CHAPTER 1

Plain Kolmogorov complexity

1.1. Definition and main properties

Let us recall the definition of Kolmogorov complexity from the introduction. This version of complexity was defined by Kolmogorov in his seminal paper [**78**]. In order to distinguish it from later versions we call it the *plain* Kolmogorov complexity. Later, starting from Chapter 4, we will also consider other versions of Kolmogorov complexity, including prefix versions and monotone versions, but for now by Kolmogorov complexity we always mean the plain version.

Recall that a *description mode*, or a *decompressor*, is a partial computable function D from the set of all binary strings Ξ into Ξ. A partial function D is *computable* if there is an algorithm that terminates and returns $D(x)$ on every input x in the domain of D and does not terminate on all other inputs. We say that y is a *description* of x *with respect to* D if $D(y) = x$.

The complexity of a string x *with respect to description mode* D is defined as

$$C_D(x) = \min\{l(y) \mid D(y) = x\}.$$

The minimum of the empty set is $+\infty$.

We say that a description mode D_1 *is not worse* than a description mode D_2 if there is a constant c such that $C_{D_1}(x) \leqslant C_{D_2}(x) + c$ for all x and write this as $C_{D_1}(x) \leqslant C_{D_2}(x) + O(1)$.

A description mode is called *optimal* if it is not worse than any other description mode. By the Solomonoff–Kolmogorov universality theorem (Theorem 1, p. 3) optimal description modes exist. Let us briefly recall its proof. Let U be an interpreter of a universal programming language, that is, $U(p, x)$ is the output of the program p on input x. We assume that programs and inputs are binary strings. Let

$$D(\hat{p}x) = U(p, x).$$

Here $p \mapsto \hat{p}$ stands for any computable mapping having the following property: given $\hat{p}$ we can effectively find p and also the place where $\hat{p}$ ends (in particular, if $\hat{p}$ is a prefix of $\hat{q}$, then $p = q$). This property implies that D is well defined. For any description mode D' let p be a program of D'. Then

$$C_D(x) \leqslant C_{D'}(x) + l(\hat{p}).$$

Indeed, for every description y of x with respect to D' the string $\hat{p}y$ is a description of x with respect to D.

Fix any optimal description mode D, and let $C(x)$ (we drop the subscript) denote the complexity of x with respect to D. (As we mentioned, in the first paper of Kolmogorov [**78**] the letter K was used, while in his second paper [**79**] the letter H was used. We follow here the notation used by Li and Vitányi [**103**].)

As the optimal description mode is not worse than the identity function $x \mapsto x$, we obtain the inequality $C(x) \leqslant l(x) + O(1)$ (Theorem 2, p. 5).

Let A be a partial computable function. Comparing the optimal description mode D with the description mode $y \mapsto A(D(y))$, we conclude that

$$C(A(x)) \leqslant C(x) + O(1),$$

showing the non-growth of complexity under algorithmic transformations (Theorem 3, p. 5).

Using this inequality, we can define Kolmogorov complexity of other finite objects, such as natural numbers, graphs, permutations, finite sets of strings, etc., that can be naturally encoded by binary strings.

For example, let us define the complexity of natural numbers. A natural number n can be written in binary notation. Another way to represent a number by a string is as follows. Enumerate all the binary strings in lexicographical order

$$\Lambda, 0, 1, 00, 01, 10, 11, 000, 001, 010, 011, 100, \ldots$$

using the natural numbers $0, 1, 2, 3, \ldots$ as indexes. This enumeration is more convenient compared to binary representation as it is a bijection. Every string can be considered as an encoding of its index in this enumeration. Finally, one can also encode a natural number n by a string consisting of n ones.

Using either of these three encodings, we can define the complexity of n as the complexity of the string encoding n. Three resulting complexities of n differ at most by an additive constant. Indeed, for every pair of these encodings there is an algorithm translating the first encoding into the second one. Applying this algorithm, we increase the complexity at most by a constant. Note that the Kolmogorov complexity of binary strings is defined up to an additive constant, so the choice of a computable encoding does not matter.

As the length of the binary representation of a natural number n is equal to $\log n + O(1)$, the Kolmogorov complexity of n is at most $\log n + O(1)$. (By log we denote binary logarithms.)

Here is another application of the non-growth of complexity under algorithmic transformations. Let us show that deleting the last bit of a string changes its complexity at most by a constant. Indeed, all three functions $x \mapsto x0$, $x \mapsto x1$, $x \mapsto$ (x without the last bit) are computable.

The same is true for the first bit. However this does not apply to every bit of the string. To show this, consider the string x consisting of 2^n zeros; its complexity is at most $C(n) + O(1) \leqslant \log n + O(1)$. (By log we always mean binary logarithm.) There are 2^n different strings obtained from x by flipping one bit. At least one of them has complexity n or more. (Recall that the number of strings of complexity less than n does not exceed the number of descriptions of length less than n, which is less than 2^n; see Theorem 5, p. 8.)

Incrementing a natural number n by 1 changes $C(n)$ at most by a constant. This implies that $C(n)$ satisfies the *Lipschitz property*: for some c and for all m, n, we have $|C(m) - C(n)| \leqslant c|m - n|$.

1 Prove a stronger inequality: $|C(m) - C(n)| \leqslant |m - n| + c$ for some c and for all $m, n \in \mathbb{N}$, and, moreover, $|C(m) - C(n)| \leqslant 2 \log |m - n| + c$ (the latter inequality assumes that $m \neq n$).

