Electronic Colloquium on Computational Complexity, Report No. 34 (2026)

Limit on the computational power of C-random strings

Alexey Milovanov

February 14, 2026

Abstract

We construct a universal decompressor U for plain Kolmogorov complexity C_U such that the Halting Problem cannot be decided by any polynomial-time oracle machine with access to the set of random strings $R_{\mathrm{C}_U} = \{x : \mathrm{C}_U(x) \geq |x|\}$. This result resolves a problem posed by Eric Allender in [1] regarding the computational power of Kolmogorov complexity-based oracles.

Contents

ISSN 1433-8092

1 Introduction

"Informally", the Kolmogorov complexity of a string x is defined as the minimal length of a program that outputs x on the empty input. This definition requires refinement; there are various types of complexity (plain, prefix, and others), but even once the type is fixed, there remains a dependency on the choice of the programming language or decompressor. We provide all the definitions needed for this paper in the next section; we also refer the reader to [8, 12] for formal definitions and the main properties of Kolmogorov complexity.

The concept of Kolmogorov complexity allows us to define the notion of an individual random string. This can be done as follows: a string x is called random if its Kolmogorov complexity $\gtrsim |x|$, where $|x|$ is the length of x. However, the formal definition requires refinement: in addition to the aforementioned subtleties with the definition of complexity, it is necessary to formally define what $\gtrsim$ means. Assume that all technical parameters are fixed. Let R denote the set of all random strings. A natural question is how powerful the oracle R is? For example, what languages belong to P^R?

This and similar questions were investigated in [2, 3, 6].

In these papers, some lower bounds for $\mathrm{P}^R, \mathrm{P/poly}^R$ and other complexity classes were established. These results are robust in the following sense: they are valid for all reasonable definitions of R; in particular, it does not matter what type of Kolmogorov complexity we consider.

The situation is different for upper bounds. The results in [4, 5, 9] establish limits on the computational power of R specifically for prefix complexity.

For example, the results in [4, 5] together with the lower bound obtained in [6] show that

$$\mathrm{EXP}^{\mathrm{NP}} \subseteq \bigcap_U \mathrm{P}^{R_{\mathrm{K}_U}} \subseteq \mathrm{EXPSPACE}.$$

Here, the intersection is taken over all universal prefix-free decompressors U. While there is a significant gap between the lower and upper bounds for prefix complexity, it is known that the intersection does not contain any undecidable language. For plain complexity, even this remains unknown.

This paper is a step towards clarifying the situation for plain complexity.

Let $\mathrm{C}_U(x)$ denote the plain Kolmogorov complexity of a string x with respect to a universal decompressor U. Instead of working directly with the set of random strings, we consider the stronger cumulative complexity oracle:

$$F_{\mathrm{C}_U} = \{\langle x, k\rangle \in \{0,1\}^* \times \mathbb{N} : \mathrm{C}_U(x) \le k\}.$$

Let H denote the Halting Problem:

$$H := \{x \mid \text{program } x \text{ halts on the empty input}\}.$$

Our main result is the following theorem.

Theorem 1.1. *There exists a universal decompressor U such that $H \notin \mathrm{P}^{F_{\mathrm{C}_U}}$.*

Corollary 1.2. *Let $R_{\mathrm{C}_U} = \{x \in \{0,1\}^* : \mathrm{C}_U(x) \ge |x|\}$ be the set of random strings. Then $H \notin \mathrm{P}^{R_{\mathrm{C}_U}}$.*

Related works

In [10], it was shown that for *some* universal U the Halting Problem belongs to $\mathrm{P}^{R_{\mathrm{C}_U}}$. Together with our main result, this implies that the computational power of $\mathrm{P}^{R_{\mathrm{C}_U}}$ significantly depends on the choice of the universal machine U.

In [7], Kummer showed that for each universal machine U, there is a time-bounded disjunctive truth-table reduction from H to R_U. That is, there is a computable function that takes an input x and produces a list of strings, with the property that $x \in H$ if and only if at least one of the strings is in R_U. Therefore, the polynomial restriction in Theorem 1.1 is significant.

Roadmap

In the next section, we recall the definition of plain Kolmogorov complexity and its basic properties.

In Sections 3 and 4, we present the framework for the proof of the main result. In Section 5, we formulate the key lemma that we prove in Section 7. Finally, in Subsection 6.1, we give an outline of the proof of the main result, and in the rest of Section 6, we give the full proof.

2 Definitions and Basic Properties of Kolmogorov Complexity

We recall the definitions of plain conditional and unconditional Kolmogorov complexity.

Let U be an algorithm whose inputs and outputs are binary strings. We call such an algorithm a *decompressor* and define the (unconditional) complexity $\mathrm{C}_U(x)$ of a binary string x with respect to U as follows:

$$\mathrm{C}_U(x) := \min\{|y| : U(y) = x\}.$$

We call any y such that $U(y) = x$ a *description* (or *U-description*) of x. Thus, the complexity of x is defined as the length of the shortest description of x.

A decompressor U is called *universal* if for every decompressor U' there exists a constant M such that for every string x,

$$\mathrm{C}_U(x) \leq \mathrm{C}_{U'}(x) + M.$$

Now let us recall the definition of *conditional* Kolmogorov complexity. Let $D(p, y)$ be a computable partial function (which we will also call a decompressor) of two string arguments. We may think of D as an interpreter for some programming language, where the first argument p is considered a program and the second argument y is the input for this program. We define the conditional complexity function as:

$$\mathrm{C}_D(x \mid y) := \min\{|p| : D(p, y) = x\}.$$

Here $|p|$ denotes the length of the binary string p; thus, the right-hand side is the minimal length of a program that produces output x given input y.

Theorem 2.1 (Kolmogorov-Solomonoff)**.** *There exists a universal conditional decompressor U such that for every other conditional decompressor D, there exists a constant M such that*

$$\mathrm{C}_U(x \mid y) \leq \mathrm{C}_D(x \mid y) + M$$

for all strings x and y.

The proofs for all statements in this section can be found in [12].

Note that unconditional complexity can be naturally derived from the conditional version by considering the condition to be the empty string. Let U be a universal conditional decompressor; we define:

$$\mathrm{C}_U(x) := \mathrm{C}_U(x \mid \text{empty string}).$$

It is easy to verify that if U is universal for conditional complexity, then the restricted decompressor is also universal for unconditional complexity.

We will use the following basic properties of Kolmogorov complexity, which hold for any fixed universal decompressor U:

- **Bounded by length:** There exists a constant M such that for all x:

$$\mathrm{C}_U(x) \le |x| + M.$$

- **Computable transformations:** For every total computable function f, there exists a constant M such that for all x:

$$\mathrm{C}_U(f(x)) \le \mathrm{C}_U(x) + M.$$

- **Counting descriptions:** For every string y and every integer k, the number of strings x satisfying $\mathrm{C}_U(x \mid y) \le k$ is at most 2^{k+1}.

The Kolmogorov complexity $\mathrm{C}_U(x, y)$ of a pair of strings x and y is defined using a computable pairing function. Let $(x, y) \mapsto [x, y]$ be an injective computable function that maps a pair of strings to a single string. Then:

$$\mathrm{C}_U(x, y) := \mathrm{C}_U([x, y]).$$

This definition depends on the choice of the pairing function, but only up to an additive $O(1)$ term. Similarly, one can define the complexity of triples of strings, integers, or any finite objects.

For every natural number n, its Kolmogorov complexity $\mathrm{C}_U(n)$ is at most $\log n + O(1)$ (corresponding to the length of its binary representation).

Prefix complexity

Although the main result concerns plain Kolmogorov complexity, we will use prefix complexity as a technical tool in the final section. We briefly recall the necessary definitions and standard facts; see [8, 12] for details.

Prefix complexity. Fix a universal prefix-free decompressor D. The conditional prefix complexity is defined by

$$\mathrm{K}(x \mid y) := \min\{|p| : D(p, y) = x\},$$

where for every y the domain of $D(\cdot, y)$ is prefix-free.

The unconditional version is $\mathrm{K}(x) := \mathrm{K}(x \mid \varepsilon)$, where ε denotes the empty string.

Proposition 2.2 (Basic properties [8, Ch. 3],[12])**.** *For all strings x, y, z:*

$$\begin{aligned}
\mathrm{K}(x \mid y, z) &\le \mathrm{K}(x \mid y) + O(1),\\
\mathrm{K}(x, y \mid z) &\ge \mathrm{K}(x \mid z) + O(1),\\
\mathrm{C}_U(x \mid y) &\le \mathrm{K}(x \mid y) + O(1),\\
\mathrm{K}(x \mid y) &\le \mathrm{C}_U(x \mid y) + O(\log \mathrm{C}_U(x \mid y)).
\end{aligned}$$

Universal semimeasure. A function $\mu(x \mid y)$ is a *conditional semimeasure* if for every y,

$$\sum_x \mu(x \mid y) \le 1.$$

It is *lower semicomputable* if $\mu(x \mid y)$ can be effectively approximated from below by rationals.

It is well known (see [8, 12]) that there exists a maximal lower semicomputable conditional semimeasure $\mathbf{m}(x \mid y)$, meaning that for every such μ there is a constant $c > 0$ such that

$$\mu(x \mid y) \le c\,\mathbf{m}(x \mid y) \quad \text{for all } x, y.$$

This $\mathbf{m}$ is called the *universal semimeasure* (or universal a priori probability).

Theorem 2.3 (Coding Theorem [8, Th. 4.3.3], [12])**.** *For all strings x, y,*

$$\mathrm{K}(x \mid y) = -\log \mathbf{m}(x \mid y) + O(1).$$

Theorem 2.4 (Chain Rule [8, Th. 3.9.1])**.** *For all strings x, y, z,*

$$\mathrm{K}(x, y \mid z) = \mathrm{K}(x \mid z) + \mathrm{K}(y \mid x, \mathrm{K}(x \mid z), z) + O(1).$$

Corollary 2.5. *For all strings x, y, z,*

$$\mathrm{K}(y \mid z) \leq \mathrm{K}(x \mid z) + \mathrm{K}(y \mid x) + O(1).$$

Proposition 2.6 (Complexity Self-Sufficiency)**.** *For all strings x, y,*

$$\mathrm{K}(x, \mathrm{K}(x \mid y) \mid y) = \mathrm{K}(x \mid y) + O(1).$$

Proof. Apply Theorem 2.4 with $z = y$ and let $k = \mathrm{K}(x \mid y)$. Then

$$\mathrm{K}(x, k \mid y) = \mathrm{K}(x \mid y) + \mathrm{K}(k \mid x, \mathrm{K}(x \mid y), y) + O(1).$$

Since k is explicitly given in the condition, the second term is $O(1)$. □

Proposition 2.7 (Plain vs. Prefix [12, Th. 72])**.** *For every string x,*

$$\mathrm{C}_U(x) = \mathrm{K}(x \mid \mathrm{C}_U(x)) + O(1).$$

Corollary 2.8 (Equivalence of Bounds)**.** *For every string x and integer q:*

1. *If $\mathrm{C}_U(x) \leq q$, then $\mathrm{K}(x \mid q) \leq q + O(1)$.*
2. *If $\mathrm{K}(x \mid q) \leq q$, then $\mathrm{C}_U(x) \leq q + O(1)$.*

Proof. **1.** Assume $\mathrm{C}_U(x) \leq q$ and let $k = \mathrm{C}_U(x)$. By Corollary 2.5,

$$\mathrm{K}(x \mid q) \leq \mathrm{K}(x \mid k) + \mathrm{K}(k \mid q) + O(1).$$

By Proposition 2.7, $\mathrm{K}(x \mid k) = k + O(1)$. Let $\delta = q - k \geq 0$. Then $\mathrm{K}(k \mid q) = \mathrm{K}(\delta \mid q) + O(1)$. Since $\mathrm{K}(\delta \mid q) \leq 2 \log \delta + O(1)$, we obtain

$$\mathrm{K}(x \mid q) \leq k + 2 \log(q - k) + O(1) \leq q + O(1).$$

2. Assume $\mathrm{K}(x \mid q) \leq q$. Let p be a shortest prefix-free program producing x from q, and let $|p| = l \leq q$. Put $\delta = q - l \geq 0$. To describe x for the plain machine, it suffices to give p and δ. Then

$$\mathrm{C}_U(x) \leq l + 2 \log \delta + O(1) = q - \delta + 2 \log \delta + O(1) \leq q + O(1),$$

since $2 \log \delta - \delta$ is bounded from above. □

3 Proof Strategy: The Counting Argument

We aim to show that there exists a universal machine U such that $H \notin \mathrm{P}^{F_{\mathrm{C}_U}}$. In this section, we introduce the auxiliary task `COUNT-SIMPLE` and demonstrate that if the Halting Problem were solvable in polynomial time relative to the cumulative complexity oracle F_{C_U}, then `COUNT-SIMPLE` would also be solvable in polynomial time with access to the same oracle. In the subsequent sections, we will construct a specific U for which `COUNT-SIMPLE` cannot be solved efficiently, thereby establishing the main result by contraposition.

