0,1\}^{l(n)}}[D(y)=1] - \mathop{\mathbb{P}}_{x\in\{0,1\}^n}[D(g_n(x))=1] \geq \frac{1}{s}. \tag{1}$$

In contrast to the definition of deterministic hardness, the order of the two possibilities on the left-hand side is crucial. This order forces a nondeterministic distinguisher to certify the randomness of a given input. Reversing the order or keeping the absolute value trivialize the task of breaking g: a distinguisher D can simply guess a seed x and check if $g(x)$ equals the given input. For such a D, we have $\mathbb{P}[D(g(x))=1]=1$ and $\mathbb{P}[D(y)=1]\leq 1/2$.

We call exponentially super-hard generators super-bits:

▶ **Definition 16** (Super-bits). *A generator (in* P/poly*) $g_n : \{0,1\}^n \to \{0,1\}^{n+c}$ for some $c:\mathbb{N}\to\mathbb{N}$ is called c* ***super-bit(s)*** *(or a c-super-bit(s)) if $H_{\mathrm{nh}}(g_n) \geq 2^{n^{\varepsilon}}$ for some $\varepsilon > 0$ and all sufficiently large n's. In particular, if $c=1$, we call g_n a super-bit.*

The term *super-bits* thus stands for pseudorandom bits that can fool "super" powerful adversaries. Rudich constructed a candidate super-bit based on the subset sum problem and conjectured that:

▶ **Conjecture 17** (Super-bit conjecture). *There exists a super-bit.*

The main theorem in [25] is the following one, which is proved based on the stretchability of super-bits as discussed above.

▶ **Theorem 18** ([25]). *If super-bits exist, then there are no $N\tilde{P}/qpoly$-natural properties useful against* P/poly*, where $N\tilde{P}/qpoly$ is the class of languages recognised by non-uniform, quasi-polynomial-size circuit families (where quasi-polynomial means $n^{log^{O(1)}(n)}$).*

[3] For weak models, both complexity-theoretic generators and cryptographic generators are known. Complexity-theoretic generators fooling AC^0 were shown by Nisan (which is the Nisan-Wigderson generator with PARITY as the hard function, and is earlier than [20]). Cryptographic generators are constructed for example in [6]. For P/poly, both complexity-theoretic generators and cryptographic generators are unknown. However, the assumptions needed for complexity-theoretic generators (e.g, E requires exponential-size) are much weaker than those needed for cryptographic generators (e.g., that one way functions exists).

We remark that, in this theorem, the "largeness" requirement of $N\tilde{P}/qpoly$-natural properties can in fact be relaxed to $|C'_n| \geq 2^{-n^{O(1)}} \cdot |F_n|$ (cf. Definition 10).

Rudich also proposed another notion, called demi-hardness, which he considered to be more intuitive than super-hardness:

▶ **Definition 19** (Demi-hardness). *Let $g_n : \{0,1\}^n \to \{0,1\}^{l(n)}$ be a generator (in* P/poly*). Then the* ***demi-hardness*** *$H_{\mathrm{dh}}(g_n)$ of g_n is the minimal s for which there exists a nondeterministic circuit D of size at most s such that*

$$\underset{y\in\{0,1\}^{l(n)}}{\mathbb{P}}[D(y)=1] \geq \frac{1}{s} \quad \text{and} \quad \underset{x\in\{0,1\}^{n}}{\mathbb{P}}[D(g_n(x))=1]=0. \tag{2}$$

We note that (2), which requires a distinguisher to make no mistakes on any generated strings, is a stronger requirement than (1). Thus, $H_{\mathrm{nh}}(g) \leq H_{\mathrm{dh}}(g)$ for every generator g.

We call exponentially demi-hard generators demi-bits, where "demi" is meant to stand for "half" here:

▶ **Definition 20** (Demi-bits). *A generator (in* P/poly*) $g_n : \{0,1\}^n \to \{0,1\}^{n+c}$ for some $c : \mathbb{N} \to \mathbb{N}$ is called c* ***demi-bit(s)*** *(or a c-demi-bit(s)) if $H_{\mathrm{dh}}(g_n) \geq 2^{n^{\varepsilon}}$ for some $\varepsilon > 0$ and all sufficiently large n's. In particular, if $c = 1$, we call g_n a demi-bit.*

As $H_{\mathrm{nh}}(g) \leq H_{\mathrm{dh}}(g)$, it is natural to conjecture:

▶ **Conjecture 21** (Demi-bit conjecture [25]). *There exists a demi-bit.*

Accordingly, it is natural to ask:

▶ **Open problem** ([25]). *Does the existence of a demi-bit imply the existence of a super-bit?*

As discussed above, a super-bit can be stretched to polynomially many super-bits and to pseudorandom function generators secure against nondeterministic adversaries. In contrast, whether a demi-bit can be stretched to even two demi-bits was unknown prior to our work:

▶ **Open problem** ([25]; Resolved in Section 4). *Given a demi-bit, is it possible to stretch it to 2-demi-bits?*

The question that remains open is:

▶ **Open problem** ([25]). *Given a demi-bit, is it possible to build a pseudorandom function generator with exponential demi-hardness?*

A positive answer to the last problem would answer the following:

▶ **Open problem** ([25]). *Does the existence of a demi-bit rule out the existence of $N\tilde{P}/qpoly$-natural properties against* P/poly*?*

3.6 Infinitely often super-bits and demi-bits

Here, we formalize a weaker variant of super-bits and demi-bits, that only requires infinitely many n's to be "hard" (recall super-bits/demi-bits require hardness for all sufficiently large n's). This variant occasionally appears implicitly in the literature but may not have been formally defined.