Several times we have used the upper bound 2^n for the number of strings x with $C(x) < n$. Note that, in contrast to other bounds, it involves no constants. Nevertheless this bound has a hidden dependence on the choice of the optimal description mode: if we switch to another optimal description mode, the set of strings x such that $C(x) < n$ can change!

2 Show that the number of strings of complexity less than n is in the range $[2^{n-c}; 2^n]$ for some constant c for all n.

(*Hint*: The upper bound 2^n is proved in the introduction, the lower bound is implied by the inequality $C(x) \leqslant l(x)+c$: the complexity of all the strings of length less than $n-c$ is less than n.)

Show that the number of strings of complexity *exactly* n does not exceed 2^n but can be much less: e.g., it is possible that this set is empty for infinitely many n.

(*Hint*: Change an optimal description mode by adding 0 or 11 to each description so that all descriptions have even length.)

3 Prove that the average complexity of strings of length n is equal to $n+O(1)$.

(*Hint*: Let α_k denote the fraction of strings of complexity $n-k$ among strings of length n. Then the average complexity is by $\sum_k k\alpha_k$ less than n. Use the inequality $\alpha_k \leqslant 2^{-k}$ and the convergence of the series $\sum k/2^k$.)

In the next statement we establish a formal relation between upper bounds of complexity and upper bounds of cardinality.

THEOREM 7. (a) *The family of sets* $S_n = \{x \mid C(x) < n\}$ *is enumerable, and* $|S_n| < 2^n$ *for all* n. *Here* $|S_n|$ *denotes the cardinality of* S_n.

(b) *If* V_n $(n = 0, 1, \dots)$ *is an enumerable family of sets of strings and* $|V_n| < 2^n$ *for all* n, *then there exists* c *such that* $C(x) < n + c$ *for all* n *and all* $x \in V_n$.

In this theorem we use the notion of an enumerable family of sets. It is defined as follows. A set of strings (or natural numbers, or other finite objects) is *enumerable* (= *computably enumerable* = *recursively enumerable*) if there is an algorithm generating all elements of this set in some order. This means that there is a program that never terminates and prints all the elements of the set in some order. The intervals between printing elements can be arbitrarily large; if the set is finite, the program can print nothing after some time (unknown to the observer). Repetitions are allowed, but this does not matter since we can filter the output and delete the elements that have already been printed.

For example, the set of all n such that the decimal expansion of $\sqrt{2}$ has exactly n consecutive nines is enumerable. The following algorithm generates the set: compute decimal digits of $\sqrt{2}$ starting with the most significant ones. Once a sequence of consecutive n nines surrounded by non-nines is found, print n and continue.

A family of sets V_n is called enumerable if the set of pairs $\{\langle n, x\rangle \mid x \in V_n\}$ is enumerable. This implies that each of the sets V_n is enumerable. Indeed, to generate elements of the set V_n for a fixed n, we run the algorithm enumerating the set $\{\langle n, x\rangle \mid x \in V_n\}$ and print the second components of all the pairs that have n as the first component. However, the converse statement is not true. For instance, assume that V_n is finite for every n. Then every V_n is enumerable, but at the same time it may happen that the set $\{\langle n, x\rangle \mid x \in V_n\}$ is not enumerable (say $V_n = \{0\}$ if $n \in S$ and $V_n = \varnothing$ otherwise, where S is any non-enumerable set of integers). One can verify that a family is enumerable if and only if there is an algorithm that

given any n finds a program generating V_n. A detailed study of enumerable sets can be found in every textbook on computability theory, for instance, in [**184**].

PROOF. Let us prove the theorem. First, we need to show that the set

$$\{\langle n, x\rangle \mid x \in S_n\} = \{\langle n, x\rangle \mid C(x) < n\},$$

where n is a natural number and x is a binary string, is enumerable.

Let D be the optimal decompressor used in the definition of C. Perform in parallel the computations of D on all the inputs. (Say, for $k = 1, 2, \dots$ we make k steps of D on k first inputs.) If we find that D halts on some y and returns x, the generating algorithm outputs the pair $\langle l(y) + 1, x\rangle$. Indeed, this implies that the complexity of x is less than $l(y) + 1$, as y is a description of x. Also it outputs all the pairs $\langle l(y) + 2, x\rangle, \langle l(y) + 3, x\rangle \cdots$ in parallel to the printing of other pairs.

For those familiar with computability theory, this proof can be compressed to one line:

$$C(x) < n \Leftrightarrow \exists y\, (l(y) < n \wedge D(y) = x).$$

(The set of pairs $\langle x, y\rangle$ such that $D(y) = x$ is enumerable, being the graph of a computable function. The operations of intersection and projection preserve enumerability.)

The converse implication is a bit harder. Assume that V_n is an enumerable family of finite sets of strings and $|V_n| < 2^n$. Fix an algorithm generating the set $\{\langle n, x\rangle \mid x \in V_n\}$. Consider the description mode D_V that deals with strings of length n in the following way. Strings of length n are used as descriptions of strings in V_n. More specifically, let x_k be the kth string in V_n in the order the pairs $\langle n, x\rangle$ appear while generating the set $\{\langle n, x\rangle \mid x \in V_n\}$. (We assume there are no repetitions, so $x_0, x_1, x_2, \dots$ are distinct.) Let y_k be the kth string of length n in lexicographical order. Then y_k is a description of x_k, that is, $D(y_k) = x_k$. As $|V_n| < 2^n$, every string in V_n gets a description of length n with respect to D.