3.1 The Target Task: COUNT-SIMPLE

Let U be an arbitrary universal decompressor. We define the computational task COUNT-SIMPLE relative to U as follows.

Definition 3.1 (COUNT-SIMPLE)**.** *Given input 1^m, output the integer N_m representing the count of all strings with complexity at most m with respect to U:*

$$N_m = |\{x \in \{0,1\}^* : \mathrm{C}_U(x) \leq m\}|\,.$$

The value N_m is well-defined. Since any string x with $\mathrm{C}_U(x) \leq m$ is generated by a program of length at most m, and there are at most $2^{m+1} - 1$ such programs, we have the bound $0 \leq N_m < 2^{m+1}$.

3.2 Connection to the Halting Problem

We argue that if the Halting Problem were efficiently solvable using the oracle F_{C_U}, it would imply that COUNT-SIMPLE is also efficiently computable.

Lemma 3.2 (Implication of $H \in \mathrm{P}^{F_{\mathrm{C}_U}}$)**.** *If $H \in \mathrm{P}^{F_{\mathrm{C}_U}}$, then there is a polynomial-time machine M with oracle F_{C_U} that on input 1^m outputs N_m (i.e., solves COUNT-SIMPLE).*

Proof. Assume $H \in \mathrm{P}^{F_{\mathrm{C}_U}}$. Consider the predicate $Q(m,k)$: "Are there at least k distinct strings x such that $\mathrm{C}_U(x) \leq m$?"

This predicate is computably enumerable (c.e.), as one can enumerate all programs of length up to m and count their distinct outputs. Since it is c.e., $Q(m,k)$ is many-one reducible to the Halting Problem H. By our assumption $H \in \mathrm{P}^{F_{\mathrm{C}_U}}$, there exists a polynomial-time procedure to decide $Q(m,k)$ using the oracle F_{C_U}.

We can now compute the exact value of N_m using binary search on the range $[0, 2^{m+1}]$:

1. Initialize the search range with $L = 0$ and $R = 2^{m+1}$. Initialize the answer $ans = 0$.
2. While $L \leq R$, perform the following steps:
   - Let $mid = \lfloor (L+R)/2 \rfloor$.
   - Query the oracle to check if $Q(m, mid)$ is true (i.e., is $N_m \geq mid$?).
   - If the answer is **yes**: The true count is at least mid. Update the tentative answer $ans \leftarrow mid$ and search the upper half by setting $L = mid + 1$.
   - If the answer is **no**: The true count is strictly less than mid. Search the lower half by setting $R = mid - 1$.
3. Return ans.

This algorithm performs $O(m)$ queries. Since the input length is m, and each query takes polynomial time, the total running time is polynomial in m. Thus, a machine M solving COUNT-SIMPLE exists. □

4 Framework for the Construction of U

We construct U by combining a fixed optimal machine V with a constructed partial function G.

Definition 4.1 (Universal Machine Structure)**.** *Let V be a fixed optimal universal decompressor. The construction of our machine U depends on a constant $d \geq 1$ and a partial computable function G, both of which will be determined in subsequent sections. We define U on input $w \in \{0,1\}^*$ as follows:*

1. ***Simulation Mode:*** *If $w = 0^d p$ (a string of d zeros followed by p), then $U(w) = V(p)$.*
2. ***Diagonalization Mode:*** *If $w = 1p$, then $U(w) = G(p)$.*

This structure ensures universality. Specifically, for any string x, we have:

$$\mathrm{C}_U(x) \leq \mathrm{C}_V(x) + d.$$

This bound holds regardless of how we define G, provided d is a fixed constant ($d \geq 1$ guarantees that the domains of the two modes are disjoint). Our strategy relies on defining G to create "short" programs (via the $1p$ prefix) for specific target strings, effectively bypassing the d-bit overhead for those specific cases.

4.1 The Diagonalization Goal

Let $\{M_1, M_2, \dots\}$ be an enumeration of all polynomial-time oracle Turing machines. We require that this enumeration is redundant: every distinct polynomial-time oracle machine appears in the sequence $\{M_i\}_{i\in\mathbb{N}}$ infinitely often.

We will construct U such that for every machine M_i in this enumeration, there exists a specific input length where M_i fails to count correctly. Specifically, we ensure:

$$\forall i, \exists m : M_i^{F_{\mathrm{C}_U}}(1^m) \neq N_m.$$

4.2 Stage Intervals and the Tower Sequence

To manage the dependencies between oracle queries and the construction of U, we define a rapidly growing sequence of lengths. Let $a_0 = 1$ and define the sequence inductively by:

$$a_{l+1} = 2^{a_l}.$$

(This is often referred to as the tower sequence).

For each step $l \geq 1$, we define the target input length m for the diagonalization against machine M_l as:

$$m = n + d, \quad \text{where } n = a_{2l}.$$

Remark 4.2 (Notation: n vs m)**.** *We distinguish between the parameter n (tied to the base machine V) and the target length m (tied to U). Since U simulates V with an overhead of d bits (Definition 4.1), any string with $\mathrm{C}_V(x) \leq n$ automatically satisfies $\mathrm{C}_U(x) \leq n + d = m$. Thus, by manipulating the enumeration of n-simple strings in V, we directly influence the set of m-simple strings in U.*

We associate a specific "working zone" of lengths with each machine M_l. The zone for level l consists of strings with lengths strictly between a_{2l-1} and a_{2l+1}. Within this zone, we will define the partial function G (and thus the values of the oracle F_{C_U}) to ensure that the count produced by M_l is incorrect.

The logic of the construction is as follows:

1. The machine M_l, on input 1^m, may query the oracle F_{C_U} about various strings.
2. We do not control the oracle's answers regarding complexity values $k \leq \log n$. Note that $\log n = \log a_{2l} = a_{2l-1}$. These values are effectively determined by previous levels of the construction (or the base machine V).
3. We explicitly define G for strings in the current zone (lengths from $a_{2l-1} + 1$ to a_{2l+1}) to force the discrepancy $M_l^{F_{\mathrm{C}_U}}(1^m) \neq N_m$.

Crucially, our construction is designed to work regardless of the oracle's answers for "small" complexities ($k \leq \log n$).

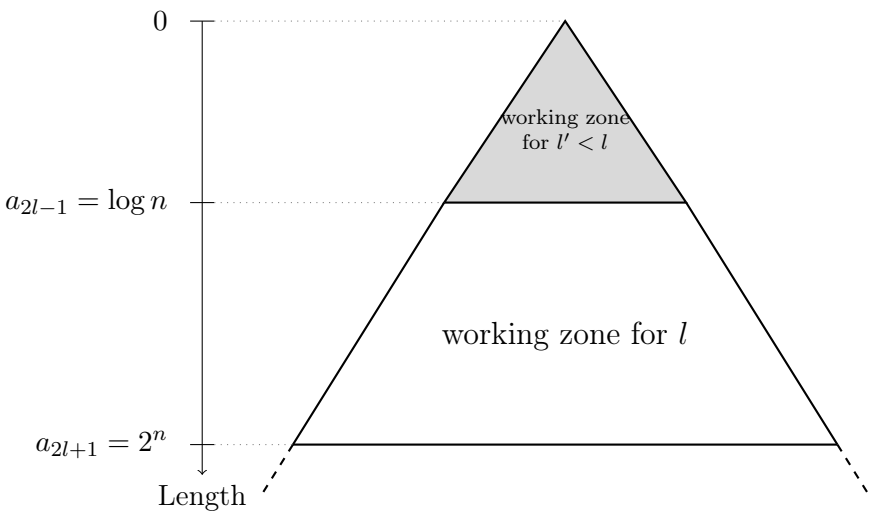

Figure 1: Structure of the construction. The gray area represents the history, while the white area is the active zone for level l.

4.3 Heuristic Model: The Marking Game

In this subsection, we outline the intuition behind the construction using a game-theoretic model.

The function G (from Definition 4.1) will be defined via a winning strategy in a specific game played between two players: Alice (representing the constructed function G) and Bob (representing the fixed optimal decompressor V), overseen by a Referee.

The rules of this game are dictated by the properties of Kolmogorov complexity. If we rely solely on the "basic", standard properties of complexity, we arrive at the formulation described below. The central difficulty is that the winner of this specific "basic" game is currently unknown.

In the subsequent sections, we will prove more advanced structural properties of Kolmogorov complexity. These properties allow us to define a "refined combinatorial game" (a modification of the one presented here), for which we can rigorously prove that Alice possesses a winning strategy. This strategy effectively defines G.

Thus, formally, the game described in this subsection is not used in the final proof. However, we strongly recommend reading this discussion for two reasons:

1. **Potential Simplification:** It is possible that the reader might identify a winning strategy for Alice in this simpler game. Such a discovery would yield a significantly simpler proof of the main theorem.
2. **Didactic Value:** The mechanics of this game provide the necessary intuition to understand the more complex game and the formal construction presented in the later sections.

The Setup. Fix a polynomial-time oracle machine M and an input length m. The interaction involves an oracle that answers queries of the form: "Is $\mathrm{C}_V(x) \leq k$?".

- **Initialization:** Initially, the complexity of every string x is defined to be its length: $\mathrm{C}_V(x) = |x| + O(1)$.
- **The Players: Alice** (representing G) controls the diagonalization logic. **Bob** (representing V) controls the standard compression.
- **The Goals:** The machine M attempts to compute the count $N_m = |\{x \mid \mathrm{C}_V(x) \leq m\}|$.
  - Bob wants the output to be correct: $M(1^m) = N_m$.
  - Alice wants the output to be incorrect: $M(1^m) \neq N_m$.

The Rules. The game proceeds in rounds. In each round, a player can choose a string x and a value $k < |x|$ to **reduce** the complexity of x to k (effectively declaring $\mathrm{C}_V(x) = k$). The players are constrained by the following budgets:

- **Bob's Budget:** For every integer k, Bob can ensure that at most 2^{k-d} strings have complexity $\leq k$.
- **Alice's Budget:**
  - For "short" complexities ($k \leq \log m$), Alice cannot make any moves.
  - For "long" complexities ($k > \log m$), Alice can ensure that at most 2^{k-2} strings have complexity $\leq k$.

Winning Condition. The game ends when a player is unable to make a move to change the outcome. Specifically, a player **loses** if the current output of the machine M relative to the current state is unsatisfactory for them (i.e., $M(1^m) = N_m$ for Alice, or $M(1^m) \neq N_m$ for Bob), but they have no legal moves or remaining budget to alter the machine's behavior or the target count N_m.

If Alice has a winning strategy against *any* allowable strategy of Bob, we can construct the partial function G to satisfy Theorem 1.1.

The Obstacle. It is currently unknown whether Alice has a winning strategy in this general game. The difficulty lies in the interactions with the oracle. Since Alice cannot modify complexities for $k \leq \log m$, she has no control over the answers to queries about small complexities made by M. If M's behavior depends critically on these short queries (which are effectively controlled by Bob or fixed by history), Alice might find herself unable to force a change in M's output, regardless of her moves on longer strings.

The Resolution. To resolve this, we refine the model by observing that "Bob" (the optimal decompressor V) cannot mark strings in an arbitrary order. The enumeration of non-random strings satisfies specific properties. For instance, after the last string of complexity n appears in the enumeration, a large number (roughly 2^n) of strings with complexity up to $2n$ will appear.

In the following section, we will formalize these properties. We will then define a "refined combinatorial game" that incorporates these constraints on Bob's moves. We will show that in this strictly constrained setting, Alice possesses a provable winning strategy, which in turn defines the required function G.

5 Structural Properties of the Enumeration

In the previous section, we stated that the enumeration of non-random strings (represented by Bob) is constrained. Here, we formalize this constraint.

Fix a canonical computable enumeration of the set $S = \{\langle x, k\rangle \mid \mathrm{C}_V(x) \leq k\}$. This corresponds to the standard enumeration of the domain of the optimal decompressor V.

Definition 5.1. *Denote by w_n the last string such that $\mathrm{C}_V(x) \leq n$ that appears in this enumeration.*

For any $u > n$, let us analyze the "tail" of the enumeration for threshold u relative to the moment w_n appears.

Definition 5.2. *Denote by* $\mathrm{Rest}(n, u)$ *the number of strings of complexity at most u that appear in the enumeration* after w_n.