▶ **Definition 22** (Infinitely often super-bits/demi-bits). *A generator (in* P/poly*) $g_n : \{0,1\}^n \to \{0,1\}^{n+c}$, for some $c : \mathbb{N} \to \mathbb{N}$ is called c* ***infinitely often (i.o.) super-bit(s)/demi-bit(s)****, if $H_{\mathrm{nh}}(g_n) \geq 2^{n^{\varepsilon}}/H_{\mathrm{dh}}(g_n) \geq 2^{n^{\varepsilon}}$ for some $\varepsilon > 0$ and infinitely many n's. In particular, if $c = 1$, we call g_n an i.o. super-bit/demi-bit.*

In fact, it is an easy observation that we can construct from a reasonably frequent i.o. super-bit/demi-bit, a super-bit/demi-bit, by properly choosing a prefix of a given input n and applying the i.o. algorithm to the prefix. More details follow. However, we are unaware if we can construct a super-bit/demi-bit from any i.o. super-bit/demi-bit.

▶ **Lemma 23.** *Assume $g_n : \{0,1\}^n \to \{0,1\}^{n+c}$ for some $c \in \mathbb{N}$ are c i.o. super-bits/demi-bits. If there exist a polynomial p and an infinite monotone sequence $(n_i)_{i\in\mathbb{N}} \subseteq \mathbb{N}$ such that: (1) $n_{i+1} \le p(n_i)$, and (2) for some $\varepsilon > 0$ and every $n \in (n_i)_{i\in\mathbb{N}}$, $H_{\text{nh}}(g_n) \ge 2^{n^\varepsilon}/H_{\text{dh}}(g_n) \ge 2^{n^\varepsilon}$, then there exists a c-super-bits/c-demi-bits constructed from g_n.*

Proof. We present the proof for constructing super-bits from i.o. super-bits, and the proof for constructing demi-bits from i.o. demi-bits is almost identical.

Assume g, (n_i), p are as given in the lemma statement. Denote $m = n+c$ and $m_i = n_i+c$. We construct a new generator G as follows:

Given $x_n \in \{0,1\}^n$, there is an i such that $n_i \le n < n_{i+1}$. Define $G(x_n) = g(a) \cdot b$, where $a = x_n[1...n_i], b = x_n[n_i + 1...n]$. We note $|G(x_n)| = |g(a)| + |b| = n + c$.

We want to show that G is indeed super-bits. Suppose, for a contradiction, G is not. Then there exist an infinite monotone sequence $S \subseteq \mathbb{N}$, a sub-exponential function s, and a distinguisher D of size s such that for every $n \in S$,

$$\begin{aligned}1/s(n) &\le \mathbb{P}[D(U_m) = 1] - \mathbb{P}[D(G(U_n)) = 1]\\ &= \mathbb{P}[D(U_{m_i}U_{m-m_i}) = 1] - \mathbb{P}[D(g(U_{n_i})U_{m-m_i}) = 1]\end{aligned}$$

Thus, for every $n \in S$, there exists a fixed string $w = w(n)$ such that

$$1/s(n) \le \mathbb{P}[D(U_{m_i}w) = 1] - \mathbb{P}[D(g(U_{n_i})w) = 1].$$

We now construct a new distinguisher D' for g as follows:

Given $Y \in \{0,1\}^{m_i}$ as input, if there exists an $n \in S$ such that $n_i \le n < n_{i+1}(\le p(n_i))$, D' outputs $D(Yw(n))$, and otherwise D' always outputs 0 (which means D' fails to do anything for such an n_i).

Note that the (1) $n_{i+1} \le p(n_i)$ assumption guarantees the efficiency of D'. As S is infinite and every $n \in S$ is between some n_i and n_{i+1}, there are infinitely many n_i's such that:

$$\mathbb{P}\left[D'(U_{m_i}) = 1\right] - \mathbb{P}\left[D'(g(U_{n_i})) = 1\right] = \mathbb{P}[D(U_{m_i}w) = 1] - \mathbb{P}[D(g(U_{n_i})w) = 1] \ge 1/s(n)$$

This shows, for infinitely many n_i's, $H_{\text{nh}}(g(n_i)) \le 1/s'(n)$ for some sub-exponential s', which contradicts the assumption (2) in the lemma statement. ◀

▶ Remark. Lemma 23 is often used implicitly in the constructions of super-bits and demi-bits.

4 Stretching demi-bits

In this section, we answer affirmatively whether a demi-bit is stretchable, which had been an open problem from the original work of Rudich [25].