We need to verify that the description mode D_V defined in this way is computable. To compute $D_V(y)$, we find the index k of y in the lexicographical ordering of strings of length $l(y)$. Then we run the algorithm generating pairs $\langle n, x\rangle$ such that $x \in V_n$ and wait until k different pairs having the first component $l(y)$ appear. The second component of the last of them is $D_V(y)$.

By construction, for all $x \in V_n$ we have $C_{D_V}(x) \leqslant n$. Comparing D_V with the optimal description mode, we see that there is a constant c such that $C(x) < n + c$ for all $x \in V_n$. Theorem 7 is proven. □

The intuitive meaning of Theorem 7 is as follows. The assertions "the number of strings with a certain property is small" (is less than 2^i) and "all the strings with a certain property are simple" (have complexity less than i) are equivalent provided the property under consideration is enumerable and provided the complexity is measured up to an additive constant (and the number of elements is measured up to a multiplicative constant).

Theorem 7 can be reformulated as follows. Let $f(x)$ be a function defined on all binary strings and which takes as values natural numbers and a special value $+\infty$. We call f *upper semicomputable*, or *enumerable from above*, if there is a computable function $\langle x, k\rangle \mapsto F(x, k)$ defined on all strings x and all natural numbers k such that

$$F(x, 0) \geqslant F(x, 1) \geqslant F(x, 2) \geqslant \cdots$$

and

$$f(x) = \lim_{k\to\infty} F(x,k)$$

for all x. The values of F are natural numbers as well as the special constant $+\infty$. The requirements imply that for every k the value $F(x,k)$ is an upper bound of $f(x)$. This upper bound becomes more precise as k increases. For every x there is a k for which this upper bound is tight. However, we do not know the value of that k. (If there is an algorithm that given any x finds such k, then the function f is computable.) Evidently, any computable function is upper semicomputable.

A function f is upper semicomputable if and only if the set

$$G_f = \{\langle x,n\rangle \mid f(x) < n\}$$

is enumerable. This set is sometimes called the "upper graph of f", which explains the strange names "upper semicomputable" and "enumerable from above".

Let us verify this. Assume that a function f is upper semicomputable. Let $F(x,k)$ be the function from the definition of semicomputability. Then we have

$$f(x) < n \Leftrightarrow \exists k\, F(x,k) < n.$$

Thus, performing in parallel the computations of $F(x,k)$ for all x and k, we can generate all the pairs in the upper graph of f.

Assume now that the set G_f is enumerable. Fix an algorithm enumerating this set. Then define $F(x,k)$ as the best upper bound of f obtained after k steps of generating elements in G_f. That is, $F(x,k)$ is equal to the minimal n such that the pair $\langle x,n+1\rangle$ has been printed after k steps. If there is no such pair, let $F(x,k) = +\infty$.

Using the notion of an upper semicomputable function, we can reformulate Theorem 7 as follows.

THEOREM 8. (a) *The function C is upper semicomputable and*

$$|\{x \mid C(x) < n\}| < 2^n$$

for all n.

(b) *If a function C' is upper semicomputable and $|\{x \mid C'(x) < n\}| < 2^n$ for all n, then $C(x) \leqslant C'(x) + c$ for some c and for all x.*

Note that the upper bound 2^n of the cardinality of $|\{x \mid C'(x) < n\}|$ in item (b) can be replaced by a weaker upper bound $O(2^n)$.

Theorem 8 allows us to define Kolmogorov complexity as a minimal (up to an additive constant) upper semicomputable function k that satisfies the inequality

$$|\{x \mid k(x) < n\}| \leqslant O(2^n).$$

One can replace the requirement of minimality in this definition by some other properties of C. In this way we obtain the following axiomatic definition of Kolmogorov complexity [**173**]:

THEOREM 9. *Let k be a natural-valued function defined on binary strings. Assume that k satisfies the following properties:*

(a) *k is upper semicomputable* (enumerability axiom)*;*

(b) *for every partial computable function A from Ξ to Ξ, the inequality*

$$k(A(x)) \leqslant k(x) + c$$

is valid for some c and all x in the domain of A (the axiom guarantees that complexity does not increase)*;*

(c) *the number of strings x such that $k(x) < n$ is in the range $[2^{n-c_1}; 2^{n+c_2}]$ for some c_1, c_2 and for any n* (calibration axiom).

Then $k(x) = C(x) + O(1)$, that is, the difference $|k(x) - C(x)|$ is bounded by a constant.

PROOF. Theorem 8 implies that $C(x) \leqslant k(x) + O(1)$. So we need to prove that

$$k(x) \leqslant C(x) + O(1).$$

LEMMA 1. *There is a constant c and a computable sequence of finite sets of binary strings*

$$M_0 \subset M_1 \subset M_2 \subset \cdots$$

with the following properties: the set M_i has exactly 2^i strings and $k(x) \leqslant i + c$ for all $x \in M_i$ and all i.

Computability of $M_0, M_1, M_2, \ldots$ means that there is an algorithm that given any i computes the list of elements of M_i.