We now prove that this tail cannot be arbitrarily large. Specifically, if u is computable from n, the remaining count is bounded by $2^{u-n+O(1)}$. Although this result is likely known, we provide its proof for the sake of completeness. First, we establish the complexity property of the marker string w_n.

Proposition 5.3. *The conditional complexity of w_n satisfies:*

$$\mathrm{C}_V(w_n \mid n) = n - O(1).$$

Proof. The upper bound is immediate; we prove the lower bound.

Let p be a shortest program for w_n given n. Let $|p| = n - c$ for some c.

Observe that the value n can be reconstructed from the pair (p, c) (specifically, $n = |p| + c$). Consider the following constructive procedure given p and c:

1. Compute $n = |p| + c$.
2. Compute $w_n = U(p, n)$.
3. Run the enumeration of the set S until the pair (w_n, n) appears.
4. At this point, the list of all strings with complexity $\leq n$ is complete. Output the lexicographically first string z not present in this list.

By definition, the resulting string z must satisfy $\mathrm{C}_V(z) > n$. On the other hand, z is specified by the pair (p, c). The complexity of this description is:

$$\mathrm{C}_V(z) \leq |p| + O(\log c) = (n - c) + O(\log c).$$

Combining these inequalities, we get:

$$n < \mathrm{C}_V(z) \leq n - c + O(\log c).$$

This implies $c \leq O(\log c)$, which is only possible if $c = O(1)$. Thus, $|p| \geq n - O(1)$. □

We can now prove the main bound on the enumeration tail.

Proposition 5.4 (Tail Upper Bound)**.** *Let $n < u$ be integers such that u is efficiently computable from n (i.e., $\mathrm{C}_V(u \mid n) = O(1)$). Then:*

$$\mathrm{Rest}(n, u) \leq 2^{u-n+O(1)}.$$

Proof. Let $R = \mathrm{Rest}(n, u)$. Let h be the maximal integer satisfying:

$$R \geq 2^{u-n+h}.$$

Our goal is to prove that $h = O(1)$.

Let N_u be the number of strings of complexity at most u. Note that $N_u = 2^{u+O(1)}$.

We claim that w_n can be reconstructed given n, u, and the first $k = n - h + O(1)$ bits of the binary representation of N_u. Indeed, by using this information we can enumerate all strings of complexity at most u except the last portion of size at most 2^{u-n+h}.

By definition of h, all strings of complexity at most n will be enumerated within the known portion. We identify w_n as the last string in this enumeration.

Complexity Analysis. Since u is computable from n in $O(1)$, the description length of w_n given n is dominated by the prefix of N_u:

$$\mathrm{C}_V(w_n \mid n) \leq k + O(1) = n - h + O(1).$$

We now apply the lower bound from Proposition 5.3, which states $\mathrm{C}_V(w_n \mid n) \geq n - O(1)$. Combining these inequalities we get that $h = O(1)$. □

Throughout the remainder of the paper, we denote $\mathcal{N} = 2^n$.

Corollary 5.5. *There exists a constant c_1 (dependent only on V) such that for every n:*

$$\mathrm{Rest}(n, \mathcal{N}) \leq c_1 \cdot 2^{\mathcal{N}-n}.$$

However, for an arbitrary length u, the exact value of $\mathrm{Rest}(n, u)$ is equal to $2^{u-\mathrm{K}(w_n|u)+O(1)}$, where K denotes *prefix complexity*.

We cannot use such an expression in a combinatorial game directly. For strings of arbitrary complexity, we perform, in a sense, a reduction to strings of complexity $\mathcal{N}$.

More precisely, we assert that $\mathrm{C}_V(x) \leq k$ if and only if there are "many" extension tuples of the form (x, k, w, t) among the strings with complexity at most $\mathcal{N}$. But what exactly does "many" mean? It turns out that there exists a certain threshold that can be computed given w_n. The precise statement is as follows.

Lemma 5.6 (Key Lemma)**.** *There exists a deterministic algorithm $\mathcal{A}$ with the following properties.*

Input:

- *An integer n such that $n = a_i$ for some i in the tower sequence ($a_0 = 1, a_{i+1} = 2^{a_i}$).*
- *A binary string w (candidate for the last simple string w_n).*
- *A target string x of length at most $n^{\log n}$.*
- *An integer q such that $n = a_i \leq q < n^{\log n}$ (the complexity threshold).*

Output: *An integer B.*

Properties: *The algorithm $\mathcal{A}$ always halts and produces an integer B. Furthermore, if w is indeed the last string in the enumeration of complexity $\leq n$ relative to V (i.e., $w = w_n$), then the following implications hold.*

Define for arbitrary integer d the standard and relaxed extension sets relative to V:

$$T_n(x, q, w) = \{s : \mathrm{C}_V(x, q, w, s) \leq \mathcal{N}\}$$
$$T_n^{\sqrt{d}}(x, q, w) = \{s : \mathrm{C}_V(x, q, w, s) \leq \mathcal{N} + \sqrt{d}\}$$

Then, for all sufficiently large d:

1. ***Low Complexity implies High Count:*** *If the string is simple, it has many extensions.*

$$\mathrm{C}_V(x) \leq q \implies |T_n(x, q, w)| \geq B.$$

2. ***High Count (in relaxed set) implies Low Complexity:*** *If there are many extensions even in the relaxed sense, the string must be simple.*

$$|T_n^{\sqrt{d}}(x, q, w)| \geq B \implies \mathrm{C}_V(x) \leq q + \frac{d}{2}.$$

6 Game-Based Construction of G

We now turn to the formal definition of the partial function G. Recall our objective: to construct the universal decompressor U (and consequently the oracle F_{C_U}) such that for every polynomial-time oracle machine M, there exists a length m where the output $M^{F_{C_U}}(1^m)$ differs from the true count N_m.

As outlined in Section 4, the decompressor U operates in two modes: it either simulates the fixed optimal decompressor V with a constant overhead of d bits, or it executes the function G directly.

The construction proceeds by partitioning the set of string lengths into distinct "working zones." For a specific target length $m = n + d$ (derived from the tower sequence), the corresponding working zone consists of all lengths strictly between $\log n$ and 2^n.

Our goal for a given level is to define G for strings with lengths falling within this interval. We aim to define G such that the machine $M = M_l$ associated with this level, upon receiving input 1^m, produces an output distinct from N_m.

6.1 Informal Description of the Construction

Before proceeding to the formal definitions, we explain the intuition behind the construction of G.

The function G is constructed dynamically based on the behavior of the optimal decompressor V. We monitor the enumeration of all strings with V-complexity at most $n = m - d$ (note that their U-complexity is automatically at most m). Every time a new string w with $C_V(w) \leq n$ appears, we update the state of a specific combinatorial game.

This game is played between two agents: **Alice** (representing the constructed function G) and **Bob** (representing the fixed decompressor V). Bob's goal is to ensure that the machine M, running with the oracle constructed so far, outputs the correct count N_m. Alice's goal is the opposite: to force M to output an incorrect value.

The game resembles the "Marking Game" described in Subsection 4.3, but with three crucial modifications derived from our structural lemmas and the multi-stage nature of the problem:

1. **Restricted Simplification Levels:** Strings can only be simplified to complexity levels $q \geq m+1$. (This constraint reflects the assumption that the set of strings with complexity $\leq m$ is effectively stabilized by the "hidden" strings we add, and we focus on manipulating the oracle for queries larger than m).

2. **Resource-Based Budget:** There is no fixed budget for every length. Instead, there is a single global budget derived from the "Tail Upper Bound" (Corollary 5.5):

   $$\text{Budget} = c_1 \cdot 2^{\mathcal{N}-n}.$$

   Furthermore, for every string x and every target complexity q, there is a specific **price** $B(x, q)$ required to make x q-simple. This price is determined by the threshold value B from the **Key Lemma** (Lemma 5.6): $B(x, q) = \mathcal{A}(n, w, x, q - d)$.

3. **Inter-Stage Priority:** The construction involves multiple levels indexed by l running concurrently. An action taken at a lower level (e.g., making a string simple) changes the global complexity landscape, affecting the target counts N_m for all higher levels. To handle this, we employ a **priority argument**: whenever the game at level l requires defining a value of G, we immediately **abort** all active games at all higher levels. These higher stages must wait until the optimal decompressor V "catches up" and provides a new stable state (a new candidate w).

We interpret the output of V as Bob's moves, the definition of G corresponds to Alice's moves.

The Winning Strategy Argument. If Alice has a winning strategy in this game, she plays according to it, defining G such that M outputs a value different from N_m.

But what if Bob has a winning strategy? In this case, we employ a "strategy stealing" argument:

1. Before the game begins, we use G to create one additional m-simple string (a "hidden" string of length sufficiently large so that M cannot query it) that is never queried by M.
2. This action increments the true target count: $N'_m = N_m + 1$.
3. Now, Alice considers the game relative to the original target N_m. Since Bob had a strategy to force the output to be N_m, Alice can now **simulate Bob's strategy**. By playing "as Bob," she forces M to output N_m.
4. However, the true target is now $N'_m = N_m + 1$. Since M outputs N_m, the result is incorrect (Out $\neq N'_m$). Thus, Alice achieves her goal.

Correctness of U. Finally, we must ensure that U remains a valid decompressor, meaning that for any length q, we do not assign too many descriptions of length q.

- For the target length m, we add at most 2^{m-d+1} strings (corresponding to the number of times we might need to restart the game), which is a negligible fraction of 2^m.
- For lengths q between $m+1$ and 2^m, we define G according to the game. The prices $B(x, q-d)$ are derived from the Key Lemma. The second property of the Key Lemma ("High Count implies Low Complexity") guarantees that if we construct many q-simple strings via G, they must have already been relatively simple with respect to V (specifically, $\mathrm{C}_V(x) \leq q - d/2$). Since there are few such V-simple strings, the number of strings simplified by G is strictly bounded by 2^{q-2}.

This ensures that the constructed machine U is a valid universal decompressor.

6.2 The Local Game

We define an abstract combinatorial game played between two players, Alice and Bob. This game isolates the logical core of the diagonalization, abstracting away the low-level details of Turing machines.

Game Parameters. The game is played with the following fixed components:

- **The Referee:** A polynomial-time oracle machine M and a fixed input 1^m.
- **The Target:** An integer N_m (the count Bob defends).
- **Initial Complexity:** For every string x, there is an initial complexity $C_{\text{init}}(x)$.
- **Prices:** A set of price thresholds $B(x, q) \in \mathbb{N}$ for pairs (x, q).
- **Initial Counters:** Initial values $\mathrm{St}_{\text{init}}(x, q) \in \mathbb{N}$ representing partial progress.
- **Player Budgets:** Each player is endowed with an initial budget R.

Dynamic State and Virtual Complexity. The state of the game at any step t is defined by the current values of the counters $\mathrm{St}_t(x, q)$ and the remaining budgets of both players. Initially, $\mathrm{St}_0(x, q) = \mathrm{St}_{\mathrm{init}}(x, q)$.

The counters determine a "virtual complexity" function $C_t(x)$ according to the threshold rule:

$$C_t(x) = \min\left(C_{\mathrm{init}}(x), \quad \min\{q \mid \mathrm{St}_t(x, q) \geq B(x, q)\}\right).$$

In words, the complexity of string x becomes q as soon as the investment in the counter $\mathrm{St}(x, q)$ reaches the price $B(x, q)$. Note that $C_t(x)$ is non-increasing with respect to time t.

The oracle F_t is derived directly from this complexity function:

$$F_t = \{\langle x, q\rangle \mid C_t(x) \leq q\}.$$

The Game Protocol. The game proceeds in discrete turns. At each step t, we perform the following:

1. **Evaluation:** We run the machine M with the current oracle F_t and observe the output $\mathrm{Out}_t = M^{F_t}(1^m)$.
2. **Determine Active Player:** The player unsatisfied with the current result must move.
   - If $\mathrm{Out}_t = N_m$, the current state favors Bob. **Alice is active** (she must force a change).
   - If $\mathrm{Out}_t \neq N_m$, the current state favors Alice. **Bob is active** (he must force a change back).
3. **Action:** The active player chooses a set of non-negative increments $\Delta(x, q)$ to add to the counters:
   $$\mathrm{St}_{t+1}(x, q) = \mathrm{St}_t(x, q) + \Delta(x, q).$$
   The total cost of the move is $\sum_{x,q} \Delta(x, q)$. This cost is deducted from the **active player's** remaining budget.

Winning Condition. Since the budgets are finite and integer-valued, the game must eventually terminate. A player **loses** immediately if they are the active player but cannot make a move that changes the machine's output (either due to a lack of sufficient budget or because no such set of increments exists).

