What Can You Do with an Oracle for the Random Strings?

Nathan Sheffield and William Wang

Apr 2024

1 Introduction

1.1 Friendly discussion

The notion of ***Kolmogorov complexity*** captures the sense in which a string looks inherently random or unstructured to any computable process. We'll say that a string looks "random" if there's no program of length shorter than the string itself that prints it – if you pulled some random numbers out of your hat, chances are they'd have this property, but a very structured string like "01010101010101..." does not, since it can be generated quite succinctly with a simple `for` loop. Call the set of all such random-looking strings R_K. (Of course, this notion should depend to some degree on what language you're writing these programs in – more on that soon.)

A first question might be: can we write down an algorithm to determine on input x whether $x \in R_K$? The answer to this question is a resounding "no": if we could, then we could modify that algorithm to say "search through all strings until we find a random-looking one 100 times longer than this program's code, and print that", which would give a short program to generate a long random-looking string – contradiction! A second, more subtle, follow-up question you might ask is: ok, so R_K is hard to compute – but is it hard in a useful way? That is, if I had some magic oracle that could answer any question of the form "is $x \in R_K$?", would this be useful for the sorts of computations I might actually want to do? It seems like this could be the case – for instance, if I'm a deterministic machine running in polynomial time, I could use R_K queries to find a string that looks random, and then simulate a randomized machine. But it's not immediately obvious if that's all I can do, or if maybe I can solve any computable problem at all, or somewhere in-between.

This survey will be dedicated to trying to answer this question. We'll outline a line of results that show, perhaps surprisingly, that the question of what can be *efficiently* reduced to R_K gives rise to some interesting concrete complexity classes.

1.2 Outline of the survey

As we defined it, the question of Kolmogorov complexity depends on what programming language you're using. The way we actually formalize this is by fixing a specific universal Turing machine U, and looking at the length of input to that Turing machine required to generate a given output. Also note that we're mainly going to be considering the prefix-free complexity, K_U, which has a slightly different definition from the plain Kolmogorov complexity C_U. For the reader unfamiliar with these things, they'll be defined in Section 2 – for now just think of R_{K_U} as meaning "the set of strings with no shorter descriptions than themselves in programming language U".

The main goal of this survey will be to show that $\mathsf{EXP}^{\mathsf{NP}} \subseteq \bigcap_U \mathsf{P}^{R_{K_U}} \subseteq \mathsf{EXPSPACE}$. That is, the class of problems that can be solved in polynomial time with an oracle for the random-looking strings **no matter what "programming language" you use** to define "random-looking" is a concrete, computable complexity class, lying somewhere between the lower levels of the exponential time hierarchy and exponential space. The EXPSPACE inclusion comes from combining the results of Allender, Friedman and Gasarch [AFG13] with those of Cai et al [Cai+14]; the arguments used come from computability theory. The $\mathsf{EXP}^{\mathsf{NP}}$ containment, a surprising advance on previously-known bounds, was proven by Hirahara [Hir20] – it involves constructing a PRG based on an $\mathsf{EXP}^{\mathsf{NP}}$ complete problem, and then using Hirahara's earlier result on poly-time "selectors" for $\mathsf{EXP}^{\mathsf{NP}}$ to show that a distinguisher for the PRG gives a uniform algorithm for the original problem [Hir15].

We will also briefly discuss what's known for the plain complexity, R_{C_U}. The story here is not so nice as for prefix-free complexity: Hirahara's proof of $\mathsf{EXP}^{\mathsf{NP}}$ containment still holds for $\cap_U \mathsf{P}^{R_{C_U}}$, but the proof of $\mathsf{EXPSPACE}$ inclusion no longer works. It's not even known whether $\cap_U \mathsf{P}^{R_{C_U}}$ contains the halting problem – Allender has offered a \$1000 bounty to anyone who can resolve this question (could be you, dear reader!) [All23]. Most of the results known for R_{C_U} involve ***disjunctive truth-table reductions***, where an algorithm asks a list of questions non-adaptively, and accepts if the oracle says `YES` on any query. In this setting, R_{C_U} behaves meaningfully differently from R_{K_U}, which suggests that perhaps our lack of knowledge about efficient reductions to R_{C_U} compared to R_{K_U} comes from fundamental differences as opposed to just technical limitations of known arguments [Kum96; ABK06; MP02].

Finally, we will state some recent results on approximations to the Kolmogorov-random strings. That is, you have an oracle that promises to say `YES` if a string has very high Kolmogorov complexity, and `NO` if a string has very low Kolmogorov complexity, but there's some allowable error window where it's allowed to do either. This is a rather nice notion because the choice of universal Turing machine U only affects the Kolmogorov complexity by a constant, and the difference between the prefix-free K_U and plain C_U complexities is at most $O(\log n)$, so as long as the allowable error window is sufficiently large we don't have to worry about which of these definitions we're using. Saks and Santhanam showed that ***honest*** (a technical condition forbidding queries on very small strings) single-call poly-time reductions to approximate $\widetilde{R}_K$, if the error window is larger than $\omega(\log n)$, can't do anything outside of P [SS22]. Building on their work, Allender, Hirahara, and Tirumala recently showed that, for any error window of size $\omega(\log n)$ but $n^{o(1)}$, the class of problems honestly reducible to $\widetilde{R}_K$ in randomized poly-time is *exactly* the class NISZK, of problems with non-interactive statistical zero knowledge proofs [AHT23]. There's quite a long string of modifiers in this statement ("honest", "randomized", "non-interactive", "statistical", etc), but it is interesting as a case where a notion of reductions to the the random strings exactly characterizes a previously-studied complexity class. It's also interesting because Allender has proposed an optimistic approach to separate NL from NP using the log-space version of this result (which they also showed) – as far as the authors of this survey are aware, that approach has yet to be ruled out.

2 Preliminaries

Definition 1. Given a Turing machine M, for any $x \in \{0,1\}^*$ we say that the ***(plain) Kolmogorov complexity*** C_M of x is

$$C_M(x) = \min\{|p| : M(p) = x\}$$

where $|p|$ denotes the length of p. p forms a ***description*** of x relative to machine M.

This definition of complexity captures the idea of how simply we can express x, if we have some underlying machine M. Typically we think of C as being defined relative to a universal Turing machine.

Definition 2. Define a *universal Turing machine* U to be a Turing machine such that for all other machines M, there exists a constant c_M such that

$$C_U(x|y) \le C_M(x|y) + c_M, \ \forall\ x,y$$

This is nice, because it means C_U captures some "universal" notion of complexity – defining relative to any other machine will differ only by a constant.

While C is a useful notion of complexity, we will often instead focus on the related notion of prefix-free Kolmogorov complexity.

Definition 3. A prefix-free Turing machine is any machine M with the property that, if M halts on an input x, it does not halt on input xy for any y. That is, no string on which it halts is a prefix of another. The ***prefix-free complexity*** $K_M(x)$ is again defined as

$$K_M(x) = \min\{|p| : K(p) = x\}.$$

Like before, we can show the existence of universal prefix-free Turing Machines, so we can fix an arbitrary universal machine and just speak of the prefix-free Kolmogorov complexity of a string. Despite looking slightly different, prefix-free Kolmogorov complexity measures roughly the same quantity as standard Kolmogorov complexity:

Theorem 1. Fixing arbitrary universal machines and omitting $O(1)$ terms, we have

$$C(x) \leq K(x) + 2\log_2 C(x)$$

So, it seems that prefix-free complexity is just a slightly more obfuscated version of our theoretically clean standard complexity. In that case, why do we care? While the definition is somewhat more complicated, prefix-free Kolmogorov complexity is generally preferred as an object of study, as it has many convenient properties which do not hold for standard Kolmogorov complexity:

Theorem 2. Once again ignoring $O(1)$ terms, we have:

- **Subadditivity:** $K(\langle x,y\rangle) \leq K(x) + K(y)$
- **Symmetry:** $K(\langle x,y\rangle) = K(\langle y,x\rangle)$
- **Kraft Inequality:** $\sum_{x\in\{0,1\}^*} 2^{-K(x)} \leq 1$

One property in particular that we will use is the relationship between the prefix-free complexity and what Allender, Friedman and Gasarch call a "prefix-free entropy function". Essentially, we have that any computably-enumerable sub-probability measure that's nonzero on all strings corresponds to the complexity under some prefix machine.

Definition 4. For any function $f:\{0,1\}^* \to \mathbb{N}$, we define the ***overgraph***

$$\mathrm{ov}(f) = \{(x,y) : f(x) \leq y\}$$

Definition 5. We call a function $f:\{0,1\}^* \to \mathbb{N}$ a ***prefix-free entropy function*** if $\sum_{x\in\{0,1\}^*} 2^{-f(x)} \leq 1$ and $\mathrm{ov}(f)$ is computationally enumerable.

Lemma 1 (Allender, Friedman, Gasarch [AFG13])**.** For any prefix-free entropy function f, there exists a prefix-free machine M such that $f(x) = K_M(x) - 2$.

3 The K-random strings won't let you beat EXPSPACE

Our goal in this section will be to show an upper bound on the power of an oracle for the random strings — namely, that $\bigcap_U \mathsf{P}^{R_{K_U}} \subseteq \mathsf{EXPSPACE}$. Our proof strategy for doing so will be an approach common in computability theory called a ***finite injury*** argument. Fixing some hard language L, we want to show the existence of a universal prefix machine U such that no polynomial time reduction succeeds in reducing L to R_{K_U}. We can think of this as a countably infinite list of ***requirements*** that the machine U we construct must satisfy: for every possible reduction, to R_{K_U}, we need that reduction's output to differ from L on some input x. In a standard diagonalization argument, we might ensure that over the course of our construction each of these requirements eventually becomes satisfied, and that once a requirement becomes satisfied it is never subsequently "injured" (i.e. never stops being satisfied). In a finite injury argument, we permit the construction to occasionally injure previously-satisfied requirements, but show that every time a requirement is injured it will eventually be re-established, and that any given requirement can only be injured a finite number of times. This ensures that, as the construction runs off to infinity, eventually each requirement stops changing and remains satisfied forever. A typical approach to finite injury arguments is to give each of the requirements (in our case, each potential reduction we want to thwart) a ***priority*** based on what position they are in some canonical ordering, and show that we can only injure one requirement through actions taken to satisfy a higher-priority one — since every requirement has only a finite number of higher-priority requirements, this ensures each is injured only a finite number of times.

