What is this book about?

What is Kolmogorov complexity?

Roughly speaking, Kolmogorov complexity means "compressed size". Programs like `zip`, `gzip`, `bzip2`, `compress`, `rar`, `arj`, etc., compress a file (text, image, or some other data) into a presumably shorter one. The original file can then be restored by a "decompressing" program (sometimes both compression and decompression are performed by the same program). Note that we consider here only lossless compression.

A file that has a regular structure can be compressed significantly. Its compressed size is small compared to its length. On the other hand, a file without regularities can hardly be compressed, and its compressed size is close to its original size.

This explanation is very informal and contains several inaccuracies—both technical and more essential. First, instead of files (sequences of bytes) we will consider binary strings (finite sequences of bits, that is, of zeros and ones). The length of such a string is the number of symbols in it. (For example, the string 1001 has length 4, and the empty string has length 0.)

Here are the more essential points:

- We consider only decompressing programs; we do not worry at all about compression. More specifically, a *decompressor* is any algorithm (a program) that receives a binary string as an input and returns a binary string as an output. If a decompressor D on input x terminates and returns string y, we write $D(x) = y$ and say that x is a *description* of y with respect to D. Decompressors are also called *description modes*.
- A description mode is not required to be total. For some x, the computation $D(x)$ may never terminate and therefore produces no result. Also we do not put any constraints on the computation time of D: on some inputs the program D may halt only after an extremely long time.

Using recursion theory terminology, we say that a description mode is a partial computable (=partial recursive) function from Ξ to Ξ, where $\Xi = \{0,1\}^*$ stands for the set of all binary strings. Let us recall that we associate with every algorithm D (whose inputs and outputs are binary strings) a function d computed by D; namely, $d(x)$ is defined for a string x if and only if D halts on x, and $d(x)$ is the output of D on x. A partial function from Ξ to Ξ is called *computable* if it is associated with (=computed by) some algorithm D. Usually we use the same letter to denote the algorithm and the function it computes. So we write $D(x)$ instead of $d(x)$ unless it causes a confusion.

Assume that a description mode (a decompressor) D is fixed. (Recall that D is computable according to our definitions.) For a string x consider all its descriptions,

that is, all y such that $D(y)$ is defined and equals x. The length of the shortest string y among them is called the *Kolmogorov complexity* of x with respect to D:

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

Here $l(y)$ denotes the length of the string y; we use this notation throughout the book. The subscript D indicates that the definition depends on the choice of the description mode D. The minimum of the empty set is defined as $+\infty$, thus $C_D(x)$ is infinite for all the strings x outside the range of the function D (they have no descriptions).

At first glance this definition seems to be meaningless, as for different D we obtain quite different notions, including ridiculous ones. For instance, if D is nowhere defined, then C_D is infinite everywhere. If $D(y) = \Lambda$ (the empty string) for all y, then the complexity of the empty string is 0 (since $D(\Lambda) = \Lambda$ and $l(\Lambda) = 0$), and the complexity of all the other strings is infinite.

Here is a more reasonable example: consider a decompressor D that just copies its input to output, that is, $D(x) = x$ for all x. In this case every string is its own description and $C_D(x) = l(x)$.

Of course, for any given string x we can find a description mode D that is tailored to x and with respect to which x has small complexity. Indeed, let $D(\Lambda) = x$. This implies $C_D(x) = 0$.

More generally, if we have some class of strings, we may look for a description mode that favors all the strings in this class. For example, for the class of strings consisting of zeros only we may consider the following decompressor:

$$D(\mathrm{bin}(n)) = 000\ldots000 \quad (n \text{ zeros}),$$

where $\mathrm{bin}(n)$ stands for the binary notation of natural number n. The length of the string $\mathrm{bin}(n)$ is about $\log_2 n$ (does not exceed $\log_2 n + 1$). With respect to this description mode, the complexity of the string consisting of n zeros is close to $\log_2 n$. This is much less that the length of the string (n). On the other hand, all strings containing symbol 1 have infinite complexity C_D.

It may seem that the dependence of complexity on the choice of the decompressor makes impossible any general theory of complexity. However, that is not the case.

Optimal description modes

A description mode is better when descriptions are shorter. According to this, we say that a description mode (decompressor) D_1 is *not worse* than a description mode D_2 if

$$C_{D_1}(x) \leqslant C_{D_2}(x) + c$$

for some constant c and for all strings x.