PROOF. By axiom (c) there exists a constant c such that for all i the set

$$A_i = \{x \mid k(x) < i + c\}$$

has at least 2^i elements. By item (a) the family A_i is enumerable. Remove from A_i all the elements except 2^i strings generated first. Let B_i denote the resulting set. The list of the elements of B_i can be found given i: we wait until the first 2^i strings are generated. The set B_i is not necessarily included in B_{i+1}. To fix this we define M_i inductively. We let $M_0 = B_0$, and we let M_{i+1} be equal to M_i plus any 2^i elements of B_{i+1} that are outside M_i. Lemma 1 is proven.

LEMMA 2. *There is a constant c such that $k(x) \leqslant l(x) + c$ for all x (recall that $l(x)$ denotes the length of x).*

PROOF. Let $M_0, M_1, M_2, \ldots$ be the sequence of sets from the previous lemma. There is a computable one-to-one function A defined on the union of all M_i that maps $M_{i+1} \setminus M_i$ onto the set of binary strings of length i. (Recall that the set $M_{i+1} \setminus M_i$ has exactly 2^i strings.) By item (b) we have $k(A(y)) \leqslant k(y) + c'$ for some c' and all x. For all x of length i there is $y \in M_{i+1} \setminus M_i$ such that $A(y) = x$, hence $k(x) \leqslant k(y) + c' \leqslant i + c$ for some c and all i. Lemma 2 is proven.

Let us finish the proof of the theorem. Let D be the optimal description mode, and let p be a shortest description of x with respect to D. Then

$$k(x) = k(D(p)) \leqslant k(p) + O(1) \leqslant l(p) + O(1) = C(x) + O(1).$$

Note that we have used property (b) twice: in the proof of Lemma 2 and just now. □

4 Assume that strings over the alphabet $\{0, 1, 2, 3\}$ are used as descriptions. Prove that in this case the Kolmogorov complexity, defined as the length of the shortest description (with respect to an optimal description mode), is equal to half of the regular complexity (up to an additive constant).

5 (Continued) Formulate and prove a similar statement for the n-letter alphabet.

6 Assume that $f\colon \mathbb{N} \to \mathbb{N}$ is a total computable increasing function and

$$\liminf f(n+1)/f(n) > 1.$$

Let A_n be an enumerable family of finite sets such that $|A_n| \leqslant f(n)$ for all n. Prove that there is a constant c such that $C(x) \leqslant \log f(n) + c$ for all n and all $x \in A_n$.

7 Prove that for some constant c and for every n the following holds. For every string x of length n one can flip a bit in x so that the resulting string y satisfies the inequality $C(y) \leqslant n - \log n + c$.

(*Hint*: For a given natural k consider a Boolean matrix of size $k \times (2^k - 1)$ whose columns are all non-zero strings of length k. (Such matrix is used for Hamming codes.) Consider the linear mapping $\mathbb{B}^{2^k-1} \to \mathbb{B}^k$ defined by this matrix, where $\mathbb{B}$ denotes the field $\{0, 1\}$. It is easy to verify that for every vector x one can flip one bit in x so that the resulting string y is in the kernel of this mapping, and the elements of the kernel have complexity at most $2^k - k + O(1)$. This gives the desired result for $n = 2^k - 1$; if n does not have the form $2^k - 1$, we can flip one of the first $2^k - 1$ bits for an appropriate k.)

1.2. Algorithmic properties

The function C is upper semicomputable. On the other hand, it is not computable and, moreover, it has no unbounded computable lower bounds (Theorem 6, p. 9).

This implies that all optimal description modes are necessarily non-total, that is, some strings describe nothing. Indeed, if a description mode D is total, then we can compute $C_D(x)$ just by trying all descriptions in lexicographical order until we find the shortest one.

At first glance, this contradicts to our intuition: the bigger the domain of D, the better D is. If the optimal decompressor D is undefined on some string y, then we can define another description mode D' as follows. Let $D'(y)$ be equal to a string z of complexity (with respect to D) greater than $l(y)$, and let D' coincide with D on all other strings. The description mode D' is a bit better than D, as the complexity of all strings except z remains the same while the complexity of z has been decreased.

There is no formal contradiction here, as D is still not worse than D' (they differ only at one point, the difference between the complexities is bounded by a constant, and both D and D' are optimal). However, this is still a bit strange. This observation was made by Yu. Manin in his book *Computable and non-computable* [**114**] (by the way, in this book he also discussed the computational power of quantum mechanics long before quantum computing became fashionable).

A similar argument shows that the domain of every optimal description mode is undecidable. (The set of strings is called *decidable*, or *computable*, if there is an algorithm that for any given string decides whether it belongs to the set or not.) Indeed, if there were an algorithm deciding whether $D(x)$ is defined or not, then there would be a total computable extension of D (for example, let $D(x) = 0$ for all x outside the domain of D). This extension would be a total optimal description mode, but this is impossible as we have seen.

As a byproduct we get an algorithm whose domain is undecidable. This is one of the central theorems in computability theory (see, for example, [**184**]).

In general the notion of Kolmogorov complexity has a number of connections with computability theory. Recently, many interesting facts were discovered; see [**147, 49**]. We consider here only two basic examples (a simple set of simple strings and the complexity of large numbers).

1.2.1. Simple strings and simple sets. In this section, the word "simple" has two unrelated meanings. First, when applied to strings, it means that the Kolmogorov complexity of the string is small. Second, it is applied to sets of strings. The notion of a simple set was introduced by the American logician Emil Post and has no relation to Kolmogorov complexity.