Consequently:

- **Alice wins** if the game stabilizes at an output $\mathrm{Out} \neq N_m$.
- **Bob wins** if the game stabilizes at an output $\mathrm{Out} = N_m$.

6.3 Definition of G

We define the partial function G implicitly through an auxiliary generative process called **Algorithm $\mathcal{F}$** ("The Fighter"). This algorithm runs in parallel with the fixed optimal decompressor V and manages the diagonalization games for all levels simultaneously based on a priority system.

The algorithm $\mathcal{F}$ outputs **directives**—instructions to assign specific complexities to strings. All directives have the uniform structure:

"Make string z L-simple."

The function G is determined by these directives. To implement a directive that requires a string z to have complexity L, we consume the next available input string p of length $L - 1$ and

define $G(p) = z$. (In the next subsection, we will prove that for any q, $\mathcal{F}$ issues at most 2^{q-1} simplification directives, ensuring U remains a valid decompressor).

Let $\{M_1, M_2, \dots\}$ be an enumeration of all polynomial-time oracle Turing machines such that every machine appears in the sequence infinitely often.

Levels and Status. We partition the problem into **Levels**. Level l is responsible for handling strings with complexities q in the interval:

$$a_{2l-1} < q \leq a_{2l+1}.$$

For each level l, we denote the "base threshold" as $n = a_{2l-1}$. At any point in time, each level l is in one of two states:

- **Waiting (Standby):** The level is inactive, waiting for a stable candidate w from V.
- **Active (Game):** The level has a valid candidate w and is actively playing the local game against machine M_l.

The Event Loop. Algorithm $\mathcal{F}$ monitors the output of V. Suppose V outputs a new description of length q for a string x (establishing a new current upper bound $\mathrm{C}_V(x) \leq q$). We determine the level l such that $a_{2l-1} < q \leq a_{2l+1}$. Let $n = a_{2l-1}$. We distinguish two cases based on the value of q.

Case 1: New Candidate ($q \leq n$)

If $q \leq n$, we treat this string (let's call it w) as a new candidate for the *last* string of complexity $\leq n$ in the enumeration of V.

The appearance of a new candidate invalidates any previous game at this level. We switch the status of Level l to **Active** and initialize a new **Local Game** (as described in Subsection 6.2) with the following parameters:

1. **The Referee:** The polynomial-time oracle machine M_l (from our enumeration) running on fixed input 1^m, where $m = n + d$.
2. **The Target:** The current value of N_m (the current number of strings whose U-complexity does not exceed m).
3. **Initial Complexity:** For every string x, we set $C_{\text{init}}(x)$ to be its current complexity $\mathrm{C}_U(x)$ (determined by V and previous directives).
4. **Prices:** We invoke algorithm $\mathcal{A}$ (Lemma 5.6) to compute thresholds:
$$B(x, q) = \mathcal{A}(n, w, x, q - d).$$
5. **Initial Counters:** We initialize the counters $\mathrm{St}_{\text{init}}(x, q)$ by calculating the number of extensions already present in the system. Specifically:
$$\mathrm{St}_{\text{init}}(x, q) = |\{s : \mathrm{C}_U(x, q - d, w, s) \leq \mathcal{N}_d\}|,$$
where $\mathcal{N}_d = 2^n + d$.
6. **Player Budgets:** The global budget for each player is set to:
$$R = c_1 \cdot 2^{\mathcal{N}-n},$$
where c_1 is the constant from Corollary 5.5.

Immediately upon initialization, we determine the **Strategy**:

- **If Bob wins** (the current state forces output N_m): We apply **Strategy Stealing**. We select a "dummy" string z (sufficiently long to be unqueried by M_l) and issue a directive to **make** z m**-simple**. This changes the target to $N_m + 1$, effectively handing the winning position to Alice.
- **If Alice wins** (or after Strategy Stealing): We proceed to execute Alice's strategy (described below).

Priority Enforcement: Since a new game has started at level l, we immediately **abort** all games at higher levels ($l' > l$). These levels switch to **Waiting** status.

Case 2: Non-Candidate Output ($q > n$)

If V outputs a string with complexity $q > n$ (within the level's range), the reaction depends on the current status of Level l:

- **Status: Waiting.** We do nothing.
- **Status: Active.** We continue executing Alice's winning strategy based on the updated state.

Executing Alice's Strategy. When the level is Active, $\mathcal{F}$ generates directives to advance Alice's position.

- **Increments (Fueling):** If the strategy requires increasing a counter $\mathrm{St}(x, q)$, we issue a directive to **make the string encoding the tuple** $\langle x, q - d, w, s\rangle$ $\mathcal{N}_d$**-simple**, where s is a new unique index.
- **Payoff (Simplification):** If a counter $\mathrm{St}(x, q)$ reaches the price $B(x, q)$, we add x to a *preliminary list* of simple strings. We then monitor the **Safety Check**:

  $$\text{Is } \mathrm{C}_V(x) \leq q - d/2?$$

  If this condition holds currently or becomes true at any point in the future, $\mathcal{F}$ issues a directive to **make** x q**-simple**. (If w is truly the last string, Lemma 5.6 guarantees this check will pass).

Global Priority Rule: Whenever $\mathcal{F}$ issues *any* directive for Level l, the global complexity landscape changes. Consequently, we immediately **abort** all games at higher levels ($l' > l$), switching them to **Waiting** status.

6.4 Correctness of the Construction: Resource Accounting

To prove that the function G is well-defined, we must demonstrate that the algorithm $\mathcal{F}$ never runs out of input strings. Since G maps inputs p to outputs y to enforce complexity constraints $\mathrm{C}_U(y) \leq |p|$, we must ensure that for any target complexity length L, the number of inputs of length $L - 1$ consumed by the construction is strictly bounded by the total number of available strings (2^{L-1}).

We analyze the three types of operations consuming input strings. We assume the constant d is chosen sufficiently large to satisfy the inequalities derived below.

1. Dummy Strings (Target complexity m). These strings are used in the "Strategy Stealing" step (Target Flipping). We define a new dummy string only when a local game phase restarts (i.e., a new candidate w appears).

- **Consumption:** The number of phases corresponds to the number of strings w such that $\mathrm{C}_V(w) \leq n$ (where $n = m - d$). This count is bounded by 2^{n+1}.
- **Available Space:** The inputs used here have length $m - 1 = n + d - 1$. Thus, there are 2^{n+d-1} available strings.
- **Check:** We require $2^{n+1} < 2^{n+d-1}$. This inequality holds for any $d \geq 3$.

2. Tuple Strings (Target complexity $\mathcal{N}_d$). These strings are used in the "Fuel" step to increment counters.

- **Consumption:** The total number of increments is the product of the number of games played and the budget allocated to each game.
  - Max number of games (candidates w): 2^{n+1}.
  - Budget per game (from Corollary 5.5): $c_1 \cdot 2^{\mathcal{N}-n}$.

  Multiplying these, we get the total consumption bound:

$$\text{Total Fuel} \leq 2^{n+1} \cdot c_1 \cdot 2^{\mathcal{N}-n} = 2c_1 \cdot 2^{\mathcal{N}}.$$

- **Available Space:** The inputs used here have length $\mathcal{N}_d - 1 = \mathcal{N} + d - 1$. Thus, the capacity is $2^{\mathcal{N}+d-1}$.
- **Check:** We need to ensure:

$$2c_1 \cdot 2^{\mathcal{N}} < 2^{\mathcal{N}+d-1}.$$

  This inequality clearly holds for sufficiently large d.

3. Realization Strings (Target complexity q). These strings are used in the "Payoff" step to declare a string x to be q-simple.

- **Consumption:** The algorithm $\mathcal{F}$ issues a directive for x *only if* the safety check passes: $\mathrm{C}_V(x) \leq q - d/2$. The number of such strings x is bounded by $2^{q-d/2+1}$.
- **Available Space:** There are 2^{q-1} input strings of length $q - 1$.
- **Check:** We require $2^{q-d/2+1} < 2^{q-1}$. This simplifies to $d > 4$.

Conclusion. By choosing a sufficiently large constant d (dependent only on the base machine V), we guarantee that for every relevant length, the construction utilizes only a negligible fraction of the available input strings. Thus, G is well-defined and can fully realize Algorithm $\mathcal{F}$.

6.5 Proof of the Main Theorem

We are now ready to prove Theorem 1.1.

The decompressor U is defined in Definition 4.1, and the function G is defined in Subsection 6.3. The constant d is chosen sufficiently large to satisfy the resource accounting conditions from the previous subsection, as well as the conditions derived below.

Recall that by Lemma 3.2, to show that $H \notin \mathrm{P}^{F_{\mathrm{C}_U}}$, it suffices to show that for every polynomial-time oracle machine M, there exists an input length m such that $M^{F_{\mathrm{C}_U}}(1^m) \neq N_m$, where $N_m = |\{x : \mathrm{C}_U(x) \leq m\}|$.

Fix a machine M from the enumeration. Let l be the index such that $M = M_l$. Consider the parameters associated with level l:

- Base threshold: $n = a_{2l-1}$.
- Target length: $m = n + d$.

Consider the execution of the fixed optimal machine V. Let w be the *last* string appearing in the enumeration of V such that $\mathrm{C}_V(w) \leq n$. Once w appears, the algorithm $\mathcal{F}$ identifies it as a candidate and initializes the final phase of the game at level l. At this moment, the value of N_m is determined (possibly incremented by exactly one if the construction injected a dummy string in the "Strategy Stealing" step). From this point forward, the construction of G executes **Alice's winning strategy**.

Stability against Lower Stages. Note that the candidate string w is produced by V and satisfies $\mathrm{C}_V(w) \geq n - O(1)$ (by Proposition 5.3). Any string x affected by games at lower levels $l' < l$ has complexity $O(\log n)$, which is drastically smaller than n. Thus, the candidate w is distinct from any artifact produced by lower stages.

We must verify that the interaction between V (representing Bob) and G (representing Alice) proceeds according to the rules of the Local Game, and that the moves made by G are valid.

1. Bob (V) follows the Game Rules. We interpret the subsequent output of V as moves by Bob.

- **Budget Constraint:** Bob's moves are strings s with complexity $\mathrm{C}_V(\dots) \leq \mathcal{N}_d$. By the *Tail Upper Bound* (Corollary 5.5), the number of such strings appearing after w is bounded by $c_1 \cdot 2^{\mathcal{N}-n}$. This matches the finite budget allocated to the game.
- **Consistency Constraint:** If V outputs a description that makes a string x simple (i.e., $\mathrm{C}_V(x)$ drops to $q - d$), then by Item 1 of the *Key Lemma*, the number of simple extension tuples (of the form $x, q-d, w, s$) in V must be at least $B(x, q)$. Since U simulates V, these tuples also exist in U. Thus, whenever Bob forces the complexity of x to drop, the counters $\mathrm{St}(x, q)$ naturally satisfy the threshold. Bob is strictly bound by the price mechanism.

2. Alice (G) makes Valid Moves. Since Alice plays a winning strategy, she forces the abstract game to an outcome Out $\neq N_m$. However, for this to hold in reality, every time Alice "buys" a heavy query $\langle x, q\rangle$ by filling the counter $\mathrm{St}(x, q) \geq B(x, q)$, the function G must successfully define a short description for x. This requires the **Safety Check** $\mathrm{C}_V(x) \leq q - d/2$ to pass.

We prove that for the correct string w, this check always passes.

Lemma 6.1 (Realization Lemma)**.** *Let w be the true last string of V-complexity $\leq n$. Let x be a string and q be a threshold such that Alice has filled the reservoir with tuples in U:*

$$|\{s : \mathrm{C}_U(x, q - d, w, s) \leq \mathcal{N}_d\}| \geq B(x, q),$$

where $\mathcal{N}_d = 2^n + d$ and $B(x, q) = \mathcal{A}(n, w, x, q - d)$. Then, for all sufficiently large d:

$$\mathrm{C}_V(x) \leq q - d/2.$$

Proof. Consider the set of all strings with U-complexity at most $\mathcal{N}_d$:

$$S = \{y \in \{0, 1\}^* : \mathrm{C}_U(y) \leq \mathcal{N}_d\}.$$

The size of this set is bounded by the total number of descriptions provided by V plus the total budget used by G. As established by the *Tail Upper Bound* (which limits V) and the *Resource Accounting* (which limits G), the total count is bounded by:

$$|S| \leq c_{\text{total}} \cdot 2^{\mathcal{N}},$$

where c_{total} is a constant independent of d.