We propose Demi-Bit Stretching Algorithm 24, which stretches a single demi-bit to n^c demi-bits for any $c < 1$, and verify the correctness of this stretching algorithm (i.e., verify the exponential demi-hardness of the new elongated generator). Intuitively, Demi-Bit Stretching Algorithm 24 partitions a given seed into disjoint pieces and apply the 1-demi-bit generator to each piece. The algorithm proposed here is very similar to other stretching or amplification algorithms known in the literature (e.g., cf. [31, 19]). The real difficulty in stretching demi-bits comes from to need to prove the correctness of the attempted stretch (i.e., to proof the stretching algorithm applied to preserve exponential demi-hardness).

▶ **Demi-Bit Stretching Algorithm 24.** *Suppose $b_n : \{0,1\}^n \to \{0,1\}^{n+1}$ is a demi-bit and $0 < c < 1$ is a constant. We define a new generator $g : \{0,1\}^N \to \{0,1\}^{N+m}$ with input length N and stretching length $m = \lceil N^c \rceil$ as follows: given input x (of length N), let $n = \lfloor \frac{N}{m} \rfloor$ and $x = x_1 x_2 \dots x_m r$, where each x_i has length n, and define $g(x) = b(x_1) \dots b(x_m) r$.*

The correctness proof proceeds by contrapositive. That is, we assume there is a distinguisher D which breaks demi-bits g (stretched from a single demi-bit b by Demi-Bit Stretching Algorithm 24) in the desired sense, and we want to construct a distinguisher C which breaks b. Rather than applying the hybrid argument directly to D, we apply the hybrid argument to a new distinguisher D' defined based on D: the new distinguisher D' can use nondeterminism to change the pseudorandom part of the hybrids and thus "amplify" the probability of certificating randomness (intuitively, this can be viewed as changing the average-case analysis in the standard hybrid argument to a worst-case or "existence" analysis). By applying the hybrid argument to D', we are able to identify a non-empty class S_2 of random strings $y_{i+1} \dots y_m$, which are not random witnesses (in the sense that, for each $y_{i+1} \dots y_m$ in this class, there are no seeds $x_1, \dots, x_i$ such that $D(b(x_1) \dots b(x_i)\, y_{i+1} \dots y_m) = 1$). Thus, for $y_{i+1} \dots y_m$ in S_2, $y_i y_{i+1} \dots y_m$ can become a random witness (i.e., there are seeds $x_1, \dots, x_{i-1}$ such that $D(b(x_1) \dots b(x_{i-1})\, y_i \dots y_m) = 1$) only if y_i is truly random (i.e., not equal to $b(x)$ for some seed x). The hybrid argument also implies a "good" such $y_{i+1} \dots y_m$ in S_2, that can identify a sufficient portion of truly random y_i. We can thereby build a new distinguisher C to distinguish truly random strings from pseudorandom ones.

In the proof, we reserve the **bold face** for random variables.

▶ **Theorem 25** (Main theorem for stretching demi-bits)**.** *The generator g (with a sub-linear stretching-length), as defined in Demi-Bit Stretching Algorithm 24, has at least exponential demi-hardness.*

Proof. Demi-bit b and constant c are as given in Demi-Bit Stretching Algorithm 24. Suppose, towards contradiction that g does not have exponential demi-hardness. That is, there is a sub-exponential (in the input length, denoted by N) size nondeterministic circuit D such that, for infinitely many N's (recall $m = \lceil N^c \rceil$ and $n = \lfloor \frac{N}{m} \rfloor$, and without loss of generality, we may assume $m|N$), $\mathbb{P}[D(\mathbf{y_1} \dots \mathbf{y_m}) = 1] \geq 1/|D|$ and $\mathbb{P}[D(b(\mathbf{x_1}) \dots b(\mathbf{x_m})) = 1] = 0$, where $\mathbf{x_1}, \dots, \mathbf{x_m}$ are totally independent length-n random strings and $\mathbf{y_1}, \dots, \mathbf{y_m}$ are totally independent length-$(n+1)$ random strings. Our aim is to efficiently break b in the desired sense.

We define a new nondeterministic circuit D' that takes a pair of inputs: the first is the same input as D's, that is, an $(N+m)$-bit string $y_1 \dots y_m$, and the second is $i \in \{0, 1, \dots, m\}$ (with i properly encoded):

Given input $(y_1 \dots y_m, i)$, D' guesses n-bit strings $x_1, \dots, x_i$ and does whatever D does on $b(x_1) \dots b(x_i)\, y_{i+1} \dots y_m$.

We observe that:

- $\mathbb{P}[D'(\mathbf{y_1} \dots \mathbf{y_m}, 0) = 1] = \mathbb{P}[D(\mathbf{y_1} \dots \mathbf{y_m}) = 1] \geq 1/|D|$ (by the definition of D');
- $\mathbb{P}[D'(b(\mathbf{x_1}) \dots b(\mathbf{x_m}), m) = 1] = \mathbb{P}[D(b(\mathbf{x_1}) \dots b(\mathbf{x_m})) = 1] = 0$ (by assumption on D; otherwise $\mathbb{P}[D'(b(\mathbf{x_1}) \dots b(\mathbf{x_m}), m) = 1] > 0$ would mean there exist $x_1, \dots, x_m$ such that $D(b(x_1) \dots b(x_m)) = 1$);

- D' is also of sub-exponential size.