Definition. An enumerable set A is *simple* (according to Post) if its complement is infinite but has no infinite enumerable subset.

Call a string x *simple* if $C(x) < l(x)/2$.

Theorem 10. *The set of all simple strings is simple in the sense of Post.*

Proof. That set S of all simple strings is enumerable. Indeed, the function C is upper semicomputable, and if $C(x)$ is less than $|x|/2$, this can be seen while approximating $C(x)$ from above.

The number of strings of complexity less than $n/2$ does not exceed $2^{n/2}$. Therefore the fraction of simple strings among strings of length n is negligible, and the complement of S is infinite.

Assume now that the complement of S has an infinite enumerable subset U. We can use U to obtain a computable unbounded lower bound of C. To find a string of complexity greater than t, we can generate elements of U until we find a string u_t of length greater than $2t$. As U is infinite, there is such a string. The complexity of u_t is greater than t; otherwise, u_t is simple. Without loss of generality we can assume that the strings u_t, $t = 1, 2, \ldots$ are pairwise different. Thus the function $u_t \mapsto t$ is a computable unbounded lower bound for C. This contradicts to Theorem 6 (page 9). □

Note that the choice of the threshold $l(x)/2$ in the definition of a simple string was not essential. The proof of Theorem 10 would work as well with $l(x) - 1$ or $\log\log l(x)$ in place of $l(x)/2$.

1.2.2. Complexity of large numbers. Let us identify a natural number m with the binary string having index m in the standard enumeration of binary strings. In this way C becomes a function of a natural argument. The function $C(m)$ goes to infinity as $m \to \infty$. Indeed, for all n there are only finitely many integers of complexity less than n. However, the convergence is not effective. That is, there is no algorithm that, for every given n, finds a number N such that the complexity of N and of all larger numbers is bigger than n. Indeed, such an algorithm would provide an effective way to describe the number N, whose complexity is at least n, by $\log n + O(1)$ bits. We have seen this in the proof of Theorem 6 (p. 9).

In this section, we study in detail the rate of convergence of C to infinity. Following Chaitin [**31**], we consider for every natural n the largest number $B(n)$ whose complexity is at most n:

$$B(n) = \max\{m \in \mathbb{N} \mid C(m) \leqslant n\}.$$

The function $n \mapsto B(n)$ may be called the modulus of the convergence of $C(m)$ to infinity (see Figure 1). Indeed, $C(x) > n$ for all $x > B(n)$ (and $B(n)$ is the minimal

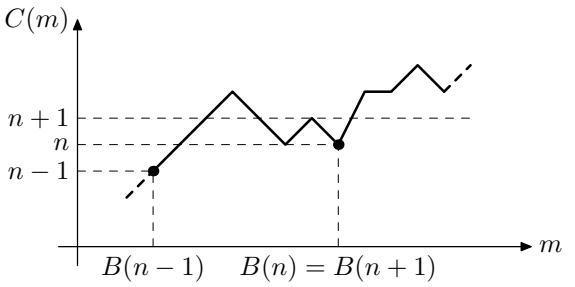

FIGURE 1. The definition of $B(n)$: the value $C(m)$ does not exceed $n-1$ for $m = B(n-1)$ (the case when $C(B(n-1)) = n-1$ is shown), and $C(m) \geqslant n$ for all $m > B(n-1)$. At the point $m = B(n)$, the value of C does not exceed n (the case when $C(B(n)) = n$ is shown), and $C(m) > n$ for all $m > B(n)$. The case when $C(m)$ is even greater than $n+1$ for all $m > B(n)$ is shown, thus $B(n+1) = B(n)$. For $m \in (B(n-1), B(n)]$, the value of the function $C_{\geqslant}(m)$ is equal to n.

number with this property). Note also that it can happen (for small values of n) that $C(m) > n$ for all m. In this case we let $B(n) = -1$.

The function B can be considered as an inverse function to the function

$$C_{\geqslant}(N) = \min\{C(m) \mid m \geqslant N\}.$$

The function $C_{\geqslant}$ grows very slowly. It takes the value n between $B(n-1)$ and $B(n)$, more precisely, on the interval $(B(n-1), B(n)]$. The slow increase of $C_{\geqslant}$ corresponds to the fast increase of B. The latter can be illustrated by the following result.

THEOREM 11. *Let f be a computable function from $\mathbb{N}$ to $\mathbb{N}$. Then $B(n) \geqslant f(n)$ for all but finitely many n.*

Note that f may be a partial function. In this case we claim that $B(n) \geqslant f(n)$ for all sufficiently large n that are in the domain of f.

PROOF. As algorithmic transformations do not increase complexity, for some constant c for all n we have

$$C(f(n)) \leqslant C(n) + O(1) \leqslant \log n + c.$$

On the other hand, the definition of B and the inequality $f(n) > B(n)$ imply $C(f(n)) > n$. Thus

$$n < C(f(n)) \leqslant \log n + c$$

whenever $f(n) > B(n)$. This can happen only for finitely many n. □

Let us reformulate the definition of $B(n)$ as follows. Let D be the optimal description mode used in the definition of Kolmogorov complexity. Then $B(n)$ is the maximal value of D on strings of length at most n:

$$B(n) = \max\{D(x) \mid l(x) \leqslant n\}.$$

Recall that we identify natural numbers and binary strings and consider the values of D as natural numbers. The minimum of the empty set is defined as -1.