In order to get control over what R_{K_U} looks like, we will use Lemma 1. Note that this fact is specific to the prefix-free complexity, which is why we only know the main result for R_{K_U} and not R_{C_U}. That is, instead of giving a universal prefix machine directly, we will show a procedure for enumerating values into the overgraph of a prefix-free entropy function, using the fact that this can be turned into a corresponding prefix machine.

The actual statement we prove will be about ***non-adaptive*** reductions — we say $A \leq_{tt}^{\mathsf{P}} B$ if there exists a poly-time algorithm for A that makes poly many queries to a B-oracle non-adaptively, meaning that future queries can't depend on previous responses. From the fact that $\bigcap_U \{A : A \leq_{\text{tt}}^{\text{p}} R_{K_U}\} \subseteq \mathsf{PSPACE}$, we will be able to deduce $\bigcap_U \mathsf{P}^{R_{K_U}} \subseteq \mathsf{EXPSPACE}$ as an easy corollary.

Theorem 3 (Allender, Friedman, Gasarch [AFG13], Cai et al [Cai+14])**.**

$$\bigcap_U \{A : A \leq_{\text{tt}}^{\text{p}} R_{K_U}\} \subseteq \mathsf{PSPACE}$$

Proof. We will omit the proof, given by Cai et al, that $\bigcap_U \{A : A \leq_{\text{tt}} R_{K_U}\} \subseteq \Delta_0^1$, and instead only show that $\Delta_0^0 \bigcap_U \{A : A \leq_{\text{tt}}^{\text{p}} R_{K_U}\} \subseteq \mathsf{PSPACE}$ — that is, that $\bigcap_U \{A : A \leq_{\text{tt}}^{\text{p}} R_{K_U}\}$ contains no *decidable* languages outside of PSPACE[1]. For this purpose it suffices to find, for any fixed $L \in \Delta_0 \setminus \mathsf{PSPACE}$, a universal prefix machine U such that $L \not\leq_{\text{tt}}^{\text{p}} R_{K_U}$. In fact, we will find a U such that $L \not\leq_{\text{tt}}^{\text{p}} \text{ov}(K_U)$[2]. By making $|x|$ many queries to $\text{ov}(K_U)$, one can compute the exact value of $K_U(x)$ — so an oracle for $\text{ov}(K_U)$ is at least as strong as an oracle for R_{K_U}.

In order to find such a U, we will construct a function H such that $H-2$ is a prefix-free entropy function, which will by Lemma 1 give a corresponding universal prefix machine with $K_U = H$. To build H, we'll start with some standard prefix-free Kolmogorov complexity measure K, defined relative to an arbitrary universal machine. Then, we'll set

$$H(x) = \min(K(x)+5, F(x)+3)$$

for some computably enumerable function F we construct. The idea is, if we started with $F(x)$ initialized to be infinitely large everywhere, then $H(x)-2$ would be a prefix-free entropy function with some "wiggle-room", in the sense that by the Kraft inequality we'd have $\sum_{x\in\{0,1\}^*} 2^{-(H(x)-2)} = \frac{1}{8}\sum_{x\in\{0,1\}^*} 2^{-K(x)} \leq \frac{1}{8}$. As long as we don't go "over-budget", this wiggle-room will allow us to decrease H at the indices we care about, which we'll do through our choice of F. Since $\text{ov}(F)$ and $\text{ov}(K)$ are computably enumerable, $\text{ov}(H)$ is computably enumerable — so the only condition we need to maintain to ensure that H corresponds to some prefix-free Kolmogorov measure is that $\sum_{x\in\{0,1\}^*} 2^{-(H(x)-2)} \leq 1$, or in other words

$$\sum_{x\in\{0,1\}^*} 2^{-H(x)} \leq \frac{1}{4}. \tag{*}$$

Constructing F in a computably enumerable fashion to both thwart all Turing reductions and preserve (*) will require some care, because the values of K aren't computable. We do, however, know that $\text{ov}(K)$ is computably enumerable. So, we'll construct F in stages. That is, we'll run the enumerator for $\text{ov}(K)$, and every time it prints a new string, we'll possibly enumerate things into F in response. At every stage, we let F^* and K^* be the "current versions" of F and K, respectively, and every time an element is enumerated into $\text{ov}(F)$ or $\text{ov}(K)$ we update these functions accordingly. Note that F^*, K^* will be monotone in the sense that $F^*(x)$ (resp. $K^*(x)$) can only decrease as we continue enumerating elements from $\text{ov}(F)$ (resp. $\text{ov}(K)$). At the beginning, we set

[1]The proof by Cai et al follows a similar format to this one, adapting the proof of Muchnik that R_{K_U} fails to be truth table complete for some U[MP02]. However, the details are involved and we've chosen not to include them here.

[2]Recall that we define $(z,r) \in \text{ov}(K_U)$ if $K_U(z) \leq r$.

$F^*(x) = 100|x| + 100$, and $K^*(x)$ to be the empty function. Note that, at any point, F^*,K^* naturally define $H^*(x) := \min(K^*(x)+5, F^*(x)+3)$.

We can now think of the updates to F^* and K^* as a game to determine the value of a truth table reduction. In particular, consider a circuit λ from a truth table reduction for some input x with input queries of the form "$(z_i,r_i) \in \text{ov}(H^*)$?" By updating F^*,K^*, we change the value of H^*, which may change the output of the circuit. To formalize these ideas, we define a graph G with nodes labeled with tuples (h,b) where h is a function $\{z_i\} \to \mathbb{N}$ and b is the output of λ if H^* agrees with h on $\{z_i\}$. We draw an edge between $(h_1,b_1) \to (h_2,b_2)$ whenever $h_2(z_i) \le h_1(z_i)$ for all z_i. Now, we can think of F^* and K^* playing a game where they place a token on the node corresponding to the starting value of H^* over $\{z_i\}$, and every update to either F^* or K^* moves the token along an edge. These moves always occur along edges since updating F^* and K^* can only decrease the image of $H^*(x)$. To further constrain the game, we will define an associated "cost" for moving from h_1 to h_2, equal to

$$\sum_{x \in \{z_i\}} 2^{-h_2(x)} - 2^{-h_1(x)}$$

which is how much $\sum 2^{-H^*(x)}$ changes by under this move. Over the game, both players will accrue cost, and we will set a rule whereby players cannot exceed a total cost of ϵ. We also give the players roles, one of them the YES player who is trying to finish the game at a node with $b = 1$, and one as the NO player who is trying to finish at $b = 0$. In our actual procedure, we will give F^* the role which makes the truth table reduction not work for L. If we assume that the YES player always starts first, note that there can only be a finite number of moves in the game, so either the YES or NO player has a winning strategy, meaning that they can force a win no matter what the other player does, assuming both players play by the rules.

Our approach now will be to enumerate $\text{ov}(K)$, interpret these updates to K^* as moves on G, and then enumerate F^* in a way such that the game ends in a state (h,b) where $b=1 \iff x \notin L$, meaning that F^* prevents the truth table reduction corresponding to λ from working for L. Note that we have no control over the value of $\text{ov}(K)$ or how it is enumerated, so it is possible that the moves K^* takes will not respect our total cost $\le \epsilon$ condition. In this case, we say that K^* is "cheating" at the game. Part of our analysis will also be devoted to showing that K^* cannot cheat too much.

With the general idea of a game between F^* and K^* defined, we will now flesh out our procedure a bit more. First, enumerate all possible polynomial-time truth table reductions $\gamma_1,\gamma_2,\gamma_3,\ldots$. Our ultimate goal is to produce F,H such that $L \not\le^{\text{p}}_{\text{tt}} \text{ov}(H)$, and we can break this goal into requirements $R_1,R_2,\ldots$, defining R_e as the requirement that γ_e is not a polynomial-time truth table reduction of L to $\text{ov}(H)$. For each γ_e on input x, we can construct the corresponding circuit $\lambda_{e,x}$.

At any given point in our procedure, we will maintain a list of games corresponding to the $\lambda_{e,x}$, with at most one for each value of e, and as we receive more enumerated values from $\text{ov}(K)$, we will move K^* in all these games and respond accordingly. As a technical note, we want these games to be somewhat independent. So, when constructing a game on $\lambda_{e,x}$ with queries $\{(z_i,r_i)\}$, we will make the following modifications:

1. If $H^*(z_i) \le r_i$, note that future modifications of H^* will not change the input value for query (z_i,r_i) which will always remain 1, since $H^*(z_i)$ can only decrease. So, just set it to a constant in $\lambda_{e,x}$ and simplify the circuit.
2. If z_i is part of a circuit $\lambda_{e',x'}$ for $e' < e$, then set the input value for query (z_i,r_i) to 0. So, defining X_e to be the set of $\{z_i\}$ for which the simplified $\lambda_{e,x}$ actually runs on, we can express X_e iteratively by

$$X_e = \{z_i\} \setminus \bigcup_{e' < e} X_{e'}$$

The reasoning behind this is that 0 will be the correct answer to the corresponding query unless an element from F^* or K^* gets enumerated which causes $H^*(z_i)$ to be at most r_i. But, in our construction, if this ever happens, we will destroy this instance of the game as it is now "out of date."

We use $\lambda_{e,x}$ to denote the simplified circuit. Define $\mathcal{G}_{e,x}$ to be the game played on $\lambda_{e,x}$, and $\text{val}(\mathcal{G}_{e,x},\epsilon)$ to be 1 if the YES player has a winning strategy, and 0 otherwise with cost limit parameter ϵ. Since our ultimate goal is to satisfy all the requirements R_e, we say that R_e is satisfied if we have a game $\mathcal{G}_{e,x}$ at (h,b) with $b \neq L(x)$, meaning x is a witness to γ_e not being a valid truth table reduction.