We can now apply the hybrid argument to D':

$$\begin{aligned}1/|D| &\le \mathbb{P}\left[D'(\mathbf{y_1}\ldots\mathbf{y_m},0)=1\right]-\mathbb{P}\left[D'(b(\mathbf{x_1})\ldots b(\mathbf{x_m}),m)=1\right]\\ &=\sum_i(\mathbb{P}\left[D'(b(\mathbf{x_1})\ldots b(\mathbf{x_{i-1}})\,\mathbf{y_i}\ldots\mathbf{y_m},i-1)\right]-\mathbb{P}\left[D'(b(\mathbf{x_1})\ldots b(\mathbf{x_i})\,\mathbf{y_{i+1}}\ldots\mathbf{y_m},i)\right])\end{aligned}$$

yields there is an $i=i(N)$ (for infinitely many N's) such that

$$P_{i-1}-P_i\ge 1/(m\cdot|D|),$$

where $P_{i-1}:=\mathbb{P}\left[D'(b(\mathbf{x_1})\ldots b(\mathbf{x_{i-1}})\mathbf{y_i}\mathbf{y_{i+1}}\ldots\mathbf{y_m},i-1)=1\right]$ and
$P_i:=\mathbb{P}\left[D'(b(\mathbf{x_1})\ldots b(\mathbf{x_{i-1}})b(\mathbf{x_i})\mathbf{y_{i+1}}\ldots\mathbf{y_m},i)=1\right]$. As m is sublinear in n, $m\cdot|D|$ is still sub-exponential in N.

We denote by $S=\{0,1\}^{(n+1)\times(m-i)}$ the set of $(m-i)$-tuples of $(n+1)$-bit strings, and let $S_1=\{(y_{i+1},\ldots,y_m)\in S:\exists(x_1,\ldots,x_i)\in\{0,1\}^{n\times i}\ D(b(x_1)\ldots b(x_i)y_{i+1}\ldots y_m)=1\}$, and $S_2=S\setminus S_1$. We note that:

- $D'(b(x_1)\ldots b(x_{i-1})\ y_iy_{i+1}\ldots y_m,i-1)=1$ if and only if $D'(O\ y_iy_{i+1}\ldots y_m,i-1)=1$, where $O=0^{(n+1)\cdot(i-1)}$ (because, by construction, $D'(\ldots,i-1)$ ignores its first $i-1$ input strings);
- $D'(b(x_1)\ldots b(x_{i-1})b(x_i)\ y_{i+1}\ldots y_m,i)=1$ if and only if $(y_{i+1},\ldots,y_m)\in S_1$, and thus $\mathbb{P}\left[\mathbf{y_{i+1}}\ldots\mathbf{y_m}\in S_1\right]=P_i$.

Therefore,

$$\begin{aligned}P_{i-1}&=\mathbb{P}\left[D'(b(\mathbf{x_1})\ldots b(\mathbf{x_{i-1}})\mathbf{y_i}\mathbf{y_{i+1}}\ldots\mathbf{y_m},i-1)=1\right]\\ &=\mathbb{P}\left[D'(b(\mathbf{x_1})\ldots b(\mathbf{x_{i-1}})\mathbf{y_i}\mathbf{y_{i+1}}\ldots\mathbf{y_m},i-1)=1|\mathbf{y_{i+1}}\ldots\mathbf{y_m}\in S_1\right]\cdot\mathbb{P}\left[\mathbf{y_{i+1}}\ldots\mathbf{y_m}\in S_1\right]\ +\\ &\quad\ \mathbb{P}\left[D'(b(\mathbf{x_1})\ldots b(\mathbf{x_{i-1}})\mathbf{y_i}\mathbf{y_{i+1}}\ldots\mathbf{y_m},i-1)=1|\mathbf{y_{i+1}}\ldots\mathbf{y_m}\in S_2\right]\cdot\mathbb{P}\left[\mathbf{y_{i+1}}\ldots\mathbf{y_m}\in S_2\right]\\ &\le 1\cdot\mathbb{P}\left[\mathbf{y_{i+1}}\ldots\mathbf{y_m}\in S_1\right]\ +\\ &\qquad\quad \mathbb{P}\left[D'(O\ \mathbf{y_i}\mathbf{y_{i+1}}\ldots\mathbf{y_m},i-1)=1|\mathbf{y_{i+1}}\ldots\mathbf{y_m}\in S_2\right]\cdot\mathbb{P}\left[\mathbf{y_{i+1}}\ldots\mathbf{y_m}\in S_2\right]\\ &=P_i+\mathbb{P}\left[D'(O\ \mathbf{y_i}\mathbf{y_{i+1}}\ldots\mathbf{y_m},i-1)=1|\mathbf{y_{i+1}}\ldots\mathbf{y_m}\in S_2\right]\cdot\mathbb{P}\left[\mathbf{y_{i+1}}\ldots\mathbf{y_m}\in S_2\right],\end{aligned}$$

and thus

$$\begin{aligned}&\mathbb{P}\left[D'(O\ \mathbf{y_i}\mathbf{y_{i+1}}\ldots\mathbf{y_m},i-1)=1|\mathbf{y_{i+1}}\ldots\mathbf{y_m}\in S_2\right]\cdot\mathbb{P}\left[\mathbf{y_{i+1}}\ldots\mathbf{y_m}\in S_2\right]\\ &\qquad\ge P_{i-1}-P_i\ge 1/(m\cdot|D|).\end{aligned}$$