Since V is an optimal universal machine, it can simulate the enumeration of the set S. Any string $y \in S$ can be described by its index in this enumeration and a fixed overhead for describing the machine U and the game parameters. The description length in V is bounded by:

$$\mathrm{C}_V(y) \leq \log |S| + O(\log d) \leq \mathcal{N} + O(\log d).$$

Thus, for sufficiently large d, all tuples s generated by U in the game fall into the "relaxed" set defined in the Key Lemma:

$$T_n^{\sqrt{d}}(x, q-d, w) = \{s : \mathrm{C}_V(x, q-d, w, s) \leq \mathcal{N} + \sqrt{d}\}.$$

The premise of the lemma states that the size of this set is at least $B(x, q)$. Applying Item 2 of the *Key Lemma* (with target complexity $q - d$), we conclude:

$$\mathrm{C}_V(x) \leq (q-d) + d/2 = q - d/2.$$

□

Conclusion. The Realization Lemma ensures that every move dictated by Alice's strategy passes the safety check in the definition of G. Consequently, G successfully forces $\mathrm{C}_U(x) \leq q$ whenever the strategy requires it. Since Alice follows a winning strategy in the Local Game, the final output of the machine M with oracle F_{C_U} will differ from the target count N_m:

$$M^{F_{\mathrm{C}_U}}(1^m) \neq N_m.$$

This holds for every machine M_l at its corresponding level. Thus, $H \notin \mathrm{P}^{F_{\mathrm{C}_U}}$. □

7 Proof of the Key Lemma

7.1 Notations and the outline of the proof

Before detailing the proof strategy, we introduce the necessary notation.

Notation and Definitions. Throughout this section, we use the notation $\overset{*}{=}$, $\overset{*}{\leq}$, and $\overset{*}{\geq}$ to denote equalities and inequalities that hold up to a multiplicative constant factor depending only on the universal machine V (i.e., $A \overset{*}{=} B$ means $c_1 A \leq B \leq c_2 A$ for some constants $c_1, c_2 > 0$).

We define the complexity thresholds for the two scales of interest:

$$\mathcal{N} = 2^n \quad \text{and} \quad \mathcal{Q} = 2^q.$$

We define the sets of valid extensions relative to these thresholds. For any string (or tuple) y, we denote:

$$T_n(y) = \{s : \mathrm{C}_V(y, s) \leq \mathcal{N}\},$$
$$T_q(y) = \{s : \mathrm{C}_V(y, s) \leq \mathcal{Q}\}.$$

The core idea of the proof is to establish a rigorous link between the plain complexity $\mathrm{C}(x)$ and the cardinality of the extension set $T_n(x, q, w)$. The strategy involves two main steps:

1. **Translation to Prefix Complexity.** Both quantities of interest can be expressed in terms of conditional prefix complexity.
   - $\mathrm{C}(x) \leq q$ is equivalent to $\mathrm{K}(x \mid q) \leq q$ (up to an additive constant) by Corollary 2.8.

- The size $|T_n(x,q)|$ is proportional to the discrete a priori probability $\mathbf{m}(x,q \mid n)$, as we will show in Subsection 7.2.

2. **Bridging the Scales.** Our goal is to relate $\mathrm{K}(x \mid q)$ and $\mathrm{K}(x,q \mid n)$.

   First, observe that $\mathrm{K}(x \mid q) = \mathrm{K}(x \mid q,n) + O(1)$, because q contains the information about n. Specifically, n is determined by q as the unique element a_i of the Tower sequence such that $a_i \leq q < a_{i+1}$.

   Consider the following equality (Chain Rule):

$$\mathrm{K}(x,q \mid n) \approx \mathrm{K}(x \mid q,n) + \mathrm{K}(q \mid n). \tag{1}$$

   However, we cannot use it directly for two reasons:

   (a) **Precision:** This equality holds only up to a logarithmic error term.

   (b) **Computability:** The function $\mathrm{K}(\cdot)$ is not computable.

3. **The Solution.** We resolve these issues as follows:

   - To handle non-computability, we utilize the string w (the last simple string of complexity n). Since the binary length of q is logarithmic in n, whereas the precision requires only $O(1)$ loss, the string w acts as a powerful oracle, allowing us to compute $\mathrm{K}(q)$ and related terms exactly.
   - To handle the precision error, we employ a **fixed-point** construction. Instead of the raw string q, we construct a structural replacement $\tilde{q} = (q,l)$ for some l such that an equation similar to (1) holds with constant precision.

7.2 Measure vs. Extension Density

We establish a correspondence between the conditional a priori probability $\mathbf{m}(y \mid n)$ and the number of simple extensions. Denote the relaxed thresholds for arbitrary $\delta \geq 0$:

$$T_n^\delta(y) = \{s : \mathrm{C}_V(y,s) \leq \mathcal{N} + \delta\}.$$

$$T_q^\delta(y) = \{s : \mathrm{C}_V(y,s) \leq \mathcal{Q} + \delta\}.$$

Lemma 7.1 (Measure Density Lemma)**.** *Let y be a string with length $|y| < \mathcal{N}/2 - O(1)$.*

1. ***Standard Threshold:***

$$\mathbf{m}(y \mid n) \overset{*}{\leq} |T_n(y)| \cdot 2^{-\mathcal{N}},$$
$$\mathbf{m}(y \mid q) \overset{*}{\leq} |T_q(y)| \cdot 2^{-\mathcal{Q}}.$$

2. ***Relaxed Threshold:*** *Let $\delta \geq 0$ be an integer. Then:*

$$\mathbf{m}(y \mid n) \overset{*}{\geq} |T_n^\delta(y)| \cdot 2^{-\mathcal{N}-O(\delta)},$$
$$\mathbf{m}(y \mid q) \overset{*}{\geq} |T_q^\delta(y)| \cdot 2^{-\mathcal{Q}-O(\delta)}.$$

Proof of Item 1. We construct a description procedure based on the measure $\mathbf{m}$. Let c be a sufficiently large constant. Discretize the probability space into "quanta" of size $\epsilon = 2^{-(\mathcal{N}-c)}$.

Consider an algorithm that simulates the enumeration of the lower semicomputable semimeasure $\mathbf{m}(\cdot \mid n)$. The algorithm takes an input r (interpreted as an index $0 \le r < 2^{\mathcal{N}-c}$) and searches for the r-th "quantum" of probability mass. Specifically, it monitors the enumeration of $\mathbf{m}(\cdot \mid n)$ and waits for the moment when for some string y', the cumulative value $\mathbf{m}(y' \mid n)$ reaches at least $k \cdot \epsilon$ for some new integer k. The algorithm matches the global index r to this specific event and outputs the pair (y', k).

Since the input r has length $\mathcal{N} - c$, the complexity of the output pair is bounded:

$$\mathrm{C}_V(y', k) \le |r| + O(1) = \mathcal{N} - c + O(1).$$

By choosing c large enough to cover the $O(1)$ overhead, we ensure $\mathrm{C}_V(y', k) \le \mathcal{N}$. Thus, k (interpreted as a string extension) belongs to the set $T_n(y')$.

Now consider our fixed string y. From the length condition $|y| < \mathcal{N}/2 - O(1)$ we have:

$$\mathbf{m}(y \mid n) \ge 2^{-(\mathcal{N}-c)} = \epsilon.$$

Since the measure $\mathbf{m}(y \mid n)$ strictly exceeds ϵ, the number of full quanta allocated to y is strictly positive. The exact number of such distinct quanta k is $\lfloor \mathbf{m}(y \mid n)/\epsilon \rfloor$. Each such quantum k corresponds to a distinct valid extension in $T_n(y)$. Therefore, $|T_n(y)| \ge \frac{1}{2}\mathbf{m}(y \mid n) \cdot 2^{\mathcal{N}-c}$, which implies:

$$\mathbf{m}(y \mid n) \stackrel{*}{\le} |T_n(y)| \cdot 2^{-\mathcal{N}}.$$

Proof of Item 2. Let $K = \mathcal{N} + \delta$. We define a specific semimeasure P_δ relative to the threshold K.

$$P_\delta(y) = |T_n^\delta(y)| \cdot 2^{-K-1}.$$

Since the set of pairs with complexity $\le K$ is computably enumerable, $P_\delta(y)$ is lower semicomputable (given n and δ). The sum of $P_\delta(y)$ over all y is bounded by the total number of pairs with complexity $\le K$, normalized by 2^{-K}:

$$\sum_y P_\delta(y) \le \sum_{y,s:\mathrm{C}(y,s)\le K} 2^{-K-1} \le 2^{K+1} \cdot 2^{-K-1} = 1.$$

Thus, P_δ is a valid semimeasure relative to the condition $\langle n, \delta \rangle$. By the maximality property of the universal semimeasure $\mathbf{m}$, we have:

$$\mathbf{m}(y \mid n, \delta) \stackrel{*}{\ge} P_\delta(y) = |T_n^\delta(y)| \cdot 2^{-\mathcal{N}-\delta-1}.$$

Using the Coding Theorem (Theorem 2.3), we relate the measures with different conditions:

$$-\log \mathbf{m}(y \mid n) = \mathrm{K}(y \mid n) + O(1) \le \mathrm{K}(y \mid n, \delta) + \mathrm{K}(\delta) + O(1).$$

In terms of measures, this translates to:

$$\mathbf{m}(y \mid n) \stackrel{*}{\ge} \mathbf{m}(y \mid n, \delta) \cdot 2^{-\mathrm{K}(\delta)} \ge \mathbf{m}(y \mid n, \delta) \cdot 2^{-O(\delta)}.$$

Combining these inequalities:

$$\mathbf{m}(y \mid n) \ge |T_n^\delta(y)| \cdot 2^{-\mathcal{N}-O(\delta)}.$$

This yields the following immediate consequence:

Corollary 7.2 (Relaxed vs. Strict Volume)**.** *For any string y with length $|y| < \mathcal{N}/2 - O(1)$ and integer $\delta \geq 0$:*

$$|T_n(y)| \geq |T_n^{\delta}(y)| \cdot 2^{-O(\delta)},$$
$$|T_q(y)| \geq |T_q^{\delta}(y)| \cdot 2^{-O(\delta)}.$$

Corollary 7.3 (Conditioning Shift)**.** *Let y and z be strings such that $\mathrm{C}(y \mid z) = \delta$ and $|y|, |z| < \mathcal{N}/2 - O(1)$. Then:*

$$|T_n(y)| \geq |T_n(z)| \cdot 2^{-O(\delta)},$$
$$|T_q(y)| \geq |T_q(z)| \cdot 2^{-O(\delta)}.$$

Proof. We prove the statement for the parameter n. Consider any s such that $\mathrm{C}_V(z, s) \leq \mathcal{N}$. We can describe the pair (y, s) by first describing y given z, and then using the description of (z, s).

$$\mathrm{C}_V(y, s) \leq \mathcal{N} + O(\delta).$$

Thus, $s \in T_n^{\delta'}(y)$ for $\delta' = O(\delta)$. Consequently, $|T_n^{\delta'}(y)| \geq |T_n(z)|$. Applying Corollary 7.2 with parameter δ', we obtain:

$$|T_n(y)| \geq |T_n^{\delta'}(y)| \cdot 2^{-O(\delta')} \geq |T_n(z)| \cdot 2^{-O(\delta)}.$$

□

7.3 The Construction of $\tilde{q}$

We require a lemma that constructs a specific string $\tilde{q}$ (related to the threshold q) with precise complexity properties.

Lemma 7.4 (The $\tilde{q}$-Structure Lemma)**.** *There exists a deterministic algorithm $\mathcal{B}$ with the following properties.*

Input:

- *An integer n.*
- *A binary string w.*
- *An integer $q \leq n^{\log n}$.*

Output: *A binary string $\tilde{q}$.*

Properties: *The algorithm $\mathcal{B}$ always halts. Furthermore, if w is the* last *string in the enumeration of the set $\{y : \mathrm{C}_V(y) \leq n\}$, then the output $\tilde{q}$ satisfies the following conditions:*

1. ***Small Length:*** *The string $\tilde{q}$ has logarithmic length.*

$$|\tilde{q}| = O(\log q).$$

2. ***Encodes q:*** *The threshold q is effectively computable from $\tilde{q}$.*

$$\mathrm{C}_V(q \mid \tilde{q}) = O(1).$$

3. ***Complexity Decomposition:*** *Let $\alpha := \mathrm{K}(\tilde{q} \mid n)$. Then:*

$$\mathrm{K}(\tilde{q} \mid n) = \mathrm{K}\Big(\tilde{q},\ \alpha,\ \mathrm{K}\big(\tilde{q}, \alpha \mid q\big) \ \Big|\ n\Big) + O(1).$$

We explain the meaning of the Complexity Decomposition condition in the next subsection. The proof of this lemma is given in Subsection 7.9.