We will now describe our iterative procedure for constructing F and H. On step 0, we will start with $F^*(x) = 2|x|+3$, $K^*(x)$ is the empty function, and set a variable $i_e = 0$ for all $e \geq 0$. The purpose of the i_e should not be apparent right now, but it will control the value of ϵ in created games such that our construction will work. Finally, we instantiate HEAP, an object which just returns strings in $\{0,1\}^*$ in lexicographical order, starting with 0,1,00,.... We will query HEAP as we construct games to figure out what input we construct the circuit on.

Now, on a given step s, we begin by taking the next enumerated element (x',y') from $\text{ov}(K)$, update K^* by setting $K^*(x') \leftarrow \min(K^*(x'),y')$, and update H^* accordingly. For each $1 \leq e \leq s$, we have two cases:

1. **There does not exist a game corresponding to γ_e:** In this case, we will construct a game for e. Define

$$\epsilon_e = 2^{-e-i_e-6}$$

   Begin by drawing xs from the HEAP until we reach an x for which $\text{val}(\mathcal{G}_{e,x},\epsilon_e) \neq L(x)$, meaning that if we play to make the output of the circuit differ from $L(x)$, we have a winning strategy. We now add this game to our list of active games. It should not be clear that we will eventually draw such an x. As we will show later, if such an x does not exist, then we will have a PSPACE algorithm for deciding L, which would be a contradiction.

2. **There is currently a game corresponding to γ_e:** As the game is ongoing, we have a few cases to consider, depending on whose turn it is and how K^* has moved in it:
   (a) If the game existed in the previous step and enumerating (x',y') did not cause the game state to change, do nothing.
   (b) If the game existed in the previous step and enumerating (x',y') caused K^* to move in the game in such a way that their total cost exceeded ϵ_e, then K^* "cheated," so we destroy the game by removing it from the list of active games
   (c) If $\mathcal{G}_{e,x}$ was just created and K^* is the YES player, do nothing
   (d) If none of the above cases hold, then either we have just created the game and it is our turn, or K^* performed a legal move through enumerating (x',y') and it is now our turn. In this case, we consult our winning strategy for $\mathcal{G}_{e,x}$, and if it tells us to move, then we change H^* by enumerating an element from $\text{ov}(F)$ in such a way that simulates this move.

   Note that b) and d) are cases in which the values in X_e are changed, meaning future games $\mathcal{G}_{e',x'}$ with $e < e' \leq s$ may no longer be valid, since we had to assume things about the elements of X_e. In this case, we say that "R_e is acting" and for each $e' \in [e+1,s]$, we destroy the game associated with e' if it exists. As stated before, this is because our assumptions on queries of the form (z_i,r_i) with $z_i \notin X_{e'}$ and $H^*(z_i) > r_i$ might now be incorrect. If the game is destroyed due to case d), we will also increment $i_{e'}$ by 1.

Running this procedure for all steps $s \geq 0$ will give us a way to enumerate $\text{ov}(F)$. It remains to show that this procedure works and that it delivers on all the properties we claimed for F and H.

Claim 1: For all e, R_e only acts a finite number of times, and we end with R_e satisfied.

Proof. We will prove this claim with strong induction. Suppose that, for all $e' < e$, the corresponding games only act a finite number of times. Then, there exists a step s' such that for $s \geq s'$, no games $e' < e$ ever act. So, this means that ϵ_e is constant for $s \geq s'$ and the only way for the game corresponding to R_e to be destroyed is if K^* cheats. Note that if enumerating (x',y') from $\text{ov}(K)$ causes $H^*(x')$ to change, then the associated cost is

$$2^{-y'-5} - 2^{H^*(x')} \leq 2^{-y'-5} - 2^{-y-5} = \frac{1}{32}\left(2^{-y'} - 2^{y}\right)$$

where y was the value of $K^*(x')$ before the update. Thus, every cost c accrued by K^* translates to an increase by $32c$ in the value of $\sum_{i \in \{0,1\}^*} 2^{-K^*(x)} \leq \sum_{i \in \{0,1\}^*} 2^{-K(x)}$. In particular, every game K^* cheats on increases

$\sum_{i\in\{0,1\}^*} 2^{-K^*(x)}$ by at least $32\epsilon_e$. Since we have $\sum_{i\in\{0,1\}^*} 2^{-K(x)} \leq 1$, we get that K^* can only cheat a finite number of times, after which the game $\mathcal{G}_{e,x}$ will not be destroyed. After this point, just note that the values of H^* on X_e constrain the game to a finite number of nodes, so the game will only be acted on a finite number of times afterwards, and the proof is complete. □

Claim 2: The H generated by this procedure satisfies

$$\sum_{x\in\{0,1\}^*} 2^{-H(x)} \leq \frac{1}{8}$$

Proof. Define H_s to be the value of H^* after step s. Since $H = \lim_{s\to\infty} H_s$, we have that

$$\sum_{x\in\{0,1\}^*} 2^{-H(x)} = \sum_{x\in\{0,1\}^*} 2^{-H_0(x)} + \sum_{s=1}^{\infty} \sum_{x\in\{0,1\}^*} 2^{-H_{s+1}(x)} - 2^{-H_s(x)}$$

First, by definition,

$$\sum_{x\in\{0,1\}^*} 2^{-H_0(x)} = \sum_{x\in\{0,1\}^*} 2^{-2|x|-6} = \sum_{n=0}^{\infty} 2^n \cdot 2^{-2n-6} = \frac{1}{64} \sum_{n=0}^{\infty} 2^{-n} = \frac{1}{32}$$

The remaining terms in the sum correspond exactly to all the costs accrued as a result of the games. First, as was stated in the proof of Claim 1, every time K^* accrues a cost c, we have $\sum_{i\in\{0,1\}^*} 2^{-K^*(x)}$ increases by $32c$. Since we have

$$\sum_{i\in\{0,1\}^*} 2^{-K^*(x)} \leq \sum_{i\in\{0,1\}^*} 2^{-K(x)} \leq 1$$

the total cost accrued by K^* over all games is bounded by $1/32$.

Moving to the cost accrued by F^*, note that this can include cost accrued in games which have been destroyed, as well as current games. First, over all games destroyed by K^*, note that F^* plays by the rules, so its total cost accrued is less than that accrued by K^*, so this case is bounded by $1/32$. It remains to consider the total cost by F^* over extant games as well as games which were destroyed by case d) (ie. F^* moved on a game below it). Fix an e, and let d be the number of games destroyed due to case d). In each game, the total cost accrued by F^* does not exceed 2^{-e-i_e-6}, where i_e is the number of games corresponding to R_e destroyed beforehand due to case d). So, the total cost at e does not exceed

$$\sum_{i=0}^{d} 2^{-e-i-6} < \sum_{i=0}^{\infty} 2^{-e-i-6} = 2^{-e-5}$$

Summing over all e, the total cost accrued in this case is bounded by $\sum_{e=1}^{\infty} 2^{-e-5} = 1/32$. Hence, we have

$$\sum_{x\in\{0,1\}^*} 2^{-H(x)} \leq \frac{1}{32} + \frac{1}{32} + \frac{1}{32} + \frac{1}{32} = \frac{1}{8}$$

□

So, we have shown that our procedure will yield an H for which $\sum_{x\in\{0,1\}^*} 2^{-H(x)} \leq 1/8$, and eventually all the R_e are satisfied, meaning $L \not\leq^{\mathrm{p}}_{\mathrm{tt}} \mathrm{ov}(H)$. All that remains to be done is to make good on our promise in the first case of the procedure, namely that HEAP will eventually generate an x with the desired properties:

Claim 3: As described in the procedure, HEAP will always find an x for which $\mathrm{val}(\mathcal{G}_{e,x}, \epsilon_e) \neq L(x)$ in finite time.

Proof. Suppose the contrary, namely that there exist some e,ϵ,x_0 such that for all $x \geq x_0$ we have $\mathrm{val}(\mathcal{G}_{e,x},\epsilon) = L(x)$. Then, we claim there is a PSPACE algorithm A which decides L. First, hardcode $L(x)$ for all $x < x_0$ in A, so we can answer these values automatically. Also, store the current H^* and all the $X_{e'}$ for $e' < e$ so we have what we need to construct $\mathcal{G}_{e,x}$ for any $x \geq x_0$. Note that H^* is a function over the integers, but we can still store it in constant space since it is computable from just the values which have been enumerated thus far.