In particular, $\mathbb{P}\left[\mathbf{y_{i+1}}\ldots\mathbf{y_m}\in S_2\right]>0$ and
$\mathbb{P}\left[D'(O\ \mathbf{y_i}\ \mathbf{y_{i+1}}\ldots\mathbf{y_m},i-1)=1|\mathbf{y_{i+1}}\ldots\mathbf{y_m}\in S_2\right]\ge 1/(m\cdot|D|)$, which imply there is a *fixed* $(y_{i+1}\ldots y_m)\in S_2$ such that

$$\mathbb{P}\left[D'(O\ \mathbf{y_i}\ y_{i+1}\ldots y_m,i-1)=1\right]\ge 1/(m\cdot|D|).$$

We now define another nondeterministic circuit C by $C(\mathbf{y_i}):=D'(O\ \mathbf{y_i}y_{i+1}\ldots y_m,i-1)$ with $\mathbf{y_i}\in\{0,1\}^{n+1}$ as the input variable. As D' is of sub-exponential size in N and n is polynomially related to N, C is of sub-exponential size in n.

We now argue that C breaks b in the desired sense. For infinitely many n's,

(1) $\mathbb{P}\left[C(\mathbf{y_i})=1\right]=\mathbb{P}\left[D'(O\ \mathbf{y_i}\ y_{i+1}\ldots y_m,i-1)=1\right]\ge 1/(m\cdot|D|)$, where $m\cdot|D|$ is sub-exponential in n;

(2) $\mathbb{P}\left[C(b(\mathbf{x_i}))=1\right]=\mathbb{P}\left[D'(O\ b(\mathbf{x_i})\ y_{i+1}\ldots y_m,i-1)=1\right]=0$, because $y_{i+1}\ldots y_m\in S_2$ implies there is no $x_1,\ldots,x_i$ such that $D(b(x_1)\ldots b(x_i)y_{i+1}\ldots y_m)=1$.

Since C breaks b in the above sense, we reach a contradiction with the assumption that b is a demi-bit. ◀

The condition $c < 1$ in Demi-Bit Stretching Algorithm 24 guarantees $n = N^{1-c}$ is polynomially related to N and an infinite monotone sequence of N yields an infinite monotone sequence of n.

A key step that makes this proof work is that nondeterministically guessing seeds $x_1, \ldots, x_{i-1}$ in $b(x_1) \ldots b(x_{i-1}) z_i$ preserves the "randomness-structure" of $b(x_1) \ldots b(x_{i-1}) z_i$, in the sense that: when $z_i = b(\cdot)$ is pseudorandom, the nondeterministic guess preserves the form $b(\cdot) \ldots b(\cdot) b(\cdot)$ (i.e., i equal-length pseudorandom chunks); and when $z_i = y$ is truly random, it preserves the form $b(\cdot) \ldots b(\cdot) y$ (i.e., $i - 1$ equal-length pseudorandom chunks followed by a truly random chunk y of the same length). For common stretching algorithms that produce exponentially many new bits (e.g., recursively applying a one-bit generator), it is unclear how to use nondeterminism in a way that respects the "randomness-structure" of a given string. Nevertheless, the new proof technique may inspire researchers to further explore the stretchability of demi-bits. On the other hand, the fact could also be that there is a specific demi-bit which cannot be stretched to exponentially many demi-bits by the standard stretching algorithms which are applied to super-bits and strong PRGs.

4.1 Applications in average-case complexity

In this section we show that Theorem 25 implies an equivalence between two different parametric regimes of zero-error average-case hardness of time-bounded Kolmogorov complexity against NP/poly machines.

We need the following: a ***hitting set generator against a class of decision problems*** $\mathcal{C} \subseteq 2^{\{0,1\}^N}$ is a function $g : \{0,1\}^n \to \{0,1\}^N$, for $n < N$, such that the image of g *hits* (namely, intersects) every dense enough set A in $\mathcal{C}$ (that is, $|A| \geq \frac{2^N}{N^{O(1)}}$).