Let us comment on the role of the constant c in this definition. We consider a change in the complexity bounded by a constant as "negligible". One could say that such a tolerance makes the complexity notion practically useless, as the constant c can be very large. However, nobody managed to get any reasonable theory that overcomes this difficulty and defines complexity with better precision.

Example. Consider two description modes (decompressors) D_1 and D_2. Let us show that there exists a description mode D which is not worse than both of

them. Indeed, let

$$D(0y) = D_1(y),$$
$$D(1y) = D_2(y).$$

In other words, we consider the first bit of a description as the index of a description mode and the rest as the description (for this mode).

If y is a description of x with respect to D_1 (or D_2), then $0y$ (respectively, $1y$) is a description of x with respect to D as well. This description is only one bit longer, therefore we have

$$C_D(x) \leqslant C_{D_1}(x) + 1,$$
$$C_D(x) \leqslant C_{D_2}(x) + 1$$

for all x. Thus the mode D is not worse than both D_1 and D_2.

This idea is often used in practice. For instance, a `zip`-archive has a preamble; the preamble says (among other things) which mode was used to compress this particular file, and the compressed file follows the preamble.

If we want to use N different compression modes, we need to reserve initial $\log_2 N$ bits for the index of the compression mode.

Using a generalization of this idea, we can prove the following theorem:

THEOREM 1 (Solomonoff–Kolmogorov). *There is a description mode D that is not worse than any other one: for every description mode D' there is a constant c such that*

$$C_D(x) \leqslant C_{D'}(x) + c$$

for every string x.

A description mode D having this property is called *optimal*.

PROOF. Recall that a description mode by definition is a computable function. Every computable function has a program. We assume that programs are binary strings. Moreover, we assume that by reading the program bits from left to right, we can determine uniquely where it ends, that is, programs are "self-delimiting". Note that every programming language can be modified in such a way that programs are self-delimiting. For instance, we can double every bit of a given program (changing 0 to 00 and 1 to 11) and append the pattern 01 to its end.

Define now a new description mode D as follows:

$$D(Py) = P(y),$$

where P is a program (in the chosen self-delimiting programming language) and y is any binary string. That is, the algorithm D scans the input string from the left to the right and extracts a program P from the input. (If the input does not start with a valid program, D does whatever it wants, say, it goes into an infinite loop. The self-delimiting property guarantees that the decomposition of input is unique: if $Py = P'y'$ for two programs P and P', then one of the programs is a prefix of the other one.) Then D applies the extracted program P to the rest of the input (y) and returns the obtained result. (So D is just a "universal algorithm", or "interpreter"; the only difference is that program and input are not separated, and therefore we need to use a self-delimiting programming language.)

Let us show that indeed D is not worse than any other description mode P. We assume that the program P is written in the chosen self-delimiting programming

language. If y is a shortest description of the string x with respect to P, then Py is a description of x with respect to D (though not necessarily a shortest one). Therefore, compared to P, the shortest description is at most $l(P)$ bits longer, and

$$C_D(x) \leqslant C_P(x) + l(P).$$

The constant $l(P)$ depends only on the description mode P (and not on x). □

Basically, we used the same trick as in the preceding example, but instead of merging two description modes, we join all of them. Each description mode is prefixed by its index (program, identifier). The same idea is used in practice. A *self-extracting archive* is an executable file starting with a small program (a decompressor); the rest is considered as an input to that program. This program is loaded into the memory, and then it decompresses the rest of the file.

Note that in our construction, the optimal decompressor works for a very long time on some inputs (as some programs have large running time) and is undefined on some other inputs.

Kolmogorov complexity

Fix an optimal description mode D and call $C_D(x)$ the *Kolmogorov complexity* of the string x. In the notation $C_D(x)$ we drop the subscript D and write just $C(x)$.

If we switch to another optimal description mode, the change in complexity is bounded by an additive constant: for any two optimal description modes D_1 and D_2 there is a constant $c(D_1, D_2)$ such that

$$|C_{D_1}(x) - C_{D_2}(x)| \leqslant c(D_1, D_2)$$

for all x. Sometimes this inequality is written as

$$C_{D_1}(x) = C_{D_2}(x) + O(1),$$

where $O(1)$ stands for a bounded function of x.

Could we then consider the Kolmogorov complexity of a particular string x without having in mind a specific optimal description mode used in the definition of $C(x)$? No, since by adjusting the optimal description mode, we can make the complexity of x arbitrarily small or arbitrarily large. Similarly, the relation "string x is simpler than y", that is, $C(x) < C(y)$, has no meaning for two fixed strings x and y: by adjusting the optimal description mode, we can make any of these two strings simpler than the other one.