Now, to construct $\mathcal{G}_{e,x}$ on input x, note that we make at most $|x|^e$ queries to H, and on a query z_i, $H(z_i)$ is bounded above by $2|z_i|+6$. $|z_i|$ is in turn bounded by $|x|^e$, so the number of possible DAG nodes is bounded above by

$$(2|x|^e+6)^{|x|^e} \leq \Theta\left(2^{|x|^{2e}}\right)$$

So, a DAG node can be expressed in a polynomial number of bits. While we can't fit the entire DAG in PSPACE since there are an exponential number of nodes, note that since we can express each node in polynomial bits, we can express $\mathrm{val}(\mathcal{G}_{e,x},\epsilon_e) = 1$ as an alternating polynomial expression of the form "for all moves NO takes, there exists a YES move such that for all moves NO takes, etc. we will stop at a node with $b = 1$." Since alternating polynomial time is contained in PSPACE, this does indeed yield a PSPACE algorithm. □

With the above 3 claims out of the way, it is now clear that the procedure is indeed good, and we have the desired result. □

Corollary 1.

$$\bigcap_U \mathsf{P}^{R_{K_U}} \subseteq \mathsf{EXPSPACE}$$

Proof. A poly-time machine making adaptive oracles can be simulated by an exponential-time machine making non-adaptive queries, by simply querying every possible value the adaptive oracle would ever query. So, $\bigcap_U \mathsf{P}^{R_{K_U}} \subseteq \bigcap_U \{A : A \leq_{\mathrm{tt}}^{\mathrm{exp}} R_{K_U}\}$. Scaling this down gives the desired statement. Note that this in fact also shows $\bigcap_U \mathsf{NP}^{R_{K_U}} \subseteq \mathsf{EXPSPACE}$. □

4 What if you're a circuit? What if you're randomized?

The upper bound we just described shows in particular that $\bigcap_U \mathsf{P}^{R_{K_U}} \subseteq \mathsf{EXPSPACE}$. Your definition of what an "efficient algorithm" means might be different, though – you might be interested in whether we can extend this to randomized algorithms (i.e. $\bigcap_U \mathsf{BPP}^{R_{K_U}} \subseteq \mathsf{EXPSPACE}$) or poly-sized circuits (i.e. $\Delta_0 \cap \bigcap_U \mathsf{P/poly}^{R_{K_U}} \subseteq \mathsf{EXPSPACE}$). The first of these extensions is possible! However, the second is not – in fact, $\bigcap_U \mathsf{P/poly}^{R_{K_U}}$ contains every decidable language. The proof of this fact will illustrate the same basic technique we'll use to show $\mathsf{EXP}^{\mathsf{NP}} \subseteq \bigcap_U \mathsf{P}^{R_{K_U}}$ in the uniform setting. But first, as promised, let's talk BPP. We'll be loose on the exact details, but here's the general idea.

Lemma 2 (Allender et al [All+06], Buhrman et al [Buh+05])**.** For all U, $\mathsf{P}^{R_{K_U}} = \mathsf{BPP}^{R_{K_U}}$.

Proof sketch. First, suppose that the $\mathsf{P}^{R_{K_U}}$ algorithm is given, in addition to its length n input, a length $\mathsf{poly}(n)$ string with full Kolmogorov complexity. Intuitively, since this string is has high Kolmogorov complexity, it avoids any "special" properties, so we can use it in place of the BPP machine's random coins and it will avoid the "special" property of making the machine fail. Formalizing this argument is a little tricky when the BPP machine has an uncomputable oracle, though – the actual way Allender et al get their "random" coins is by plugging the high Kolmogorov-complexity string as the truth table of a hard function into the IW generator and then enumerating over all outputs. But the basic principle is the same.

The other step required for this proof is to show that, given an R_{K_U} oracle, it's possible to in polynomial time find some $x \in R_{K_U}$ of length $\mathsf{poly}(n)$. Buhrman et al show how to do this incrementally, showing that a string with low complexity must have a string with slightly higher complexity within a short walk on an expander graph, thus allowing a string of full complexity to be found in polynomial time. □

Now, let's show that $\Delta_0 \subseteq \bigcap_U \mathsf{P/poly}^{R_{K_U}}$. The idea of the proof is: an oracle for R_K will let you break pretty much any PRG. A PRG's output can be described by a constant-length program (the code for the PRG) and a short seed, which together constitute a short description – so your R_K oracle will tell you it isn't random. The Nisan-Wigderson construction lets us build PRGs whose security depends on non-uniform hardness assumptions – i.e. where an algorithm distinguishing the PRG from true randomness gives a circuit for some supposedly hard problem. So let's build ourselves such a PRG, and then use our magic R_K oracle to break it and solve the original problem!

Theorem 4 (Allender et al [All+06])**.** For any computable[3] language L and any universal Turing machine U, $L \subseteq \mathsf{P/poly}^{R_{K_U}}$.

Proof. Let G_L be the Nisan-Wigderson generator, instantiated with hard language L and stretching $n/2$ bit inputs to n bits. Any string in the image of G_L has Kolmogorov complexity at most $n/2+c$, where c is a constant depending on the length of a program implementing G_L. For large enough n, $n/2+c<n$, so no output of G_L will be in R_{K_U}. But a random string has good probability of being in R_{K_U}, so this means that an R_{K_U} oracle constitutes a distinguisher for the PRG. By the hypothesis of the NW generator, there is a $\mathsf{P/poly}$ circuit family that, given this distinguisher as an oracle, decides L on every input. So, $L \in \mathsf{P/poly}^{R_{K_U}}$. $\square$

5 The K-random strings will let you do $\mathsf{EXP}^{\mathsf{NP}}$

We saw a proof in Section 3 that, with non-adaptive poly-time reductions, the random strings won't let you beat PSPACE, and that even with nondeterministic adaptive poly-time reductions you can't beat $\mathsf{EXPSPACE}$. Originally, some people had the intuition that maybe $\bigcap_U \mathsf{P}^{R_{K_U}} = \mathsf{PSPACE}$ – we knew that $\mathsf{PSPACE} \subseteq \bigcap_U \mathsf{P}^{R_{K_U}}$, if you guessed that adaptive reductions aren't any better than non-adaptive ones this would mean the bounds were tight. However, this turns out to be very false[4]: in this section we will explain a proof by Hirahara showing that $\mathsf{EXP}^{\mathsf{NP}} \subseteq \bigcap_U \mathsf{P}^{R_{K_U}}$.

The basic framework of the proof is the same idea we used to show $\Delta_0 \subseteq \mathsf{P/poly}^{R_{K_U}}$: we engineer a PRG such that any distinguisher can be black-box converted into an algorithm for a hard problem, and then use our R_{K_U} oracle as a distinguisher. We can't just plug in Nisan-Wigderson and win anymore, though, because breaking NW gives a non-uniform circuit family for the hard problem, and we're looking for a uniform algorithm. Getting PRGs from uniform assumptions is a tricky game. The basic insight that let Hirahara crank this all the way up to $\mathsf{EXP}^{\mathsf{NP}}$ is that an R_{K_U} oracle is a *really, really good* distinguisher – so good that it can break even a PRG that stretches by only $O(\log n)$ bits. He gives a construction of a such a tiny-stretch PRG where a distinguisher gives us an "only slightly non-uniform" algorithm for the hard problem – that is, we can uniformly generate polynomially-many circuits such that at least one of them is guaranteed to work. Then, assuming the hard problem is $\mathsf{EXP}^{\mathsf{NP}}$-complete, he shows that we can figure out in uniform polynomial time which of these circuits to listen to, using a ***selector*** (a relaxed notion of an instance checker) for $\mathsf{EXP}^{\mathsf{NP}}$.

There's two key tools involved in this proof. The first, which will be necessary for the PRG construction, is a result about locally list-decodable codes, which we'll state without proof.

Lemma 3 (Sudan, Trevisan, Vadhan [STV01])**.** For any size N and error parameter ε, there exists an error-correcting code mapping strings of length N to strings of length $\mathsf{poly}(N, 1/\varepsilon)$, such that

- Encoding can be done in polynomial time, and
- There is a randomized decoding algorithm Dec that runs in $\mathsf{poly}(\log N, 1/\varepsilon)$ time. On input r, if r has distance within $\frac{1}{2}-\varepsilon$ of the codeword of some $x \in \{0,1\}^N$, Dec outputs a list of $\ell = \mathsf{poly}(\log N, 1/\varepsilon)$ many deterministic oracle circuits, such that with high probability over Dec's coin flips one of those ℓ circuits computes x_i on *every* input i when given oracle access to r.

That is, given a corrupted codeword, with good probability, we can produce a small list of decoding circuits such that one of them is guaranteed to compute x exactly (but we don't know which one). The other tool in the proof will be Hirahara's construction of selectors for $\mathsf{EXP}^{\mathsf{NP}}$.

[3]The original paper we're citing actually claims that this approach shows that the halting problem can be $\mathsf{P/poly}$ truth-table reduced to the Kolmogorov random strings. However, we've been unable to understand why this should be the case, and are instead just presenting this weaker version. If you, the reader, understand why this works, please let us know! The concern is that when we plug the halting problem in as the hard problem to a NW generator, it's no longer clear that R_K should be able to distinguish the output from random.

[4]Ok, ok "very false unless $\mathsf{PSPACE} = \mathsf{EXP}^{\mathsf{NP}}$". You know what I mean.

Definition 6 (Hirahara [Hir15])**.** A ***selector*** for a language L is a randomized polynomial time algorithm that, on input x and with access to two oracles A and B such that at least one is promised to decide L correctly on every input, outputs $L(x)$ with high probability.

Lemma 4 (Hirahara [Hir15])**.** There exists a selector for any $\mathsf{EXP}^{\mathsf{NP}}$-complete language.

Lemma 5 (Hirahara [Hir15])**.** If there exists a selector for a language, there exists a selector that works even when given polynomially many oracles, as opposed to just two. Additionally, the length of the largest query asked by the selector is a fixed polynomial independent of the number of oracles being decided between.

We will defer the proofs of this lemmas until after the main theorem.

Theorem 5 (Hirahara [Hir20])**.** For any U, $\mathsf{EXP}^{\mathsf{NP}} \subseteq \mathsf{P}^{R_{K_U}}$.

Proof. First, recall that we showed in Lemma 2 that $\mathsf{P}^{R_{K_U}} = \mathsf{BPP}^{R_{K_U}}$, so it suffices to give a $\mathsf{BPP}^{R_{K_U}}$ algorithm for some $\mathsf{EXP}^{\mathsf{NP}}$-complete language L. The way we will do so is by constructing (and breaking) a PRG. First, let L_n be the indicator function of L on strings of length n, and let $\widehat{L_n}$ be the function whose truth table is obtained by encoding the truth table of L_n with the locally list-decodable code of Lemma 3. We present the following PRG, which just breaks up its input into a few chunks and applies $\widehat{L_n}$ on each of them:

$$D(x_1,\ldots,x_k) = \left(x_1,\ldots,x_k,\widehat{L_n}(x_1),\ldots,\widehat{L_n}(x_k)\right).$$

Here, $x_1,\ldots,x_k \in \{0,1\}^{\widehat{n}}$, where $\widehat{n} = \mathsf{poly}(n)$ is the length of inputs to $\widehat{L_n}$, and we choose $k = 100\log\widehat{n}$. So D stretches inputs of length $100\widehat{n}\log\widehat{n}$ to $100(\widehat{n}+1)\log\widehat{n}$. This is a small stretch, but enough that nothing in the image can be in R_{K_U}, so our oracle always rejects outputs of the PRG even though accepts random strings with probability at least $1/2$. We're going to show that this distinguisher lets us in randomized poly time find a small list of circuits such that one of them computes L_n.