Consider now any partial computable function $d\colon \Xi \to \mathbb{N}$ in place of D, and let

$$B_d(n) = \max\{d(x) \mid l(x) \leqslant n \text{ and } d(x) \text{ is defined}\}.$$

The next theorem shows that the function B is the largest function among all functions B_d in the following sense:

THEOREM 12. *For every function d there is a constant c such that*

$$B_d(n) \leqslant B(n+c)$$

for all n.

PROOF. For every x of length at most n, the complexity of $d(x)$ is less than $n+c$ for some constant c. Indeed, the complexity of $d(x)$ exceeds at most by a constant the complexity of x, which is less than $n + O(1)$. Hence $d(x)$ does not exceed the largest number of complexity $n+c$ or less, i.e., $B(n+c)$. □

This (trivial) observation is useful in the following special case. Let M be an algorithm, and let X be a set of binary strings. A *halting problem* for M restricted to X is the following problem: given a string $x \in X$, find out whether M terminates on x or not.

A classical result in computability theory states that for some algorithm M the unrestricted halting problem ($X = \Xi$) for M is undecidable.

We are interested now in the case when X is the set of all strings of bounded length. Fix some algorithm M and consider the running time $t(x)$ of M for some input x. If M does not halt on x, then $t(x)$ is undefined. Thus the domains of t and M coincide. By definition, $B_t(n)$ is the maximal running time of M on inputs of length at most n. If we know $B_t(n)$ or any larger number m, we can solve the halting problem for M and every input x of length at most n: Run M on input x; if the computation does not terminate after m steps, it never terminates.

We have seen that $B_t(n) \leqslant B(n+c)$ for some constant c (depending on M). Therefore, the knowledge of $B(n+c)$ or any greater number is enough to solve the halting problem of M on inputs of length at most n. In other words, the following holds:

THEOREM 13. *For every algorithm M there is a constant c and another algorithm A having the following property. For every n and for every number $t > B(n+c)$, the algorithm A, given n and t, produces the list of all strings x of length at most n such that M halts on input x.*

This theorem says that the halting problem for inputs of length at most n *is reducible* to the problem of finding a number greater than $B(n+c)$.

If M is the optimal decompressor D, then the converse is also true: given n and the list of all strings x of length at most n in the domain of D, we can find $B(n)$.

Continuing this argument, we can prove the following result:

THEOREM 14. *Let $BB(n)$ denote the maximal running time of the optimal decompressor D on strings of length at most n (in the domain of D). Then*

$$BB(n) \leqslant B(n+c) \quad \textit{and} \quad B(n) \leqslant BB(n+c)$$

for some c and all n.

PROOF. Let α_n be the most time-consuming description of length at most n, that is, the string x of length at most n in the domain of D that maximizes the running time of D on x. Knowing n and α_n, one can generate the list of all strings of length at most n in the domain of D, and hence the number $BB(n)$. Both n and α_n can be encoded in one string of length $n+1$, the string $0\cdots 01\alpha_n$ (there are $n-l(\alpha_n)$ zeros in the beginning). Therefore, the Kolmogorov complexity of $BB(n)$ is at most $n+O(1)$, and $BB(n) \leqslant B(n+c)$ for some c and all n.

Let us prove the second inequality of the theorem showing that every $t > BB(n)$ has complexity at least $n - O(1)$. Assume that t has a description u of length k; we need to show that $k > n - O(1)$. Knowing u and n, one can effectively obtain a string of complexity greater than n. Indeed, we reconstruct t (from u) and wait t steps for every description of size at most n. This gives us all strings of complexity at most n, and we can take some other string. By definition of $B(n)$ we conclude that the pair (u, n) has complexity at least $n - O(1)$. On the other hand, this pair can be described using $k + O(\log(n-k))$ bits if we join the self-delimited description of $n-k$ and u. Therefore, $k+O(\log(n-k)) \geqslant n-O(1)$, and $(n-k) - O(\log(n-k)) \leqslant O(1)$, hence $n - k \leqslant O(1)$. (We assumed that $n > k$; otherwise, there is nothing to prove.) □

This theorem shows that, within an additive constant in the argument, $B(n)$ is the maximal running time of the optimal decompressor on descriptions of length at most n. A similar function appeared in the literature under the name of "busy beaver function". It was introduced by T. Radó [**150**] and is defined usually as the maximal number of ones on the tape of Turing machine with n states and binary tape alphabet (1 and blank) after it terminates (starting with blank tape).

More generally, given n and any object from the following list, we can find any other object from the list for a little bit smaller value of n:

(a) the list of all strings of Kolmogorov complexity at most n with their Kolmogorov complexities;
(b) the number of such strings;
(c) $B(n)$;
(d) $BB(n)$;
(e) the list of all strings of length at most n in the domain of the optimal decompressor (the halting problem for the optimal decompressor restricted to inputs of length at most n);
(f) the number of such strings;
(g) the most time-consuming input of length at most n for the optimal decompressor;
(h) the graph T_n of the function $C(x)$ on strings x of length n;
(i) the lexicographically first string γ_n of length n with Kolmogorov complexity at least n (it exists since the number of strings of complexity less than n is less than 2^n).

More specifically, the following statement holds.