As observed by Santhanam [26]:

▶ **Proposition 26** ([26]). *Let $n < N$. A hitting set generator $g : \{0,1\}^n \to \{0,1\}^N$ computable in the class $\mathcal{D}$ against* NP/poly *exists iff there exists a demi-bit $b : \{0,1\}^n \to \{0,1\}^N$ computable in $\mathcal{D}$ (against* NP/poly*).*

Proof. If $g : \{0,1\}^n \to \{0,1\}^N$ is a hitting set generator against the class $\mathcal{C}$ of decision problems decidable by nondeterministic polynomial-size circuits, it is also a demi-bit in the sense that no machine $C \in \mathcal{C}$ of polynomial-size $|C| = N^{O(1)}$ can break g, since otherwise $\mathbb{P}[C(U_N) = 1] \geq 1/N^{O(1)}$ and $\mathbb{P}[C(g(U_n)) = 1] = 0$, contradicting the assumption that g is a hitting set generator against $\mathcal{C}$. Conversely, if b is a demi-bit, than it is also a hitting set generator against $\mathcal{C}$, because if a circuit C in $\mathcal{C}$ outputs 1 to a dense enough set of inputs it must also output 1 on a string in the image of b, or else C would break the demi-bit. ◀

Note that Demi-Bit Stretching Algorithm 24 applies also to demi-bits computable in *uniform* polynomial-time (the stretching algorithm is uniform, assuming the original demi-bit is, since it simply applies the demi-bit on different parts of the input). This is important for us, since to talk about Kolmogorov complexity we need machines to be of fixed size, even when the input length changes. Notice, on the other hand, that the proof that the stretching algorithm preserves its hardness necessitates that the adversary D is *non*-uniform (this is the reason in Theorem 28 we work against NP/poly adversaries).

We define the ***t-bounded Kolmogorov complexity of string x***, denoted $\mathrm{K}^t(x)$, to be the minimal length of a string D such that the universal Turing machine $U(D)$ (we fix some such universal machine) runs in time at most t and outputs x. See [3] for more details about time-bounded Kolmogorov complexity and Definition 9 there for the definition of time-bounded Kolmogorov complexity of strings (that definition actually produces the ith bit of the string x given an index i and D as inputs to U, but this does not change our result).

Recall Definition 6 of the languages $\mathrm{K}^t[s]$ and $\mathrm{K}^{\mathsf{poly}}[s(n)]$.

We also need to define precisely the concept of zero-error average-case hardness against the class $\mathsf{NP/poly}$ (equivalently, nondeterministic circuits as in Definition 8).

▶ **Definition 27** (Zero-error average-case hardness against $\mathsf{NP/poly}$)**.** *We say that a language $L \in \{0,1\}^*$ is* **zero-error average-case easy** *for* $\mathsf{NP/poly}$ *if there is an* $\mathsf{NP/poly}$ *machine for which all the following hold: (i) every computation-path terminates with either a Yes, No or Don't-Know state; (ii) for a given input x no two distinct computation-paths terminate with both Yes and No; (iii) we say that the machine answers Yes (No) on input x if there* exists *a computation-path terminating in Yes (resp. No) given x; (iv) otherwise (namely, all computation-paths given input x terminate in Don't-Know) we say that the machine* does not know *the answer for x; (v) the machine never makes a mistake when answering Yes or No (on the other hand, it can answer Don't-Know on either members of L or non-members); and (vi) the machine (correctly) answers Yes or No on at least a polynomial fraction of inputs in L (i.e., at least $2^n/n^c$ of strings in $L \cap \{0,1\}^n$, for every sufficiently large n and for some fixed constant c independent of n). If L is not zero-error average-case easy for* $\mathsf{NP/poly}$ *we say that L is* **zero-error average-case hard** *against* $\mathsf{NP/poly}$.

The main equivalence is the following:

▶ **Theorem 28** (Equivalence for average-case time-bounded Kolmogorov complexity)**.** $\mathrm{K}^{\mathsf{poly}}[n - O(1)]$ *is zero-error average-case hard against* $\mathsf{NP/poly}$ *machines iff* $\mathrm{K}^{\mathsf{poly}}[n - o(n)]$ *is zero-error average-case hard against* $\mathsf{NP/poly}$ *machines.*

Proof. By Theorem 25 and the equivalence of (uniform polytime computable) demi-bits and hitting set generators (HSGs) against $\mathsf{NP/poly}$ (Proposition 26), it suffices to show that for a size function $s : \mathbb{N} \to \mathbb{N}$ with $s(n) < n - O(1)$, $\mathrm{K}^{\mathsf{poly}}[s(n)]$ is zero-error average-case hard against $\mathsf{NP/poly}$ machines iff there is a HSG computable in uniform polytime $H : \{0,1\}^{s(n)} \to \{0,1\}^n$ against $\mathsf{NP/poly}$.