Now, we're going to make a hybrid argument. Define the distribution

$$H_i = (x_1,\ldots,x_k,\widehat{L_n}(x_1),\ldots,\widehat{L_n}(x_i),b_{i+1},\ldots,b_k),$$

where the x_j are chosen randomly from $\{0,1\}^{\widehat{n}}$, and the b_j are random bits. H_0 is the uniform distribution, where our distinguisher accepts with probability at least $1/2$, and H_k is the PRG's output distribution, where our distinguisher always rejects. So, if we choose a uniform random $i \in [k]$,

$$\Pr_{i\sim[k]}[\text{ distinguisher accepts } H_{i-1}\ ] - \Pr_{i\sim[k]}[\text{ distinguisher accepts } H_i\ ] \geq \frac{1}{2k}.$$

By averaging[5], we can conclude that when we randomly fix everything except x_i, with decent probability the distribution still has a decent bias. That is,

$$\Pr_{\substack{i\sim[k]\\ x_1,\ldots,x_{i-1},x_{i+1},\ldots,x_k\\ b_1,\ldots,b_k}}\left[\Pr_{x_i}[\text{ distinguisher accepts } H_{i-1}\ ] - \Pr_{x_i}[\text{ distinguisher accepts } H_i\ ] \geq \frac{1}{4k}\right] \geq \frac{1}{4k}.$$

This is very good news for us. What we're trying to do here is find a deterministic algorithm that gets $\widehat{L_n}$ correct on substantially more than half it's inputs, since then we can plug that algorithm into our local list-decoding procedure. To come up with a candidate for such an algorithm, we'll do the following:

- Choose $i \sim [k]$, $x_1,\ldots,x_{i-1},x_{i+1},\ldots,x_k \sim \{0,1\}^{\widehat{n}}$, and $b_1,\ldots b_k \sim \{0,1\}$. With probability at least $\frac{1}{4k}$, we chose "good" values for these, in the sense that

$$\Pr_{x_i}[\text{ distinguisher accepts } (x_1,\ldots,x_k,\widehat{L_n}(x_1),\ldots,\widehat{L_n}(x_{i-1}),b_i,b_{i+1}\ldots,b_k)\ ] -$$
$$\Pr_{x_i}[\text{ distinguisher accepts } (x_1,\ldots,x_k,\widehat{L_n}(x_1),\ldots,\widehat{L_n}(x_{i-1}),\widehat{L_n}(x_i),b_{i+1},\ldots,b_k)\ ] \geq \frac{1}{4k}.$$

[5]Here, we're just applying the general fact that for any function f and random variable y, $\mathbb{E}_y[f(y)] \leq \frac{1}{4k}\Pr[f(y) < \frac{1}{4k}] + \Pr[f(y) > \frac{1}{4k}]$.

- Choose random bits $a_1,\ldots,a_{i-1}$. Since we're running in poly time, we can't compute the values of $\widehat{L_n}(x_1),\ldots\widehat{L_n}(x_{i-1})$ ourself – but if we guess them at random, there's a $\frac{1}{2^{i-1}} \geq \frac{1}{2^k}$ chance that we'll guess them all correctly.
- Now, the chance that all of our guesses were "good" is at least $\frac{1}{4k} \cdot \frac{1}{2^k}$, and if we made good guesses we have

$$\Pr_{x_i}[\text{ distinguisher accepts } (x_1,\ldots,x_k,a_1,\ldots,a_{i-1},b_i,b_{i+1},\ldots,b_k)\ ] -$$
$$\Pr_{x_i}[\text{ distinguisher accepts } (x_1,\ldots,x_k,a_1,\ldots,a_{i-1},\widehat{L_n}(x_i),b_{i+1},\ldots,b_k)\ ] \geq \frac{1}{4k}.$$

  In that case, if we let our algorithm $A(x)$ be "run the distinguisher on $(x_1,\ldots,x_{i-1},x,\ldots,x_k,a_1,\ldots,a_{i-1},b_i,b_{i+1},\ldots,b_k)$, outputting b_i if it rejects and $\neg b_i$ if it accepts", we'll have $A(x) = \widehat{L_n}(x)$ on at least a $\frac{1}{2} + \frac{1}{4k}$ fraction of x.

Once we've run this randomized process, we get out some deterministic algorithm A. The process was "successful" with probability $\frac{1}{4k \cdot 2^k}$, but we can't tell yet whether it succeeded or not. Regardless, we can now run the local list-correcting code's Dec algorithm, using A to answer the queries. If the first process was successful, then A's truth table has distance less than $\frac{1}{2} - \frac{1}{4k}$ from $\widehat{L_n}$'s truth table, so with high probability over the internal coins of Dec, this decoding process will be successful, and one of the circuits it returns will be a circuit perfectly computing L_n. All in all, when we've run this whole procedure, we end up with a list of $\mathsf{poly}(n)$ many circuits, such that with probability at least $\frac{1}{8k \cdot 2^k}$ one of the circuits perfectly computes L_n. Remembering that $k = 100 \log \widehat{n}$, this probability is $\frac{1}{\mathsf{poly}(n)}$. So, by repeating this procedure $\mathsf{poly}(n)$ many times and concatenating the resulting lists together, we end up with a list of $\mathsf{poly}(n)$ many circuits, such that except with negligible failure probability the list contains a circuit computing L_n.

We now describe the $\mathsf{BPP}^{R_{K_U}}$ algorithm for deciding L. The crucial observation is that, by Lemma 4 and the fact that L is $\mathsf{EXP}^{\mathsf{NP}}$-complete, given two oracles such that at least one is guaranteed to compute L on all inputs, we can compute $L(x)$ in randomized polynomial time. In our case, we're trying to decide between $\mathsf{poly}(|x|)$ many oracles as opposed to just two – but by Lemma 5 this is also possible. The other difference is that we're going to have a list of circuits, as opposed to arbitrary oracles, and so we will only be able to make a queries of fixed length. However, by choosing L to be a paddable $\mathsf{EXP}^{\mathsf{NP}}$-complete language, we can make do with circuits. Letting $N = \mathsf{poly}(|x|)$ be the length of the largest query the selector would make on input x, we'll apply the procedure outlined above to get a list of $\mathsf{poly}(N)$ many circuits such that one of them computes L_N, and then we'll use our selector with these circuits as the oracles (padding the queries as required so that they're all of length N) to determine $L(x)$. Both the circuit listing process and the selector have negligible error probability, so this algorithm can be implemented in $\mathsf{BPP}^{R_{K_U}}$. □

We'll now outline the proofs we skipped about selectors.

Proof of Lemma 4. It is easy to show that if one language has a selector, all poly-time equivalent languages have selectors, so it suffices to show that a single specific $\mathsf{EXP}^{\mathsf{NP}}$ language has one. We will consider the $\mathsf{EXP}^{\mathsf{NP}}$-complete language Succinct Lexicographically Maximum 3SAT – this language consists of all pairs (φ, i) such that φ is a succinctly-encoded 3SAT instance, and i is an index such that the ith bit of the lexicographically last satisfying assignment to φ is 1. Suppose we have two oracles, A and B, such that either A or B is an accurate Succinct Lexicographically Maximum 3SAT oracle, and we want to compute the ith bit of the lexicographically last satisfying assignment to some succinct formula φ. The key perspective is that, because A and B are (obstensible) oracles for an $\mathsf{EXP}^{\mathsf{NP}}$ complete problem, by performing poly-time reductions on our queries we can ask them to answer arbitrary $\mathsf{EXP}^{\mathsf{NP}}$ questions, not just give us bits from the largest assignment to φ.

In particular, what we will actually ask each of A and B to do is to compute specific bits of the ***multilinear extension*** of the lexicographically maximum satisfying assignment to φ. Thinking of the original assignment as a function $\{0,1\}^n \to \{0,1\}$ returning the bit of the assignment at the index given by the input, the multilinear extension over a large field $\mathbb{F}$ is the unique multilinear function $\mathbb{F}^n \to \mathbb{F}$ that agrees with that function on $\{0,1\}^n$-valued inputs. Computing an index into the multilinear extension of the lexicographically maximum satisfying assignment can be done in exponential time with an NP oracle, so the honest oracle will do so correctly. Thus, we can think of ourselves as being given two exponentially-long strings representing two purported arithmetizations of the lexicographically maximum satisfying assignment, and trying to randomly verify which is correct. Our first step will be to run a multilinearity test on each string – if this test fails, we can safely assume the corresponding oracle was faulty and just trust the other one, but if the test succeeds we know the string is close to some multilinear function. In order to verify that the multilinear functions the strings are close to arithmetized satisfying assignments, we follow exactly the proof of $\mathsf{MIP} = \mathsf{NEXP}$, arithmetizing the formula and performing sum-checks. If one of the two strings isn't close to the multilinear extension of any satisfying assignment, this will with high probability detect that error (for details, see [BFL90]), and we are safe to trust the other oracle. The place where new ideas are needed is when both strings represent satisfying assignments, and we have to determine which is larger. Note that if they both represent the same satisfying assignment we've already won – they will both agree on the index we care about, so we can safely report that value.