THEOREM 15. *The complexity of every object in* (a)–(i) *is equal to* $n + O(1)$. *These objects are equivalent to each other in the following sense: Let* X_n *and* Y_n *be objects described in two items among* (a)–(i). *Then there is a constant* c *and an algorithm that given* n *and* X_n *finds* Y_{n-c}.

PROOF. The equivalence of (d), (e), (f), and (g) is easy. Each of the objects (d), (e), (f), and (g) together with n determines the list of all terminating computations of the optimal decompressor D on strings of length at most n. Indeed, knowing $BB(n)$, we can run D on all inputs of length at most n for $BB(n)$ steps. Knowing (e), that is, the list of strings of length at most n in the domain of D, we can run D on all those strings until all the computations terminate (and we know that this happens). Knowing (f), the number of strings of length at most n on which D terminates, we run D on all strings of length at most n until the desired number of computations do terminate. Knowing the string (g), we run D on that string, count the number of steps t, and then run D on all other strings of length at most n for t steps.

Conversely, the list of all halting computations of the optimal decompressor D on strings of length at most n together with n identifies each of the objects (d)–(g) as well as the objects (a)–(c). Therefore, by transitivity (which is easy to check), all the objects (d)–(g) are equivalent.

Let us prove now that (a)–(c) are equivalent to each other and equivalent to (d)–(g). Given the list of strings of complexity at most n, we can find the number of them (so (a)→(b)) and the largest number of complexity at most n (so (a)→(c)).

It is not that easy to find (a) given (b) and n. Given n and the number of strings of complexity at most n, we can reconstruct the list of these strings (generating them until we obtain the desired number of strings) and find a maximal number among them ((b)→(c)). But we still do not know the Kolmogorov complexity of the generated strings. We will prove the implication (c)→(a) indirectly, by showing (c)→(d); we know already that (d) implies (a). This will prove that all objects (a)–(g) are equivalent.

The implication (c)→(d) follows from Theorem 14. We know that $B(n)$ is an upper bound for $BB(n-c)$ (for appropriate c). Thus, given n and $B(n)$, we can find $BB(n-c)$ as follows: run D on all inputs of length at most $n-c$ within $B(n)$ steps. Then find $BB(n-c)$ as the number of steps in the longest run.

It remains for us to consider the objects (h) and (i). The implication (a)→(h) is easy. Indeed, for some constant c the complexity of every string of length $n-c$ does not exceed n. If we know the list (a) and n, then removing all the strings of length different from $n-c$ from the list, we get (h) for $n-c$.

The conversion (h)→(i) is straightforward.

Thus it remains to prove (i)→(a). It is enough to show that, given the lexicographically first string γ_n of length n and complexity at least n, we can find $BB(n-O(1))$ or a number greater than $BB(n-O(1))$. This can be done as follows.

Given γ_n, find n, and for each string x of length n preceding γ_n in lexicographical order, find a description p_x of x that has length n or less, and find out the running time t_x of D on p_x. (Note that p_x may be not the shortest description of x.) Let T be the maximum of t_x for those x. We claim that $T > BB(n-c)$ for some c that does not depend on n. Assume that this inequality is false, that is, $T \leqslant BB(n-c)$. We will prove that then c is small. Consider the most time-consuming description α_{n-c} of length at most $n-c$; let $n-c-d$ be its length. Given α_{n-c} and $c+d$, we can find n and $BB(n-c)$. From this we can find γ_n: run D on all strings of length at most n within $BB(n-c)$ steps. Consider all the strings of length n for which we have found descriptions of length n or less. Then γ_n is the lexicographically

first remaining string (since $T \leqslant BB(n-c)$ according to our assumption). As the complexity of γ_n is at least n, we have $n \leqslant C(\gamma_n) \leqslant (n-c-d)+2\log(c+d)+O(1)$, hence $(c+d) = O(1)$.

We have thus proven the equivalence of objects (a)–(h). It remains to prove that the complexity of each of them is $n + O(1)$.

Let X_n be one of objects (a)–(h). We have just proven that X_n can be obtained from γ_{n+c} and n (actually, we do not need n, as $n = l(\gamma_{n+c}) - c$). Therefore, $C(X_n) \leqslant C(\gamma_{n+c}) + O(1) \leqslant n + O(1)$.

To prove the lower bound of $C(X_n)$, let $n - d$ be the complexity of X_n. For some constant c the string γ_{n-c} can be obtained from the shortest description of X_n of length $n - d$ and from d (note that n can be retrieved from the length of the shortest description and d). Thus, $n - c \leqslant C(\gamma_{n-c}) \leqslant (n-d) + 2\log d + O(1)$. Therefore, $d \leqslant 2\log d + c + O(1)$ and, hence, $d = O(1)$. □

8 The objects in Theorem 15 depend on the choice of the optimal decompressor. In the proof we assumed that the same optimal decompressor is used in all the items (a)–(h). Prove that the statement of the theorem remains true if different decompressors are used.

9 Prove that the complexity of all the objects in Theorem 15 becomes $O(\log n)$ if we relativize the definition of Kolmogorov complexity by $\mathbf{0}'$, that is, if we allow the decompressor to query the oracle for the halting problem.

We have seen that there exist a constant c and an algorithm A that, given the string γ_n, solves the halting problem for the optimal decompressor on inputs of length at most $n - c$. This means that given an "oracle" that finds γ_n for every given n, we can solve the halting problem. The same can be done given an oracle deciding whether a given string x is incompressible, that is, $C(x) \geqslant l(x)$. Indeed, using that oracle, we can find γ_n by probing all strings of length n.