($\Longleftarrow$). Assume that $H : \{0,1\}^{s(n)} \to \{0,1\}^n$ is a HSG, computable in uniform polytime, against $\mathsf{NP/poly}$. Then, for every constant k (independent of n) and sufficiently large n, $\mathrm{Im}(H) \cap \{0,1\}^n$ intersects all $\mathsf{NP/poly}$-computable sets $A_n \subseteq \{0,1\}^n$ for which $|A_n| \geq 2^n/n^k$ for all n. We need to show that for every constant c, $\mathrm{K}^{n^c}[s(n)] \subseteq \{0,1\}^n$ is zero-error average-case hard for $\mathsf{NP/poly}$. We show that there is no $\mathsf{NP/poly}$ machine that answers (correctly) one of Yes or No answers on at least $2^n/n^k$ input strings from $\{0,1\}^n$, and on the rest input strings in $\{0,1\}^n$ answers Don't-Know, and moreover makes no mistakes. Assume otherwise, then there is an $\mathsf{NP/poly}$ machine that answers (correctly) No for at least $2^n/n^{O(k)}$ input strings from $\{0,1\}^n$; this is because most input strings do not have short time-bounded Kolmogorov complexity, that is, a polynomial fraction of the inputs $x \in \{0,1\}^n$, for every n, are not in $\mathrm{K}^{\mathsf{poly}}[s(n)]$, for $s(n)$ between $|x|^\varepsilon$ and $|x|$ (for a constant $0 < \varepsilon < 1$; see [3, Section 2.6] and references therein). Hence, there is an $\mathsf{NP/poly}$ machine that accepts (correctly) at least $2^n/n^{O(k)}$ "hard strings" from $\{0,1\}^n$ (namely, strings not in $\mathrm{K}^{\mathsf{poly}}[n]$), and rejects all other strings in $\{0,1\}^n$: in particular, the $\mathsf{NP/poly}$ machine guesses a witness to the effect that the input string is hard, and if the witness is correct it accepts, and otherwise it rejects.

We thus get a contradiction to H being a HSG against $\mathsf{NP/poly}$: there is a dense $\mathsf{NP/poly}$-language containing at least $2^n/n^{O(k)}$ "hard strings" from $\{0,1\}^n$. But if D is such an $\mathsf{NP/poly}$ machine for this language, then D breaks the HSG H: for every string in $\mathrm{Im}(H)$ the machine D Rejects, since it has a small $\mathrm{K}^{\mathsf{poly}}$ complexity by the assumption that H is computable in uniform polytime (in other words, every string x of length n in $\mathrm{Im}(H)$ is such that $x \in \mathrm{K}^{n^r}[s(n)+O(1)]$, for some constant r independent of n, by assumption that H is computable in uniform polytime). Hence, H does not hit the dense NP/poly-set defined by D, a contradiction.

($\Longrightarrow$). We assume that $\mathrm{K}^{\mathsf{poly}}[s(n)]$ is zero-error average-case hard against NP/poly. Let $H : \{0,1\}^{s(n)} \to \{0,1\}^n$ be a mapping defined so that the input $x \in \{0,1\}^{s(n)}$ is fed into a universal Turing machine to be ran in time n^k, for some constant k independent of n, and the output of the universal machine is a string of length n (or the string $0\cdots0$ of n zeros if the algorithm does not terminate after n^k steps). We show that H is a HSG against NP/poly (computable in uniform $n^{O(1)}$-time, by assumption).

Assume by way of contradiction that H is not a HSG against NP/poly. Then, there is an NP/poly machine D that accepts at least $2^n/n^{O(1)}$ strings in $\{0,1\}^n$, but rejects every string in $\mathrm{Im}(H)$. Therefore, there is an NP/poly machine D that correctly accepts at least $2^n/n^{O(1)}$ strings $x \in \{0,1\}^n$ with $x \notin \mathrm{K}^{\mathsf{poly}}[s(|x|)]$, and rejects all other strings in $\{0,1\}^n$. This contradicts our assumption, because we can construct an NP/poly machine D' that zero-error decides on average $\mathrm{K}^{\mathsf{poly}}[s(|x|)]$: in D simply replace an Accept state with a Yes state, and a Reject state with a Don't-Know state. ◀

References

1 Michael Alekhnovich, Eli Ben-Sasson, Alexander A. Razborov, and Avi Wigderson. Pseudorandom generators in propositional proof complexity. *SIAM J. Comput.*, 34(1):67–88, 2004. (A preliminary version appeared in Proceedings of the 41st Annual Symposium on Foundations of Computer Science (Redondo Beach, CA, 2000)).

2 W. Alexi, B. Chor, O. Goldreich, and C. P. Schnorr. Rsa and rabin functions: Certain parts are as hard as the whole. *SIAM Journal on Computing*, 17(2):194–209, 1988. `doi:10.1137/0217013`.

3 Eric Allender, Harry Buhrman, Michal Koucký, Dieter van Melkebeek, and Detlef Ronneburger. Power from random strings. *SIAM J. Comput.*, 35(6):1467–1493, 2006. `doi:10.1137/050628994`.

4 Boaz Barak, Shien Jin Ong, and Salil P. Vadhan. Derandomization in cryptography. *SIAM J. Comput.*, 37(2):380–400, 2007. `doi:10.1137/050641958`.

5 M. Blum and S. Micali. How to generate cryptographically strong sequences of pseudo-random bits. *SIAM J. Comput.*, 13(4):850–864, November 1984. `doi:10.1137/0213053`.

6 Akshay Degwekar, Vinod Vaikuntanathan, and Prashant Nalini Vasudevan. Fine-grained cryptography. Cryptology ePrint Archive, Paper 2016/580, 2016. URL: `https://eprint.iacr.org/2016/580`.