To determine which of two assignments is larger, it suffices to identify the first index on which they disagree. To do so, we first recall that by self-correction of the Reed-Muller code, given access to a string close to the multilinear extension, we can output any specific bit of the multilinear extension with good probability. This is important because we'll need to make queries to specific locations, and it could be the case that the string is corrupted at exactly those indices – we'll use the self-correction to get good success probability no matter what index we're looking at. So, we can just consider ourselves as having access to the true multilinear extensions of two assignments. The next important thing to note is that, since the multilinear extension is unique, checking whether the two strings represent the same satisfying assignment is equivalent to checking if they're close to the same multilinear function. So, we can describe the following binary-search procedure to find the first index $z \in \{0,1\}^n$ where the assignments differ:

- Choose random values $r_2,...,r_n$ from $\mathbb{F}$. Check the entries at location $(0,r_2,...,r_n)$ in both strings. If they match, set $z_1 = 1$; otherwise, set $z_1 = 0$.
- Choose new random values $r_3,...,r_n \leftarrow \mathbb{F}$, and query the strings at location $(z_1,0,r_3...,r_n)$. Set $z_2 = 1$ if the results agree, and $z_2 = 0$ if the results disagree.
- Continue until all indices of z are set.

Once we've chosen the first j bits of z, we can think of the two polynomials as having been restricted, and the choice of the r_i as querying these restricted polynomials on random inputs. As long as $|\mathbb{F}|$ is much larger than n, the Schwartz-Zippel lemma guarantees that unless the two restricted polynomials are the same, with high probability we will determine that they are different. If, for some fixed values of $z_1,...,z_{j-1}$, the multilinear polynomials obtained from each string by fixing the first j bits of the input to $(z_1,...,z_{j-1},0)$ are the same, this means that the two assignments are the same at every index starting with prefix $(z_1,...,z_{j-1},0)$ – so, if they differ on any index starting with prefix $(z_1,...,z_{j-1})$, that index must begin $(z_1,...,z_{j-1},1)$. On the other hand, if the two restricted polynomials aren't the same, the lexicographically first assignment where they differ must begin $(z_1,...,z_{j-1},0)$. So, if the two assignments differ on any index, this process will with high probability find the first such index, allowing us to determine which oracle has the larger satisfying assignment (and, consequently, is the correct oracle). □

Proof of Lemma 5. We have polynomially many oracles for L, and want to decide $L(x)$, given the fact that we have a selector that can decide between two oracles. First, we query every oracle on x, and divide them into two teams depending on whether they answered `YES` or `NO`. Then, choose an arbitrary oracle from team `YES` and an arbitrary oracle from team `NO`, and face them off against each other by feeding them into the pairwise selector. If the pairwise selector outputs `YES`, we know that the `NO` oracle was faulty, and can discard it – similarly, if the pairwise selector says `NO`, we can discard the `YES` oracle. We repeat this process until only oracles from one of the two teams survive, and output the corresponding value. Since the true oracle will never lose a pairwise selector match, this guarantees that the correct team wins. Note that the length of the largest question asked in this procedure is the same as the length of the largest question needed by the pairwise selector, since all we do is run that selector multiple times. □

6 What about R_C?

The results we've stated thus far have all been about the prefix-free complexity R_K. We noted good reason for this – the prefix-free complexity is in several ways a nice object of study than the plain Kolmogorov complexity.

But the plain complexity has its advantages, too. Could we have shown similar results in that case, or is there a fundamental difference?

The good news is that the $\mathsf{EXP}^{\mathsf{NP}}$ inclusion of Section 5 still holds. Remember that, in that case, we used the R_K oracle to break a PRG with seed $n-\Theta(\log n)$ – in order to do that, we only needed the R_K oracle to be accurate up to $O(\log(n))$ error. Since the plain complexity and prefix-free complexities never differ by more than $O(\log n)$, this means either will suffice.

The bad news is that nobody knows how to salvage the $\mathsf{EXPSPACE}$ containment of Section 3. It was very important to the construction that we could build a universal machine by finding a prefix-free entropy function, and Lemma 1 just isn't true for R_C. The proof of computability of $\bigcap_U \mathsf{P}^{R_{K_U}}$ is based on a similar approach, so we don't even know whether $\bigcap_U \mathsf{P}^{R_{C_U}}$ contains the halting problem. We also don't know whether $\bigcup_U \mathsf{P}^{R_{C_U}}$ contains the halting problem! Last year, Allender included the following in a list of challenges:

Problem 1 (Allender, \$1000 bounty [All23])**.** Find either a universal machine U such that you can prove $\mathsf{Halting} \in \mathsf{P}^{R_{C_U}}$, or such that you can prove $\mathsf{Halting} \notin \mathsf{P}^{R_{C_U}}$.

It is known that there exists a universal machine such that $\mathsf{Halting} \in \mathsf{EEXP}^{R_{C_U}}$, even non-adaptively, but it's not clear whether this can be done faster. And it is known that there's no time bound that suffices for all U under ***disjunctive*** truth-table reductions – i.e. reductions where the output is just the OR over the responses to all the queries.

Definition 7. A problem A is disjunctive truth-table reducible to B, denoted $A \leq_{\mathrm{dtt}} B$, if there exists a computable function $F:\{0,1\}^* \to \mathcal{P}(\{0,1\}^*)$ such that

$$A(x) = \bigvee_{y \in F(x)} B(y).$$

Theorem 6 (Kummer [Kum96])**.** For every U, there is some computable time-bound $t:\mathbb{N}\to\mathbb{N}$ such that $\mathsf{Halting} \leq_{\mathrm{dtt}}^{t} R_{C_U}$.

The proof of Theorem 6 involves constructing a sequence of $F_i:\{0,1\}^* \to \mathcal{P}(\{0,1\}^*)$ with the guarantee that, for any U, there is some i such that F_i computes a disjunctive truth-table reduction from the halting problem. However, the proof gives no explicit computable bound on how large that F_i can be. Allender, Friedman, and Kouky showed that, at least for some U, this reduction can be done in doubly-exponential time.

Theorem 7 (Allender, Friedman, Kouky [ABK06])**.** There exists a U such that $\overline{\mathsf{Halting}} \leq_{\mathrm{dtt}}^{\mathrm{eexp}} R_{C_U}$.

Proof. Let U be a universal machine such that $U(x) \leq |x|+1$ for all x. The fact that such a U exists is easy: given any universal machine U', we can let

$$U(bx) = \begin{cases} U(x) \text{ if } b=0 \\ x \text{ if } b=1 \end{cases}.$$

Now, we're going to construct a new machine U_{helpful} to help us solve the halting problem. We'll think of U_{helpful}'s input as being split into a 5-bit "tag" a, a decision bit b, and the main body x. U_{helpful} behaves as follows:

Algorithm 1 $U_{\text{helpful}}(abx)$

```
1: if $b=0$ then
2:     $y \leftarrow U(x)$
3:     Return $ay$
4: if $b=1$ then
5:     $y \leftarrow U(ax)$
6:     Treat $|y|$ as a Turing machine and simulate it
7:     if $|y|$ halts then
8:         Return $00000y$
```

It's easy to see that U_{helpful} is universal, since fixing the first bit to 0 just gives U. The important property of U_{helpful} is that, if $|y|$ doesn't halt, then $C_{U_{\text{helpful}}}(00000y) = C_{U_{\text{helpful}}}(11111y)$, but if $|y|$ does halt then $C_{U_{\text{helpful}}}(00000y) = C_{U_{\text{helpful}}}(11111y) - 5$. So, if we as an algorithm are given x and want to decide $\mathsf{Halting}(x)$, we can check for all y, $|y| = x$, whether $00000y \in R_{C_{U_{\text{helpful}}}}$. If $\mathsf{Halting}(x)$, none of these strings can have full complexity, but otherwise one of them must, so this gives a disjunctive truth-table reduction.

If we had an oracle for $\text{ov}(C_{U_{\text{helpful}}})$ – that is, an oracle for $C_{U_{\text{helpful}}}$ as a function – we could easily use this approach to decide the halting problem in poly time (just have U'_{helpful} simulate Turing machine y instead of $|y|$, and then have the algorithm compare $C_{U'_{\text{helpful}}}(00000y)$ against $C_{U'_{\text{helpful}}}(11111y)$). With an oracle for $R_{C_{U_{\text{helpful}}}}$, though, it's not clear whether this is possible (doing so would solve Problem 1 and win you exciting cash prizes). $\square$

However, if you swap the quantifiers in Theorem 6 it's no longer true, meaning that for instance $\bigcap_U \{A\colon A \leq^{\mathsf{P}}_{\text{dtt}} R_{C_U}\} = \mathsf{P}$.

Theorem 8 (Allender, Buhrman, Kouky [ABK06])**.** For every computable time-bound $t : \mathbb{N} \to \mathbb{N}$, there exists U such that $\mathsf{Halting} \not\leq^{\text{t}}_{\text{dtt}} R_{C_U}$.

The proof of this theorem actually shows the stronger statement that $\bigcap_U \{A\colon A \leq^{\text{t}}_{\text{dtt}} R_{C_U}\} \subseteq \mathsf{TIME}[t^3]$ – so when you factor out the universal machine, disjunctive truth table reductions to R_{C_U} really aren't buying you anything.

It's not especially clear what these results should suggest to us about $\bigcap_U \mathsf{P}^{R_{C_U}}$. Are these statements about the power/limitations of R_{C_U}, or just about the power/limitations of disjunctive truth-table reductions? A result about plain complexity by Hirahara and Kawamura seems to provide some more intuition, suggesting that maybe we should expect upper bounds like those in Section 3 to hold for the plain complexity. We give the following definition:

Definition 8. A reduction is ***α-honest*** for some function $\alpha : \mathbb{N} \to \mathbb{N}$ if, on inputs of length n, it makes no oracle queries of length less than $\alpha(n)$. We write $A \leq_{\text{tt},\alpha} B$ to denote existence of an α-honest truth table reduction from A to B. If a reduction is α-honest for some polynomial α, we call it ***polynomially honest***, and if it is α-honest for some super-constant α we call it ***weakly honest***.

Hirahara and Kawamura showed that, if you restrict to weakly honest reductions, the results of Section 3 extend to plain complexity.