Using the terminology of computability theory, we can say that the halting problem is *Turing reducible* to the set of incompressible strings (or its complement, the set of compressible strings). This implies that the halting problem is also reducible to the "upper graph" of C, that is, to the set $\{\langle x, k\rangle \mid C(x) < k\}$. Using the terminology of computability theory, we say that the set of compressible strings (as well as the upper graph of C) is *Turing complete* in the class of enumerable sets (this means that it is enumerable and that the halting problem is Turing reducible to it).

10 Find some upper bound for the number of oracle queries for the set

$$\{\langle x, k\rangle \mid C(x) < k\}$$

needed to solve the halting problem for a fixed machine M and for all strings of length at most n.

11 Let f be a computable partial function from $\mathbb{N}$ to $\mathbb{N}$. Prove that there is a constant c such that for all n, such that $f(B(n))$ is defined, the inequality $B(n+c) \geqslant f(B(n))$ is true.

(*Hint*: The complexity of $f(B(n))$ is at most $n + O(1)$.)

12 Call a set U *r-separable* [**137**] if every enumerable set V disjoint with U can be separated from U by a decidable set, that is, there is a decidable set R that includes V and is disjoint with U.

(a) Prove that the the set $\{\langle x, k\rangle \mid C(x) < k\}$ (the upper graph of C) is an r-separable set. The set of compressible strings is r-separable, too.

(*Hint*: Assume that the upper graph of C is disjoint with some enumerable set V. The set of the second components of pairs in V is finite, otherwise we get an unbounded computable lower bound for C. That is, V is included in a horizontal strip of finite height. The intersection of the strip with the upper graph is finite.)

(b) We say that a set U_1 is m-reducible to a set U_2 if there is a total computable function f such that $U_1 = f^{-1}(U_2)$. Prove that if U_2 is r-separable and U_1 is m-reducible to U_2, then U_1 is r-separable as well.

(*Hint*: If V is an enumerable set disjoint with U_1, then $f(V)$ is an enumerable set disjoint with U_2. If R is a decidable set separating $f(V)$ and U_2, then $f^{-1}(R)$ is a decidable set separating V and U_1.)

(c) Prove that there is an enumerable set that is not r-separable (such a set does not m-reduce to the upper graph of C).

(*Hint*: There is a pair of disjoint enumerable inseparable sets.)

13 Following [**74**], prove that the following problems are equivalent: "for a given integer n find some string of complexity at least n" and "for a given algorithm without input find some string that is different from its output" (if the algorithm does not terminate, any string is OK). An oracle that fulfills one of these tasks can be used to (effectively) fulfill the other.

(*Hint*: Given an algorithm, we can provide an upper bound for the complexity of its output—it is bounded by complexity (and therefore the length) of the algorithm itself. On the other hand, to provide a string of high complexity means to provide a string which is guaranteed to be different from the outputs of finitely many algorithms. At first, this looks like a more difficult task than for one algorithm (as the oracle does). However, the following trick helps: we may assume that the outputs are tuples and construct a tuple that differs from the output of ith algorithm in ith position.)

14 (Continued) Prove that both these problems are equivalent to the problem of computing a fixed-point free function: "for every algorithm construct another algorithm that computes a different function" (not the same as the first one).

15 (Continued) Prove that an enumerable oracle can solve these problems if and only if it solves the halting problem (M. Arslanov proved this result without using Kolmogorov complexity).

(*Hint*: Assume that an enumerable oracle A allows us to compute strings of arbitrarily high complexity. Then let us compute a string of complexity at least n using this oracle, and look at all elements of A that were questioned during this computation. How many steps are needed to enumerate all this elements? This is a big number: any T greater than this number, has Kolmogorov complexity of at least n, since T-approximation of A can be used instead of A. On the other hand, having an oracle for A, we can find T for a given n.)

Kolmogorov complexity and functions B and BB turn out to be useful in studying the so-called "generic" and "coarse" algorithms that solve the halting problem for most inputs (the fraction of errors converges to zero); see [**11**]. The versions of these functions based on prefix complexity were introduced by Gács [**57**]; see also [**4**] for recent results related to the busy beaver functions for different versions of Kolmogorov complexity.

We have shown only several (simple) examples that show how Kolmogorov complexity is related to computability theory (also called *recursion theory*). This area is now actively growing, so we refer the interested reader to two recent monographs [**147**] by A. Nies and [**49**] by R. Downey and D. Hirschfeldt.

Theorem 15 selects some very special objects among all objects of complexity n (in fact, one object up to equivalence is described above). At first glance, this seems strange: our intuition says that all random (incompressible) strings of length n should be indistinguishable, and any special property of a string could be used to compress it. However, we have found a very special random string γ_n of length n. This paradox can be explained as follows: the individual properties of γ_n do allow us to find a short description for γ_n, but we need the oracle for $\mathbf{0}'$ to decompress that description.

We will come back to this question in Section 5.7, which discusses "the number of wisdom" Ω, and in Section 14.3, which studies two-part descriptions.

Finally, let us note that although all the objects in Theorem 15 are equivalent, they have very different lengths. The lengths of (a), (b), (e)–(i) are about n while the length of (c) and (d) grows faster than every computable function of n.