7.4 Proof of the Complexity Symmetry Shift

We now prove the key corollary of Lemma 7.4.

Lemma 7.5 (Complexity Symmetry Shift)**.** *Assume* $\mathrm{K}(n \mid q) = O(1)$*. Let* $\tilde{q}$ *be a string satisfying the properties of Lemma 7.4. Then for any string* x*, the following identity holds up to an* $O(1)$ *additive term:*

$$\mathrm{K}(x, \tilde{q}, \mathrm{K}(\tilde{q} \mid n) \mid q) - \mathrm{K}(\tilde{q}, \mathrm{K}(\tilde{q} \mid n) \mid q) = \mathrm{K}(\tilde{q}, x \mid n) - \mathrm{K}(\tilde{q} \mid n).$$

Proof. For brevity, let us denote the Left Hand Side (relative to q) as **LHS** and the Right Hand Side (relative to n) as **RHS**.

Part 1: The "Easy" Direction (LHS $\leq$ RHS)

First, we establish that the inequality holds in one direction purely based on information content, regardless of the internal structure of $\tilde{q}$.

Analyzing the RHS. We expand the RHS using the conditional chain rule relative to n:

$$\text{RHS} = \mathrm{K}(\tilde{q}, x \mid n) - \mathrm{K}(\tilde{q} \mid n) = \mathrm{K}(x \mid \tilde{q}, \mathrm{K}(\tilde{q} \mid n), n) + O(1).$$

The condition set for the RHS is effectively:

$$C_{RHS} = \{\tilde{q},\ \mathrm{K}(\tilde{q} \mid n),\ n\}.$$

Analyzing the LHS. Let Y be the pair consisting of $\tilde{q}$ and its complexity relative to n:

$$Y = \langle \tilde{q},\ \mathrm{K}(\tilde{q} \mid n) \rangle.$$

The LHS can be rewritten using the chain rule relative to q:

$$\text{LHS} = \mathrm{K}(x, Y \mid q) - \mathrm{K}(Y \mid q) = \mathrm{K}(x \mid Y, \mathrm{K}(Y \mid q), q) + O(1).$$

Expanding Y, the condition set for the LHS is:

$$C_{LHS} = \{\tilde{q},\ \mathrm{K}(\tilde{q} \mid n),\ \mathrm{K}(Y \mid q),\ q\}.$$

Comparison. Notice that C_{LHS} contains the threshold q. Since $\mathrm{K}(n \mid q) = O(1)$, knowledge of q allows the reconstruction of n. Furthermore, C_{LHS} explicitly contains $\tilde{q}$ and $\mathrm{K}(\tilde{q} \mid n)$, which are the core components of C_{RHS}. Thus, the oracle in the LHS has access to strictly more information than the oracle in the RHS. Therefore, LHS $\leq$ RHS $+ O(1)$.

Part 2: Deriving the Necessary Structure of $\tilde{q}$

The reverse inequality, RHS $\leq$ LHS, is not true for an arbitrary string. We show that the structure of $\tilde{q}$ guaranteed by Lemma 7.4 is sufficient to make this inequality hold.

We introduce a "carrier" tuple T corresponding to the decomposition property. We require the following invariant (which is exactly Property 3 of Lemma 7.4):

$$\mathrm{K}(T \mid n) = \mathrm{K}(\tilde{q} \mid n) + O(1). \tag{2}$$

Step A: Upper Bounding the RHS. We upper bound the RHS by replacing $\tilde{q}$ with the more informative tuple T. Since T contains $\tilde{q}$, the pair $(\tilde{q}, x)$ is trivially recoverable from (T, x). Using the invariant (2), we get:

$$\begin{aligned}\text{RHS} &= \mathrm{K}(\tilde{q}, x \mid n) - \mathrm{K}(\tilde{q} \mid n) \\ &\leq \mathrm{K}(T, x \mid n) - \mathrm{K}(T \mid n) + O(1).\end{aligned}$$

Now we apply the chain rule to the term $\mathrm{K}(T, x \mid n)$:

$$\mathrm{K}(T, x \mid n) = \mathrm{K}(T \mid n) + \mathrm{K}(x \mid T, \mathrm{K}(T \mid n), n) + O(1).$$

Substituting this back, the term $\mathrm{K}(T \mid n)$ cancels out:

$$\text{RHS} \leq \mathrm{K}(x \mid T, \mathrm{K}(T \mid n), n) + O(1). \tag{3}$$

Step B: Collapsing the LHS. Recall the expansion of the LHS:

$$\text{LHS} = \mathrm{K}(x \mid \tilde{q},\ \mathrm{K}(\tilde{q} \mid n),\ \mathrm{K}(Y \mid q),\ q) + O(1),$$

where

$$Y = \langle \tilde{q}, \mathrm{K}(\tilde{q} \mid n) \rangle.$$

To enable the bound RHS $\leq$ LHS, we define T to explicitly capture all components of the LHS condition:

$$T = \Big\langle\ \tilde{q},\quad \underbrace{\mathrm{K}(\tilde{q} \mid n)}_{\alpha},\quad \underbrace{\mathrm{K}\big(\langle \tilde{q}, \alpha \rangle \mid q\big)}_{\beta}\ \Big\rangle.$$

With this specific definition:

1. T contains $\tilde{q}$ and $\alpha = \mathrm{K}(\tilde{q} \mid n)$.
2. T contains β, which is the complexity of the first two components relative to q. Thus, the term $\mathrm{K}(Y \mid q)$ in the LHS condition is computable from T.
3. Since T contains $\tilde{q}$ (and $\mathrm{C}(q \mid \tilde{q}) = O(1)$), q is computable from T.

Therefore, given T, all other conditions in the LHS are redundant:

$$\text{LHS} = \mathrm{K}(x \mid T) + O(1). \tag{4}$$

Step C: Final Comparison. We now compare the derived bounds.

- From (3): $\text{RHS} \leq \mathrm{K}(x \mid T, \mathrm{K}(T \mid n), n) + O(1)$.
- From (4): $\text{LHS} = \mathrm{K}(x \mid T) + O(1)$.

The condition in the RHS bound includes T. Since T allows the computation of $\mathrm{K}(T \mid n)$ (via α) and n (via q), the condition sets are informationally equivalent (up to $O(1)$), ensuring the inequality holds. Combining results:

$$\text{RHS} \leq \mathrm{K}(x \mid T, \dots) + O(1) \leq \mathrm{K}(x \mid T) + O(1) = \text{LHS} + O(1).$$

Thus, the Corollary is proved. □

7.5 Computability: The Oracle Power of w

The definitions of the algorithms in this section (both the threshold computation $\mathcal{A}$ and the structural search $\mathcal{B}$) rely on the prefix complexity function $\mathrm{K}(\cdot)$, which is generally uncomputable. However, these algorithms receive as input the string w, defined as the *last* string generated in the canonical enumeration of the set $\{y : \mathrm{C}_V(y) \leq n\}$.

We show that w acts as a powerful oracle. Let T_w denote the number of computation steps performed by the optimal machine V until it outputs the string w. Since w marks the completion of the enumeration for the threshold n, the time T_w acts as a universal cutoff for all computations described by simple programs.

Proposition 7.6 (Computability for Simple Objects)**.** *For any strings a and b, if their lengths satisfy*

$$|a| + |b| < n/2,$$

then the value $\mathrm{K}(a \mid b)$ is computable given inputs n, a, b and w.

Proof. First, we determine T_w by running V until it outputs w. The following procedure computes $\mathrm{K}(a \mid b)$:

1. Simulate the universal prefix decompressor on input (p, b) for all candidate programs p with length up to $|a| + O(1)$, running each for at most T_w steps.
2. Collect all programs that halt and output a.
3. Output the length of the shortest such program.

Correctness. Let p^* be a shortest program for the prefix decompressor such that it outputs a given b. We must verify that this computation halts within T_w steps.

Consider the description for the plain machine V corresponding to the execution of p^* on b. The length of this description is bounded by $|p^*| + |b| + O(1)$. Since $|p^*| \leq |a| + O(1)$, the total length is bounded by $|a| + |b| + O(1)$. By the hypothesis $|a| + |b| < n/2$, this length is strictly less than n (for sufficiently large n).

Since w is the last string generated by V with complexity $\leq n$, any computation corresponding to a shorter description must complete within T_w steps. Thus, the simulation of the prefix decompressor on p^* and b completes within the time bound. □

7.6 Volume Decomposition

We decompose the set of extensions based on the moment the string w appears. Let w be the last string in the canonical enumeration of strings with complexity at most n.

We define the **"Past" sets** St^w as the sets of extensions discovered by the universal machine prior to the appearance of w. Specifically, for any string y:

$$St_q^w(y) = \{s : \mathrm{C}_V(y, s) \leq \mathcal{Q} \text{ and } (y, s) \text{ is enumerated before } w\},$$
$$St_n^w(y) = \{s : \mathrm{C}_V(y, s) \leq \mathcal{N} \text{ and } (y, s) \text{ is enumerated before } w\}.$$

Lemma 7.7 (Volume Decomposition)**.** *Let $\tilde{q}$ and α be values defined in Lemma 7.4. Then:*

1. $|T_q(x)| \stackrel{*}{=} |\mathrm{St}_q^w(x)| + |T_q(x, \tilde{q}, \alpha)|$.
2. $|T_n(\tilde{q}, x)| \stackrel{*}{=} |\mathrm{St}_n^w(\tilde{q}, x)| + |T_n(x, q, w)|$.

Proof. We prove the equality up to a multiplicative constant ($\stackrel{*}{=}$) by establishing both the lower ($\stackrel{*}{\geq}$) and upper ($\stackrel{*}{\leq}$) bounds for each item.

Lower Bounds ($\overset{*}{\geq}$): By definition, the "Past" sets are subsets of the total extension sets, so $|T_q(x)| \geq |St_q^w(x)|$ and $|T_n(\tilde{q}, x)| \geq |St_n^w(\tilde{q}, x)|$.

For the remaining terms, we apply the Conditioning Shift (Corollary 7.3):

- For Item 1: The tuple $(x, \tilde{q}, \alpha)$ trivially determines x (so $\mathrm{C}(x \mid x, \tilde{q}, \alpha) = O(1)$). Thus, dropping the extra conditions only decreases complexity, giving $|T_q(x)| \overset{*}{\geq} |T_q(x, \tilde{q}, \alpha)|$.
- For Item 2: The tuple $(\tilde{q}, x)$ can be algorithmically reconstructed from (x, q, w) (since $\tilde{q}$ is computed from x, q, w via Lemma 7.4). Thus, $\mathrm{C}(\tilde{q}, x \mid x, q, w) = O(1)$, which implies $|T_n(\tilde{q}, x)| \overset{*}{\geq} |T_n(x, q, w)|$.

Combining these (since the sets are disjoint or one dominates), the lower bounds hold.

Upper Bounds ($\overset{*}{\leq}$): We must bound the number of "Future" extensions—those enumerated *after* w.

Proof for Item 1: Consider any string $s \in T_q(x) \setminus St_q^w(x)$. Since (x, s) has complexity $\leq \mathcal{Q}$ but is enumerated after w, any optimal program p of length $\leq \mathcal{Q}$ outputting (x, s) halts after w has appeared. Thus, by running p, we can observe the halting state. At the moment it halts, the last string of complexity $\leq n$ enumerated so far is exactly w. This allows us to algorithmically identify w during the decompression of p. From x, q, and w, we can compute $\tilde{q}$ using the algorithm from Lemma 7.4. Furthermore, since $\tilde{q}$ is simple relative to n, we can compute $\alpha = \mathrm{K}(\tilde{q} \mid n)$ using Proposition 7.6. Finally, the algorithm outputs the tuple $(x, \tilde{q}, \alpha, s)$. The length of this description is bounded by $|p| + O(1) \leq \mathcal{Q} + O(1)$. Therefore, every such "Future" extension s belongs to the relaxed set $T_q^{O(1)}(x, \tilde{q}, \alpha)$. The total number of such extensions is bounded by $|T_q^{O(1)}(x, \tilde{q}, \alpha)|$. Applying the Relaxed vs. Strict Volume property (Corollary 7.2), we get:

$$|T_q^{O(1)}(x, \tilde{q}, \alpha)| \overset{*}{\leq} |T_q(x, \tilde{q}, \alpha)|.$$

Summing the "Past" and "Future" components yields the upper bound for Item 1.