Theorem 9 (Hirahara, Kawamura [HK18])**.** For any unbounded monotonic function $\alpha : \mathbb{N} \to \mathbb{N}$,

$$\bigcap_U \{A : A \leq^{\mathsf{P}}_{\text{tt},\alpha} R_{C_U}\} \subseteq \mathsf{PSPACE}$$

The proof structure is very similar to the proofs in the prefix-free case. The honesty condition is important to the reduction to ensure that, for any given reduction, a specific x can only be relevant to a finite number of the games. However, weak honesty seems like perhaps a rather weak property for a reduction to satisfy, so this gives some evidence that the plain complexity is not such a wildly different world from the prefix-free.

7 What good is an approximation?

As we mentioned in the previous section, the only thing we needed to get $\mathsf{EXP}^{\mathsf{NP}}$ was an oracle computing an additive $\log(n)$ approximation of R_{K_U} – that is, an oracle that accepts strings with Kolmogorov complexity at least n, and rejects strings with Kolmogorov complexity less than $n - O(\log n)$, but is allowed to do whatever it wants on strings with complexity in-between. The idea of reducing to an arbitrary *approximate* R_K oracle is a rather attractive one, since as long as the error window for the approximation is larger than order $\log(n)$, this masks differences not just over which specific universal machine U we define R_K based on, but also whether we're using the plain or the prefix-free complexity [6].

[6] A note: it is somewhat easier to think, in this setting, of the threshold for R_K as being set at $n/2$ as opposed to near n – that is $R_K = \{x\colon K(x) \geq |x|/2\}$. The exact place you put this threshold usually doesn't matter, since you can just pad with random bits of your own to even things out, but it might be a little awkward to think about what error bounds are allowed to look like at the upper ends.

In 2022, Saks and Santhanam showed a sharp tradeoff in how useful such an oracle is [SS22]. We know from Section 5 that, with an oracle for any $\log(n)$ additive approximation to R_{K_U}, a deterministic poly-time algorithm can do all of $\mathsf{EXP}^{\mathsf{NP}}$. Saks and Santhanam show, however, that if the error window is any larger, the oracle is useless.

Theorem 10. If a language L is solvable in poly-time by an algorithm with each oracle for approximations $\widetilde{R}_K(x) = R_K(x)$ of some fixed additive error margin $e = \omega(\log(n))$, then $L \in \mathsf{P}$.

They also showed an upper bound of $\mathsf{AM} \cap \mathsf{coAM}$ on the power of non-adaptive randomized reductions, and an upper bound of SZK (Statistical Zero Knowledge) on the power of randomized m-reductions, under the condition that the reductions are polynomially honest (see Definition 8). The latter of these is somewhat surprising – SZK is a notion coming from cryptographic proofs, what is it doing here? However, a subsequent paper by Allender, Hirahara, and Tirumala showed that the appearance of SZK as an upper bound was no accident. First, some definitions.

Definition 9. The class SZK consists of promise problems with ***statistically hiding*** interactive proofs. That is, where a probablistic verifier interacting with an unbounded prover can decide the language with bounded error, but in such a way that on YES instances, without the help of the prover, the verifier could sample from a distribution of interaction transcripts that's statistically close to what this protocol gives. An example is graph non-isomorphism: the verifier sends a random permutation of a random one of the two graphs, and expects the prover to correctly identify which one it was.

Definition 10. The class NISZK is the class of promise problems with ***non-interactive*** statistical zero-knowledge proofs. That is, the prover and verifier both see some random coins flipped by a (trusted) external source, and then the prover sends a single message to the verifier, which should convince the verifier of the answer with bounded error probability. The zero-knowledge condition required is that the verifier should be able to reproduce on their own something statistically close to the joint distribution of prover messages and external random strings.

These classes can also be natural described in terms of their complete problems:

Definition 11. The promise problem ***Statistical Distance*** (SD) is defined over pairs of circuits (A,B), where
- if the statistical distance between the output distributions on A and B is larger than $2/3$, (A,B) is a YES instance
- if the statistical distance between the output distributions on A and B is smaller than $1/3$, (A,B) is a NO instance.

Sahai and Vadhan showed that SD is complete for SZK [SV00].

Definition 12. The promise problem ***Entropy Approximation*** (EA) is defined over circuits X and values k, where
- if the output distribution of X has entropy $H(X) > k+1$, (X,k) is a YES instance
- if the output distribution of X has entropy $H(X) < k-1$, (X,k) is a NO instance.

Goldreich, Sahai and Vadhan showed that EA is complete for NISZK [GSV99].

Using the completeness of these problems, Allender, Hirahara, and Tirumala were able to show *exact* characterizations of both SZK and NISZK in terms of reductions to $\widetilde{R}_K$. This is rather exciting – it shows that, at least if we put enough adjectives in front of everything, the answer to the question of "what can be efficiently reduced to the random strings" coincides with classes people have actually studied and care about. The result is as follows:

Theorem 11 (Allender, Hirahara, Tirumala [AHT23])**.** Taking the intersection over all oracles $\widetilde{R}_K$ that are within an additive error of e from R_K on every input, for some fixed $\omega(\log(n)) \leq e(n) \leq n^{o(1)}$, we have that
- $\{A : A \leq_{\text{hm}}^{\mathsf{RP}} \widetilde{R}_K\} = \mathsf{NISZK}$, where $\leq_{\text{hm}}$ denotes an m-reduction that's polynomially honest, in the sense of Definition 8[7].
- $\{A : A \leq_{\text{hbf}}^{\mathsf{RP}} \widetilde{R}_K\} = \mathsf{SZK}$, where $\leq_{\text{hbf}}$ denotes an honest ***boolean formula*** reduction – i.e. an honest reduction computable by uniform boolean formulas.

Proof sketch. We will sketch the first statement; the second follows a fairly similar outline. The first step, which we'll omit, is to show that the following variant of EA is also NISZK-complete, which is done by reduction from EA:

Definition 13. The language EA' is defined over circuits X, where (identifying the circuit with its output distribution on uniform input)

[7]We define an RP m-reduction to be one which maps all NO instances to NO instances, and YES instances to YES instances with probability at least $\frac{1}{n^{\omega(1)}}$. Note that both of these statements, and the proof we outline, also hold for BPP reductions, i.e. where NO instances just have to map to NO instances with probability at least $\frac{1}{n^{\omega(1)}}$.

- if $H(X) > n-2$, X is a YES instance
- if $|\mathrm{Supp}(X)| \leq 2^{n-n^{0.99}}$, X is a NO instance (where $\mathrm{Supp}(X)$ denotes the support of the distribution).

Note that these conditions are indeed mutually exclusive, since $H(X) \leq \log(|\mathrm{Supp}(X)|)$.

First, we show a randomized honest m-reduction from EA' to $\widetilde{R}_K$. We're given some circuit C as our input, and want to determine which of the conditions holds. To do so, we'll just take some large polynomial $t = 100n^{100}$ many random samples $x_1,...,x_t \sim X$, and query the $\widetilde{R}_K$ oracle on $(x_1,...,x_t)$. If $H(X) > n-2$, then it's possible to show that the typical set is large enough that except with negligible probability this will produce a string of Kolmogorov complexity at least $t(n-3)$. However, if the support of X is smaller than $2^{n-n^{0.99}}$, it is impossible for this to produce a string of Kolmogorov complexity more than $t(n-4)$ – every output will have a shorter description in terms of X. By appropriately padding with randomness, we can use our $\widetilde{R}_K$ oracle to distinguish these two cases.

For the other direction, we start with some RP circuit C that computes an honest m-reduction to $\widetilde{R}_K$, and want to show that the language the reduction computes is in NISZK. That is, C takes as input some (x,r), where x is a problem instance and r are random coins, and outputs a string $y \in \{0,1\}^m$ intended as an oracle query to $\widetilde{R}_K$ [8]. Let C_x denote the restricted circuit computing $C(x,\cdot)$ – our goal will be to show that, if C_x does represent a valid RP reduction to $\widetilde{R}_K$, we can determine the output of the reduction by computing $\mathsf{EA}(C_x)$. Recall that, for C_x to be a correct reduction, if we take y drawn from C_x's output distribution, we must have either

$$\Pr[K(y) > m/2 - e(m)] \leq \frac{1}{n^{\omega(1)}}$$

or

$$\Pr[K(y) < m/2 + e(m)] \leq \frac{1}{n^{\omega(1)}},$$

and our goal is to tell which is which.

Claim 1. If $\Pr[K(y) > m/2 - e(m)] \leq \frac{1}{n^{\omega(1)}}$, then $H(C_x) \leq m/2 - e(m) - 1$. That is, small expected Kolmogorov complexity means small entropy.

Claim 2. If $\Pr[K(y) < m/2 + e(m)] \leq \frac{1}{n^{\omega(1)}}$, then $H(C_x) \geq m/2 - e(m) + 1$. That is, big expected Kolmogorov complexity means big entropy.

To prove these two claims, we go by contradiction – let x^* be the lexicographically first x violating one of the two claims. Note that this ensures $K(x^*) \leq O(\log(n))$, since C has a uniform description, and the property of violating one of the claims is computable[9].

Case 2.1. x^* violates Claim 1.

For x^* to violate Claim 1, we must have that $\Pr[K(y) > m/2 - e(m)] \leq \frac{1}{n^{\omega(1)}}$, and also $H(C_{x^*}) > m/2 - e(m) - 1$. But the number of strings y with $K(y) \leq m/2 - e(m)$ is only at most $2^{m/2 - e(m)}$ by counting programs, which would let us upper bound $H(C_{x*}) \leq m \cdot \frac{1}{n^{\omega(1)}} + \log(2^{m/2-e(m)}) \cdot \left(1 - \frac{1}{n^{\omega(1)}}\right) \leq m/2 - e(m) - 1$.

Case 2.2. x^* violates Claim 2.