7 O. Goldreich, S. Goldwasser, and S. Micali. How to construct random functions. *J. ACM*, 33(4):792–807, August 1986. `doi:10.1145/6490.6503`.

8 O. Goldreich and L. A. Levin. A hard-core predicate for all one-way functions. In *21st Annual ACM Symposium on Theory of Computing*, STOC '89, pages 25–32, New York, NY, USA, 1989. Association for Computing Machinery. `doi:10.1145/73007.73010`.

9 Oded Goldreich. *Foundations of cryptography I: Basic Tools.* Cambridge: Cambridge University Press, 2001.

10 S. Goldwasser and S. Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28(2):270–299, 1984. `doi:10.1016/0022-0000(84)90070-9`.

11 Shuichi Hirahara. Characterizing average-case complexity of PH by worst-case meta-complexity. In Sandy Irani, editor, *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020*, pages 50–60. IEEE, 2020. `doi:10.1109/FOCS46700.2020.00014`.

12 J. Håstad, R. Impagliazzo, L. A. Levin, and M. Luby. A pseudorandom generator from any one-way function. *SIAM Journal on Computing*, 28(4):1364–1396, 1999. doi:10.1137/S0097539793244708.

13 R. Impagliazzo and M. Naor. Efficient cryptographic schemes provably as secure as subset sum. In *30th Annual Symposium on Foundations of Computer Science*, pages 236–241, 1989. doi:10.1109/SFCS.1989.63484.

14 B. S. Kaliski. Elliptic curves and cryptography : a pseudorandom bit generator and other tools. *Phd Thesis Mit*, 2005.

15 Adam R. Klivans and Dieter van Melkebeek. Graph nonisomorphism has subexponential size proofs unless the polynomial-time hierarchy collapses. *SIAM J. Comput.*, 31(5):1501–1526, 2002. doi:10.1137/S0097539700389652.

16 Jan Krajíček. Dual weak pigeonhole principle, pseudo-surjective functions, and provability of circuit lower bounds. *The Journal of Symbolic Logic*, 69(1):265–286, 2004.

17 Jan Krajíček. *Forcing with random variables and proof complexity*, volume 382 of *London Mathematical Society Lecture Notes Series.* Cambridge Press, 2010.

18 Yanyi Liu and Rafael Pass. On one-way functions and kolmogorov complexity. In Sandy Irani, editor, *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020*, pages 1243–1254. IEEE, 2020. doi:10.1109/FOCS46700.2020.00118.

19 M. Luby. *Pseudorandomness and Cryptographic Applications*, volume 1. Princeton University Press, 1996. URL: http://www.jstor.org/stable/j.ctvs32rpn.

20 N. Nisan and A. Wigderson. Hardness vs randomness. *Journal of Computer and System Sciences*, 49(2):149–167, 1994. doi:10.1016/S0022-0000(05)80043-1.

21 J. Pich. Learning algorithms from circuit lower bounds. *CoRR*, abs/2012.14095, 2020. arXiv:2012.14095.

22 Jan Pich and Rahul Santhanam. Why are proof complexity lower bounds hard? In *60th Annual IEEE Symposium on Foundations of Computer Science FOCS 2019, November 9-12, 2019, Baltimore, Maryland USA*, 2019.

23 A. A. Razborov and S. Rudich. Natural proofs. *Journal of Computer and System Sciences*, 55(1):24–35, 1997. doi:10.1006/jcss.1997.1494.

24 Alexander A. Razborov. Pseudorandom generators hard for k-DNF resolution and polynomial calculus resolution. *Annals of Mathematics*, 181:415–472, 2015.

25 S. Rudich. Super-bits, demi-bits, and NP/qpoly-natural proofs. *Journal of Computer and System Sciences*, 55:204–213, 1997.

26 Rahul Santhanam. Pseudorandomness and the minimum circuit size problem. In Thomas Vidick, editor, *11th Innovations in Theoretical Computer Science Conference, ITCS 2020, January 12-14, 2020, Seattle, Washington, USA*, volume 151 of *LIPIcs*, pages 68:1–68:26. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.ITCS.2020.68.

27 Rahul Santhanam. Personal communication, 2022.

28 Nicollas Sdroievski and Dieter van Melkebeek. Instance-wise hardness versus randomness tradeoffs for arthur-merlin protocols. *The Electronic Colloquium on Computational Complexity (ECCC)*, 2023. URL: https://eccc.weizmann.ac.il/report/2023/029/.

29 Ronen Shaltiel and Christopher Umans. Simple extractors for all min-entropies and a new pseudorandom generator. *J. ACM*, 52(2):172–216, 2005. doi:10.1145/1059513.1059516.

30 Iddo Tzameret and Lu-Ming Zhang. Stretching demi-bits and nondeterministic-secure pseudorandomness, 2023. See also Elec. Coll. Comput. Complexity https://eccc.weizmann.ac.il/report/2023/057/. arXiv:2304.14700.

31 A. C. Yao. Theory and application of trapdoor functions. In *23rd Annual Symposium on Foundations of Computer Science*, FOCS '82, pages 80–91, 1982. doi:10.1109/SFCS.1982.45.