Proof for Item 2: Consider any string $s \in T_n(\tilde{q}, x) \setminus St_n^w(\tilde{q}, x)$. Similarly, the program of length $\leq \mathcal{N}$ outputting $(\tilde{q}, x, s)$ halts after w has appeared. During its decompression, we can identify w. Since $\tilde{q}$ structurally encodes q, we can trivially extract q from $\tilde{q}$. We can then output the tuple (x, q, w, s). The complexity of this pair is bounded by $\mathcal{N} + O(1)$. Therefore, all such "Future" extensions belong to $T_n^{O(1)}(x, q, w)$. By Corollary 7.2, the number of these extensions is bounded by $|T_n(x, q, w)| \cdot 2^{O(1)}$. Summing the Past and Future components yields the required upper bound for Item 2. □

7.7 Summary of Structural Properties

In this subsection, we consolidate the results established in the previous lemmas. These properties serve as the foundation for the algorithm $\mathcal{A}$ and the final counting argument.

Lemma 7.8 (Structural Volume Lemma)**.** *Let n, x, q satisfy the assumptions of the Key Lemma 5.6. Let w be the last string in the canonical enumeration of strings with complexity at most n. Let $\tilde{q}$ be the structural witness derived from n, w, q via the $\tilde{q}$-Structure Lemma 7.4, and let $\alpha := \mathrm{K}(\tilde{q} \mid n)$.*

Then the following properties hold:

1. ***Low Complexity Implies Large Volume:***

$$\mathrm{C}_V(x) \leq q \implies |T_q(x)| \overset{*}{\geq} 2^{-q} \cdot 2^{\mathcal{Q}}.$$

2. ***Volume Decomposition (Target Level):***

$$|T_q(x)| \stackrel{*}{=} |\mathrm{St}_q^w(x)| + |T_q(x, \tilde{q}, \alpha)|.$$

3. ***Log-Volume of the Remainder:***

$$-\log |T_q(x, \tilde{q}, \alpha)| = \mathrm{K}(\tilde{q}, \alpha \mid q) + \mathrm{K}(\tilde{q}, x \mid n) - \mathrm{K}(\tilde{q} \mid n) - \mathcal{Q} + O(1).$$

4. ***Computability:*** *The terms* $\mathrm{K}(\tilde{q}, \alpha \mid q)$ *and* $\mathrm{K}(\tilde{q} \mid n)$ *can be effectively computed given* n, x, q, w.

5. ***Complexity Decomposition (Resource Level):*** *The complexity at level* n *is determined by the dominant volume component:*

$$\mathrm{K}(\tilde{q}, x \mid n) = \mathcal{N} - \log \max \left(|\mathrm{St}_n^w(\tilde{q}, x)|, |T_n(x, q, w)|\right) + O(1).$$

Proof. **Item 1.** By Corollary 2.8, $\mathrm{C}_V(x) \leq q$ implies $\mathrm{K}(x \mid q) \leq q + O(1)$. By the Coding Theorem 2.3, $\mathrm{K}(x \mid q) = -\log \mathbf{m}(x \mid q) + O(1)$. Thus:

$$-\log \mathbf{m}(x \mid q) \leq q + O(1) \implies \mathbf{m}(x \mid q) \stackrel{*}{\geq} 2^{-q}.$$

Finally, by the Measure Density Lemma 7.1 (applied to threshold $\mathcal{Q}$):

$$|T_q(x)| \cdot 2^{-\mathcal{Q}} \stackrel{*}{\geq} \mathbf{m}(x \mid q) \stackrel{*}{\geq} 2^{-q}.$$

Multiplying by $2^{\mathcal{Q}}$ yields the result.

Item 2 follows directly from the definition of the decomposition sets in Lemma 7.7.

Item 3. We first express the volume of the set in terms of its complexity. By the Measure Density Lemma 7.1 (applied to threshold $\mathcal{Q}$) and the Coding Theorem:

$$\log |T_q(x, \tilde{q}, \alpha)| = \mathcal{Q} - \mathrm{K}(x, \tilde{q}, \alpha \mid q) + O(1).$$

Equivalently:

$$-\log |T_q(x, \tilde{q}, \alpha)| = \mathrm{K}(x, \tilde{q}, \alpha \mid q) - \mathcal{Q} + O(1).$$

Next, we expand the complexity term using the Complexity Symmetry Shift (Lemma 7.5):

$$\mathrm{K}(x, \tilde{q}, \alpha \mid q) - \mathrm{K}(\tilde{q}, \alpha \mid q) = \mathrm{K}(\tilde{q}, x \mid n) - \mathrm{K}(\tilde{q} \mid n).$$

Solving for $\mathrm{K}(x, \tilde{q}, \alpha \mid q)$ and substituting into the previous expression yields:

$$-\log |T_q(x, \tilde{q}, \alpha)| = (\mathrm{K}(\tilde{q}, \alpha \mid q) + \mathrm{K}(\tilde{q}, x \mid n) - \mathrm{K}(\tilde{q} \mid n)) - \mathcal{Q} + O(1).$$

Item 4 follows from Proposition 7.6, since the lengths of $\tilde{q}$, α, and q satisfy the condition of that Proposition.

Item 5. We apply the Volume Decomposition Lemma 7.7 to level n (Item 2). The total volume decomposes (up to multiplicative constants) as:

$$|T_n(\tilde{q}, x)| \stackrel{*}{=} |\mathrm{St}_n^w(\tilde{q}, x)| + |T_n(x, q, w)|.$$

Taking the logarithm:

$$\log |T_n(\tilde{q}, x)| = \log \left(|\mathrm{St}_n^w(\tilde{q}, x)| + |T_n(x, q, w)|\right) + O(1).$$

Since $\log(A + B) = \max(\log A, \log B) + O(1)$, we have:

$$\log |T_n(\tilde{q}, x)| = \max \left(\log |\mathrm{St}_n^w(\tilde{q}, x)|, \log |T_n(x, q, w)|\right) + O(1).$$

By the Measure Density Lemma (applied to threshold $\mathcal{N}$), the complexity relates to the total volume as:

$$\mathrm{K}(\tilde{q}, x \mid n) = \mathcal{N} - \log |T_n(\tilde{q}, x)| + O(1).$$

Substituting the max-expression yields the required formula. □

7.8 Proof of the Key Lemma

The Algorithm $\mathcal{A}$

We introduce a sufficiently large constant λ (to cover precision gaps in the structural lemmas). The algorithm receives inputs n, w, x, q and operates as follows:

1. **Check Triviality at q (High Past Volume):** Compute the volume of extensions of x at scale q accumulated in the structure set:
$$V_{check} = |\mathrm{St}_q^w(x)|.$$
If $V_{check} \geq 2^{-q-\lambda} \cdot 2^{\mathcal{Q}}$, output $B = 0$ and terminate.

2. **Compute Structural Correction:** Calculate the correction factor Δ using the computable terms for $\tilde{q}$ (as guaranteed by Item 4 of Lemma 7.8):
$$\Delta = \mathrm{K}(\tilde{q}, \alpha \mid q) - \mathrm{K}(\tilde{q} \mid n).$$

3. **Compute Structure Volume at n:** Compute the volume of extensions of the pair $(\tilde{q}, x)$ at scale n accumulated so far:
$$V_{str} = |\mathrm{St}_n^w(\tilde{q}, x)|.$$

4. **Output Threshold:** Define the target count:
$$\mathrm{Num} = 2^{\mathcal{N}-q+\Delta-\lambda}.$$
If $V_{str} \geq \mathrm{Num}$, output $B = 0$ (the structure is already "heavy" enough). Otherwise, output $B = \mathrm{Num}$.

Proof of Property 1: Low Complexity implies High Count

We must show that if $\mathrm{C}_V(x) \leq q$, then $|T_n(x, q, w)| \geq B$.

Step 1: Complexity to Volume. Assume $\mathrm{C}_V(x) \leq q$. By **Item 1** of Lemma 7.8 (Low Complexity Implies Large Volume), the total volume of extensions at scale q is large:
$$|T_q(x)| \overset{*}{\geq} 2^{-q} \cdot 2^{\mathcal{Q}}.$$

Step 2: Decomposition at Scale q. By **Item 2** (Volume Decomposition), the total set partitions into "Past" and "Future":
$$|T_q(x)| \overset{*}{=} |\mathrm{St}_q^w(x)| + |T_q(x, \tilde{q}, \alpha)|.$$
If the "Past" term $|\mathrm{St}_q^w(x)|$ is large enough to satisfy the condition in Step 1 of Algorithm $\mathcal{A}$, then $B = 0$, and the condition $|T_n| \geq 0$ holds trivially. Otherwise, the "Future" term must dominate:
$$|T_q(x, \tilde{q}, \alpha)| \overset{*}{\geq} 2^{-q} \cdot 2^{\mathcal{Q}}.$$

Step 3: Symmetry Application. We use **Item 3** of Lemma 7.8 to convert the volume bound at scale q into a complexity bound at scale n.
$$-\log |T_q(x, \tilde{q}, \alpha)| = \Delta + \mathrm{K}(\tilde{q}, x \mid n) - \mathcal{Q} + O(1).$$
Substituting the lower bound from Step 2 (which implies $-\log |T_q| \leq q - \mathcal{Q}$):
$$q - \mathcal{Q} \geq \Delta + \mathrm{K}(\tilde{q}, x \mid n) - \mathcal{Q} - O(1).$$
Simplifying $\mathcal{Q}$:
$$\mathrm{K}(\tilde{q}, x \mid n) \leq q - \Delta + O(1). \tag{5}$$

Step 4: Decomposition at Scale n. We map this complexity bound back to volumes at level n using **Item 5** of Lemma 7.8. Recall that the volumes are additive:

$$\mathrm{K}(\tilde{q}, x \mid n) = \mathcal{N} - \log\left(V_{str} + |T_n(x, q, w)|\right) + O(1).$$

Substituting this into inequality (5):

$$\mathcal{N} - \log\left(V_{str} + |T_n(x, q, w)|\right) \leq q - \Delta + O(1).$$

Rearranging to solve for the total volume:

$$\log\left(V_{str} + |T_n(x, q, w)|\right) \geq \mathcal{N} - q + \Delta - O(1).$$

Step 5: Final Comparison. Exponentiating the result from Step 4, we get:

$$V_{str} + |T_n(x, q, w)| \geq 2^{\mathcal{N} - q + \Delta - O(1)}. \tag{6}$$

Recall that $\mathrm{Num} = 2^{\mathcal{N} - q + \Delta - \lambda}$. The bound (6) can be rewritten as:

$$V_{str} + |T_n(x, q, w)| \geq \mathrm{Num} \cdot 2^{\lambda - O(1)}.$$

Let $C = 2^{\lambda - O(1)}$. Since λ is chosen sufficiently large, $C \geq 2$. Thus:

$$V_{str} + |T_n(x, q, w)| \geq 2 \cdot \mathrm{Num}.$$

We now analyze the two cases of the algorithm's output:

- **Case A:** $V_{str} \geq \mathrm{Num}$. The algorithm outputs $B = 0$. Since $|T_n(x, q, w)| \geq 0$, the condition holds.
- **Case B:** $V_{str} < \mathrm{Num}$. The algorithm outputs $B = \mathrm{Num}$. From our derived inequality:

  $$|T_n(x, q, w)| \geq 2 \cdot \mathrm{Num} - V_{str}.$$

  Since $V_{str} < \mathrm{Num}$, we have:

  $$|T_n(x, q, w)| > 2 \cdot \mathrm{Num} - \mathrm{Num} = \mathrm{Num} \geq B.$$

Thus, in all cases, $|T_n(x, q, w)| \geq B$.

Proof of Property 2: High Count implies Low Complexity

We must show that if the relaxed extension count is large, i.e.,

$$|T_n^{\sqrt{d}}(x, q, w)| \geq B,$$

then the complexity is bounded by:

$$\mathrm{C}_V(x) \leq q + \frac{d}{2}.$$

Strategy. To prove that $\mathrm{C}_V(x) \le q + d/2$, it suffices to establish the following lower bound on the volume at scale q:

$$|T_q(x)| \ge 2^{\mathcal{Q}-q-O(\sqrt{d})}. \tag{7}$$

Indeed, applying the **Measure Density Lemma** (Lemma 7.1) and the **Coding Theorem** (Theorem 2.3), this volume bound implies:

$$\mathrm{K}(x \mid q) \le q + O(\sqrt{d}).$$

Thus:

$$\mathrm{K}(x \mid q + d/3) \le q + O(\sqrt{d}).$$

For sufficiently large d, the term $O(\sqrt{d})$ is strictly less than $d/3$. Therefore:

$$\mathrm{K}(x \mid q + d/3) \le q + d/3.$$

By Corollary 2.8, this implies:

$$\mathrm{C}_V(x) \le q + d/3 + O(1).$$

Finally, for sufficiently large d, we obtain the required bound

$$\mathrm{C}_V(x) \le q + d/2.$$

We consider the execution path of Algorithm $\mathcal{A}$ that determined the value B.