For x^* to violate Claim 2, we must have that $\Pr[K(y) < m/2 + e(m)] \leq \frac{1}{n^{\omega(1)}}$, and also $H(C_{x^*}) < m/2 - e(m) + 1$. But now, recall that x^* has a short description. This means that the $2^{m/2+e(m)-O(\log n)}$ highest probability elements of C_{x^*}'s output distribution must have Kolmogorov complexity at most $m/2 + e(m)$, since we can specify them by giving x^* and their ranking in terms of highest probability. But now, we can lower bound $H(C_{x^*}) \geq 1 \cdot \frac{1}{n^{\omega(1)}} + \left(m/2 + e(m) - O(\log n) + \log(n^{\omega(1)})\right)\left(1 - \frac{1}{n^{\omega(1)}}\right) \geq m/2 + e(m) + 1$.

Having established Claim 2 and Claim 1, we now have a poly-time m-reduction from determining the value of this reduction to $\widetilde{R}_K$ to EA. Since NISZK is closed under poly-time m-reductions, this completes the proof. □

[8] We can assume without loss of generality that m is well-defined here – that is, that all queries are the same length – by appropriate padding.

[9] This is using the fact that the language being decided is computable – we can't compute the response from the R_K oracle directly, but we can determine what it was supposed to be.

We note, finally one interesting application of this line of thought. Allender, Hirahara, and Tirumala showed that, in addition to the results we mentioned above, $\mathsf{NISZK_L}$ (the log-space bounded version of NISZK) can be characterized by ***projection*** reductions to $\widetilde{R}_K$ – that is, reductions computable by a nonuniform circuit family containing only NOTs, wires, and constants. One consequence of this, since the minimum time-bounded Kolmogorov problem MKTP lies in NP, is that, if $\mathsf{NISZK_L} = \mathsf{NP}$, there must exist some non-uniform projection that takes strings of large KT complexity to strings of low K complexity, and strings of small KT complexity to strings of large K complexity [10]. If that sounds like an implausible thing to exist, then maybe ruling it out is a potential approach to separating $\mathsf{NISZK_L}$ from NP (which would, among other things, separate NL and NP). This is an ambitious goal, but as far as the authors of this survey are aware there is no known barrier yet making it impossible[11].

8 Conclusion

The power of reductions to the random strings is an interesting question, albeit perhaps a strange one. Some of the most exciting open directions along these lines are:

- To what extent can "honesty" conditions be removed, either in the approximation results of Section 7, or in the upper bound on time-bounded reductions to R_{C_U} of Theorem 9? Note that if the latter could be removed, this would mean all our known bounds on $\bigcap_U \mathsf{P}^{R_{K_U}}$ also apply to $\bigcap_U \mathsf{P}^{R_{C_U}}$. It is, however, also possible that honesty is fundamentally necessary for these results – if so, it would be interesting to explore how and why these classes change when it's relaxed.
- Can we improve either bound on $\bigcap_U \mathsf{P}^{R_{K_U}}$? There's potentially a good amount of room between $\mathsf{EXP^{NP}}$ and $\mathsf{EXPSPACE}$. Both bounds are at pretty natural limits of known techniques, however. The $\mathsf{EXP^{NP}}$ barrier seems perhaps more plausible to surpass – the proof of $\mathsf{EXPSPACE}$ came from a very natural strategy of reduction involving playing games; it's quite unclear what would have to happen to show a smaller bound.
- Is Allender's magic mirror (alluded to at the end of Section 7) likely to exist? If not, are there any barriers towards proving its nonexistence, or is this in fact a plausible proof approach towards showing $\mathsf{NL} \neq \mathsf{NP}$? If we can't either rule it out or show it exists, can we get any interesting implications from its existence?
- Are there other randomized complexity classes with natural descriptions in terms of reductions to the random strings? The fact that NISZK and SZK have such characterizations is very interesting. There was once hope that BPP was characterized by poly-time truth-table reductions to R_K, but recent results have made this much less plausible – maybe there is still a different natural characterization.

References

[ABK06] Eric Allender, Harry Buhrman, and Michal Koucký. "What can be efficiently reduced to the Kolmogorov-random strings?" In: *Annals of Pure and Applied Logic* 138.1 (2006), pp. 2–19. ISSN: 0168-0072. DOI: https://doi.org/10.1016/j.apal.2005.06.003. URL: https://www.sciencedirect.com/science/article/pii/S0168007205000849.

[AFG13] Eric Allender, Luke Friedman, and William Gasarch. "Limits on the computational power of random strings". In: *Information and Computation* 222 (2013). 38th International Colloquium on Automata, Languages and Programming (ICALP 2011), pp. 80–92. ISSN: 0890-5401. DOI: https://doi.org/10.1016/j.ic.2011.09.008. URL: https://www.sciencedirect.com/science/article/pii/S0890540112001472.

[AHT23] Eric Allender, Shuichi Hirahara, and Harsha Tirumala. "Kolmogorov Complexity Characterizes Statistical Zero Knowledge". In: *14th Innovations in Theoretical Computer Science Conference, ITCS 2023, January 10-13, 2023, MIT, Cambridge, Massachusetts, USA*. Ed. by Yael Tauman Kalai. Vol. 251. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023, 3:1–3:19. DOI: 10.4230/LIPICS.ITCS.2023.3. URL: https://doi.org/10.4230/LIPIcs.ITCS.2023.3.

[10]The first author of this survey would like such a projection to be called "Allender's magic mirror". However it's not clear this name will catch on.

[11]Of course, this is probably largely because it has not had a huge amount of thought devoted to it as an approach. But hey, worth a shot.

[All+06] Eric Allender et al. “Power from Random Strings”. In: *SIAM Journal on Computing* 35.6 (2006), pp. 1467–1493. DOI: 10.1137/050628994. eprint: https://doi.org/10.1137/050628994. URL: https://doi.org/10.1137/050628994.

[All23] Eric Allender. “Guest Column: Parting Thoughts and Parting Shots (Read On for Details on How to Win Valuable Prizes!” In: *SIGACT News* 54.1 (Mar. 2023), pp. 63–81. ISSN: 0163-5700. DOI: 10.1145/3586165.3586175. URL: https://doi.org/10.1145/3586165.3586175.

[BFL90] L. Babai, L. Fortnow, and C. Lund. “Nondeterministic exponential time has two-prover interactive protocols”. In: *Proceedings [1990] 31st Annual Symposium on Foundations of Computer Science*. 1990, 16–25 vol.1. DOI: 10.1109/FSCS.1990.89520.

[Buh+05] Harry Buhrman et al. “Increasing Kolmogorov Complexity”. In: *STACS 2005*. Ed. by Volker Diekert and Bruno Durand. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 412–421. ISBN: 978-3-540-31856-9.

[Cai+14] Mingzhong Cai et al. “Random strings and tt-degrees of Turing complete C.E. sets”. In: *Logical Methods in Computer Science [electronic only]* 10 (Aug. 2014). DOI: 10.2168/LMCS-10(3:15)2014.

[GSV99] Oded Goldreich, Amit Sahai, and Salil Vadhan. “Can Statistical Zero Knowledge be made Non-Interactive? or On the Relationship of NISZK”. In: vol. 6. Dec. 1999, pp. 791–791. ISBN: 978-3-540-66347-8. DOI: 10.1007/3-540-48405-1_30.

[Hir15] Shuichi Hirahara. “Identifying an Honest $\mathsf{EXP}^{\mathsf{NP}}$ Oracle Among Many”. In: *30th Conference on Computational Complexity (CCC 2015)*. Ed. by David Zuckerman. Vol. 33. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2015, pp. 244–263. ISBN: 978-3-939897-81-1. DOI: 10.4230/LIPIcs.CCC.2015.244. URL: https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.CCC.2015.244.

[Hir20] Shuichi Hirahara. “Unexpected hardness results for Kolmogorov complexity under uniform reductions”. In: *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing*. STOC 2020. Chicago, IL, USA: Association for Computing Machinery, 2020, pp. 1038–1051. ISBN: 9781450369794. DOI: 10.1145/3357713.3384251. URL: https://doi.org/10.1145/3357713.3384251.

[HK18] Shuichi Hirahara and Akitoshi Kawamura. “On characterizations of randomized computation using plain Kolmogorov complexity”. English. In: *Computability* 7.1 (2018). Publisher Copyright: © 2018 - IOS Press and the authors. All rights reserved., pp. 45–56. ISSN: 2211-3568. DOI: 10.3233/COM-170075.

[Kum96] Martin Kummer. “On the complexity of random strings”. In: *STACS 96*. Ed. by Claude Puech and Rüdiger Reischuk. Berlin, Heidelberg: Springer Berlin Heidelberg, 1996, pp. 25–36. ISBN: 978-3-540-49723-3.

[MP02] Andrei A. Muchnik and Semen Ye. Positselsky. “Kolmogorov entropy in the context of computability theory”. In: *Theor. Comput. Sci.* 271.1-2 (2002), pp. 15–35. DOI: 10.1016/S0304-3975(01)00028-7. URL: https://doi.org/10.1016/S0304-3975(01)00028-7.

[SS22] Michael Saks and Rahul Santhanam. “On randomized reductions to the random strings”. In: *Proceedings of the 37th Computational Complexity Conference*. CCC ’22. Philadelphia, Pennsylvania: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2022. ISBN: 9783959772419. DOI: 10.4230/LIPIcs.CCC.2022.29. URL: https://doi.org/10.4230/LIPIcs.CCC.2022.29.

[STV01] Madhu Sudan, Luca Trevisan, and Salil Vadhan. “Pseudorandom Generators without the XOR Lemma”. In: *Journal of Computer and System Sciences* 62.2 (2001), pp. 236–266. ISSN: 0022-0000. DOI: https://doi.org/10.1006/jcss.2000.1730. URL: https://www.sciencedirect.com/science/article/pii/S0022000000917306.

[SV00] Amit Sahai and Salil Vadhan. *A Complete Problem for Statistical Zero Knowledge*. Cryptology ePrint Archive, Paper 2000/056. https://eprint.iacr.org/2000/056. 2000. URL: https://eprint.iacr.org/2000/056.