Case 1: Trivial Termination at Step 1 (High Past Volume at q). Suppose the algorithm terminated at Step 1. This implies the check passed:

$$|\mathrm{St}_q^w(x)| \ge 2^{-q-\lambda} \cdot 2^{\mathcal{Q}}.$$

Since the structure set is a subset of the total set ($\mathrm{St}_q^w(x) \subseteq T_q(x)$), we immediately have:

$$|T_q(x)| \ge |\mathrm{St}_q^w(x)| \ge 2^{\mathcal{Q}-q-\lambda}.$$

Since λ is a constant and d is sufficiently large, condition (7) is satisfied.

Case 2: Trivial Termination at Step 4 (High Structure Volume). Suppose the algorithm terminated at Step 4 because the condition $|\mathrm{St}_n^w(\tilde{q}, x)| \ge \mathrm{Num}$ was met. In this case $B = 0$. Although the condition $|T_n^{\sqrt{d}}| \ge B$ holds trivially, the state of the algorithm (large structure volume) is sufficient to upper bound the complexity $\mathrm{K}(\tilde{q}, x \mid n)$.

We start with the condition:

$$|\mathrm{St}_n^w(\tilde{q}, x)| \ge \mathrm{Num} = 2^{\mathcal{N}-q+\Delta-\lambda}.$$

We apply **Item 5** of Lemma 7.8 (Complexity Decomposition). Since the maximum is at least the first term:

$$\mathrm{K}(\tilde{q}, x \mid n) = \mathcal{N} - \log \max\left(|\mathrm{St}_n^w(\tilde{q}, x)|, |T_n(x, q, w)|\right) + O(1).$$

$$\mathrm{K}(\tilde{q}, x \mid n) \le \mathcal{N} - \log |\mathrm{St}_n^w(\tilde{q}, x)| + O(1).$$

Substituting the lower bound $|\mathrm{St}_n^w(\tilde{q}, x)| \ge \mathrm{Num}$:

$$\mathrm{K}(\tilde{q}, x \mid n) \le \mathcal{N} - \log(\mathrm{Num}) + O(1).$$

Substituting $\mathrm{Num} = 2^{\mathcal{N}-q+\Delta-\lambda}$:

$$\mathrm{K}(\tilde{q}, x \mid n) \le q - \Delta + O(\lambda). \tag{8}$$

(Note that $O(\lambda) \subset O(\sqrt{d})$ for large d).

Now we transfer this bound to the volume at scale q. We use **Item 3** of Lemma 7.8:

$$-\log |T_q(x, \tilde{q}, \alpha)| = \Delta + \mathrm{K}(\tilde{q}, x \mid n) - \mathcal{Q} + O(1).$$

Substituting the upper bound from (8):

$$-\log |T_q(x, \tilde{q}, \alpha)| \leq q - \mathcal{Q} + O(\sqrt{d}).$$

So:

$$|T_q(x, \tilde{q}, \alpha)| \geq 2^{\mathcal{Q}-q-O(\sqrt{d})}.$$

By using Corollary 7.3 (dropping the simple conditions $\tilde{q}, \alpha$), we conclude:

$$|T_q(x)| \geq |T_q(x, \tilde{q}, \alpha)| \cdot 2^{-O(1)} \geq 2^{\mathcal{Q}-q-O(\sqrt{d})}.$$

Case 3: Main Case (Step 4, Non-trivial). Suppose the algorithm computed the non-trivial threshold B in the 'Otherwise' branch of Step 4. This implies that $B = \mathrm{Num}$. Assume the premise of Property 2 holds: $|T_n^{\sqrt{d}}(x, q, w)| \geq B$. We derive the same complexity bound as in Case 2, but relying on the volume of extensions.

First, by Lemma 7.1 (Item 2), the strict count is related to the relaxed count as:

$$|T_n(x, q, w)| \geq |T_n^{\sqrt{d}}(x, q, w)| \cdot 2^{-O(\sqrt{d})} \geq \mathrm{Num} \cdot 2^{-O(\sqrt{d})}.$$

Next, we again apply **Item 5** of Lemma 7.8. The complexity is bounded by the logarithm of the maximum volume. In this case, we use the fact that the maximum is at least the second term:

$$\mathrm{K}(\tilde{q}, x \mid n) = \mathcal{N} - \log \max \left(|\mathrm{St}_n^w(\tilde{q}, x)|, |T_n(x, q, w)|\right) + O(1).$$

$$\mathrm{K}(\tilde{q}, x \mid n) \leq \mathcal{N} - \log |T_n(x, q, w)| + O(1).$$

Substituting the strict volume bound derived above:

$$\mathrm{K}(\tilde{q}, x \mid n) \leq \mathcal{N} - \log(\mathrm{Num} \cdot 2^{-O(\sqrt{d})}) + O(1).$$

Substituting Num:

$$\mathrm{K}(\tilde{q}, x \mid n) \leq \mathcal{N} - (\mathcal{N} - q + \Delta - \lambda) + O(\sqrt{d}).$$

Simplifying:

$$\mathrm{K}(\tilde{q}, x \mid n) \leq q - \Delta + O(\sqrt{d}).$$

The rest of the derivation (transferring to scale q) is identical to Case 2, leading to the required bound on $|T_q(x)|$.

7.9 Proof of the $\tilde{q}$ Structure Lemma

Proof. We search for $\tilde{q}$ among pairs of the form $T_l = \langle q, l \rangle$, where l is an integer in the range $0 \leq l \leq q$. For any such candidate T_l, we define its structural complexity components relative to the scales n and q:

$$\alpha(l) = \mathrm{K}(T_l \mid n),$$
$$\beta(l) = \mathrm{K}(T_l, \alpha(l) \mid q).$$

Our goal is to find an index l^* such that $\beta(l^*) = l^* + O(1)$. Note that for every l it holds that $\beta(l+1) = \beta(l) + O(1)$.

Lemma 7.9. *There exists an integer $l^* \in [0, q-1]$ such that:*

$$\beta(l^*) = l^* + O(1).$$

Proof. Consider the set of indices where the complexity exceeds the index value:

$$S = \{l \in [0, q] : \beta(l) \geq l\}.$$

1. Boundary Conditions.

- **Lower Bound ($l = 0$):** The value $\beta(0) = \mathrm{K}(\langle q, 0\rangle, \dots \mid q)$ is non-negative. Thus, $\beta(0) \geq 0$, so $0 \in S$, and the set S is non-empty.
- **Upper Bound ($l = q$):** The value $\beta(q) = \mathrm{K}(\langle q, q\rangle, \dots \mid q) = O(\log q)$. Since q is large, $\beta(q) < q$. Thus, $q \notin S$.

2. The Max Element. Define:

$$l^* = \max S.$$

By the definition of the maximum:

1. Since $l^* \in S$, we have $\beta(l^*) \geq l^*$.
2. Since l^* is the maximum, the next integer $l^* + 1$ is not in S. Therefore, $\beta(l^* + 1) < l^* + 1$.

Since $\beta(l^* + 1) = \beta(l^*) + O(1)$, we have:

$$l^* \leq \beta(l^*) \leq \beta(l^* + 1) + O(1) < l^* + 1 + O(1).$$

This implies $\beta(l^*) = l^* + O(1)$. □

We define our target string as $\tilde{q} = T_{l^*} = \langle q, l^*\rangle$.

Verification of Properties

We now verify that this choice of $\tilde{q}$ satisfies the three conditions of the Lemma.

1. Small Length. Since $\tilde{q} = \langle q, l^*\rangle$ and $l^* \leq q$, the string is composed of binary representations of two numbers bounded by q.

$$|\tilde{q}| = O(\log q).$$

2. Encodes q. Since $\tilde{q} = \langle q, l^*\rangle$, the value q is explicitly the first element of the pair.

$$\mathrm{C}_V(q \mid \tilde{q}) = O(1).$$

3. Complexity Decomposition. We must prove:

$$\mathrm{K}(\tilde{q} \mid n) = \mathrm{K}\Big(\tilde{q},\ \mathrm{K}(\tilde{q} \mid n),\ \mathrm{K}\big(\tilde{q}, \mathrm{K}(\tilde{q} \mid n) \mid q\big) \ \Big|\ n\Big) + O(1).$$

Let $\alpha = \mathrm{K}(\tilde{q} \mid n)$ and $\beta = \beta(l^*) = \mathrm{K}(\tilde{q}, \alpha \mid q)$. The RHS becomes $\mathrm{K}(\tilde{q}, \alpha, \beta \mid n)$.

By Lemma 7.9, we have $|\beta - l^*| \leq O(1)$. Since $\tilde{q} = \langle q, l^*\rangle$, the value l^* is contained in $\tilde{q}$. Therefore, given $\tilde{q}$, the value β can be described using only $O(1)$ bits. This implies that adding β to the condition provides no new information up to a constant:

$$\mathrm{K}(\tilde{q}, \alpha, \beta \mid n) = \mathrm{K}(\tilde{q}, \alpha \mid n) + O(1).$$

Now we apply Proposition 2.6 (Complexity Self-Sufficiency), which states $\mathrm{K}(x, \mathrm{K}(x \mid y) \mid y) = \mathrm{K}(x \mid y) + O(1)$. With $x = \tilde{q}$ and $y = n$, this yields:

$$\mathrm{K}(\tilde{q}, \alpha \mid n) = \mathrm{K}(\tilde{q} \mid n) + O(1).$$

Combining these steps proves the decomposition identity.

Computability of Algorithm $\mathcal{B}$

Algorithm $\mathcal{B}$ operates by iterating l from 0 to q, computing the values $\alpha(l)$ and $\beta(l)$ at each step, and halting when the condition of Lemma 7.9 is met.

The validity of this procedure relies on the computability of the prefix complexities involved. For any candidate $T_l = \langle q, l \rangle$, the length is small:

$$|T_l| = O(\log q) \ll n.$$

Therefore, the lengths of the inputs satisfy the condition of **Proposition 7.6**. This allows the algorithm to compute the exact values of $\alpha(l)$ and $\beta(l)$ by simulating the universal machine with the time bound T_w derived from the input w. Since the existence of l^* is guaranteed, the algorithm always halts. □

References

[1] Eric Allender, Parting Thoughts and Parting Shots (Read On for Details on How to Win Valuable Prizes!), Guest Piece for the Complexity Theory Column, edited by Lane Hemaspaandra, SIGACT NEWS 54 March, 2023, pp. 63–81, https://people.cs.rutgers.edu/~allender/papers/sigact.news.draft.pdf.

[2] Allender, E., Buhrman, H., Koucký, M.: What can be efficiently reduced to the Kolmogorov-random strings? *Annals of Pure and Applied Logic* 138, 2–19 (2006)

[3] E. Allender, H. Buhrman, M. Koucký, D. van Melkebeek, and D. Ronneburger. Power from random strings. *SIAM Journal on Computing*, 35:1467–1493, 2006.

[4] E. Allender, L. Friedman, and W. Gasarch. Limits on the computational power of random strings. *Information and Computation*, 222:80-92, 2013.

[5] M. Cai, R. Downey, R. Epstein, S. Lempp, and J. Miller. Random strings and tt-degrees of Turing complete c.e. sets. *Logical Methods in Computer Science*, Volume 10, Issue 3 (September 10, 2014) lmcs:1126; doi:10.2168/LMCS-10(3:15)2014

[6] Shuichi Hirahara, Unexpected hardness results for Kolmogorov complexity under uniform reductions, Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing (STOC 2020), 1038–1051.

[7] M. Kummer. On the complexity of random strings. In Symposium on Theoretical Aspects of Computer Science (STACS), volume 1046 of Lecture Notes in Computer Science, pages 25-36. Springer, 1996.

[8] Li M., Vitányi P., *An Introduction to Kolmogorov complexity and its applications*, 3rd ed., Springer, 2008 (1 ed., 1993; 2 ed., 1997), xxiii+790 pp. ISBN 978-0-387-49820-1.

[9] Alexey Milovanov, Some Games on Turing Machines and Power from Random Strings, Lecture Notes in Computer Science, Unity of Logic and Computation, 2023, 105-119, Springer Nature Switzerland.

[10] Alexey Milovanov, On the Computational Power of C-Random Strings. *CiE 2025*: 321–332.

[11] Andrej A. Muchnik and Semen Ye. Positselsky. Kolmogorov entropy in the context of computability theory. Theoretical Computer Science, 271:15-35, 2002.

[12] Shen, A., Uspensky V. and Vereshchagin N.: *Kolmogorov Complexity and Algorithmic Randomness*, ACM, (2017).

ECCC ISSN 1433-8092

https://eccc.weizmann.ac.il