One may then wonder whether Kolmogorov complexity has any sense at all. Trying to defend this notion, let us recall the construction of the optimal description mode used in the proof of the Solomonoff–Kolmogorov theorem. This construction uses some programming language, and two different choices of this language lead to two complexities that differ at most by a constant. This constant is in fact the length of the program that is written in one of these two languages and interprets the other one. If both languages are "natural", we can expect this constant to be not that huge, just several thousands or even several hundreds. Therefore if we speak about strings whose complexity is, say, about 10^5 (i.e., a text of a long and not very compressible novel), or 10^6 (which is reasonable for DNA strings, unless they are compressible much more than the biologists think now), then the choice of the programming language is not that important.

Nevertheless one should always have in mind that all statements about Kolmogorov complexity are inherently asymptotic: they involve infinite sequences of

strings. This situation is typical also for computational complexity: usually upper and lower bounds for complexity of some computational problem are asymptotic bounds.

Complexity and information

One can consider the Kolmogorov complexity of x as the *amount of information* in x. Indeed, a string of zeros, which has a very short description, has little information, and a chaotic string, which cannot be compressed, has a lot of information (although that information can be meaningless—we do not try to distinguish between meaningful and meaningless information; so, in our view, any abracadabra has much information unless it has a short description).

If the complexity of a string x is equal to k, we say that x has k bits of information. One can expect that the amount of information in a string does not exceed its length, that is, $C(x) \leqslant l(x)$. This is true (up to an additive constant, as we have already said).

THEOREM 2. *There is a constant c such that*

$$C(x) \leqslant l(x) + c$$

for all strings x.

PROOF. Let $D(y) = y$ for all y. Then $C_D(x) = l(x)$. By optimality, there exists some c such that

$$C(x) \leqslant C_D(x) + c = l(x) + c$$

for all x. □

Usually this statement is written as follows: $C(x) \leqslant l(x) + O(1)$. Theorem 2 implies, in particular, that Kolmogorov complexity is always finite, that is, every string has a description.

Here is another property of "amount of information" that one can expect: the amount of information does not increase when algorithmic transformation is performed. (More precisely, the increase is bounded by an additive constant depending on the transformation algorithm.)

THEOREM 3. *For every algorithm A there exists a constant c such that*

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

for all x such that $A(x)$ is defined.

PROOF. Let D be an optimal decompressor that is used in the definition of Kolmogorov complexity. Consider another decompressor D':

$$D'(p) = A(D(p)).$$

(We apply first D and then A.) If p is a description of a string x with respect to D and $A(x)$ is defined, then p is a description of $A(x)$ with respect to D'. Let p be a shortest description of x with respect to D. Then we have

$$C_{D'}(A(x)) \leqslant l(p) = C_D(x) = C(x).$$

By optimality we obtain

$$C(A(x)) \leqslant C_{D'}(A(x)) + c \leqslant C(x) + c$$

for some c and all x. □

This theorem implies that the amount of information "does not depend on the specific encoding". For instance, if we reverse all bits of some string (replace 0 by 1 and vice versa), or add a zero bit after each bit of that string, the resulting string has the same Kolmogorov complexity as the original one (up to an additive constant). Indeed, the transformation itself and its inverse can be performed by an algorithm.

Here is one more example of a natural property of Kolmogorov complexity. Let x and y be strings. How much information does their concatenation xy have? We expect that the quantity of information in xy does not exceed the sum of those in x and y. This is indeed true; however, a small additive term is needed.

THEOREM 4. *There is a constant c such that for all x and y*

$$C(xy) \leqslant C(x) + 2\log C(x) + C(y) + c.$$

PROOF. Let us try first to prove the statement in a stronger form, without the term $2\log C(x)$. Let D be the optimal description mode that is used in the definition of Kolmogorov complexity. Define the following description mode D'. If $D(p) = x$ and $D(q) = y$, we consider pq as a description of xy, that is, we let $D'(pq) = xy$. Then the complexity of xy with respect to D' does not exceed the length of pq, that is, $l(p) + l(q)$. If p and q are minimal descriptions, we obtain $C_{D'}(xy) \leqslant C_D(x) + C_D(y)$. By optimality the same inequality holds for D in place of D', up to an additive constant.

What is wrong with this argument? The problem is that D' is not well defined. We let $D'(pq) = D(p)D(q)$. However, D' has no means to separate p from q. It may happen that there are two ways to split the input into p and q yielding different results:

$$p_1q_1 = p_2q_2 \quad \text{but} \quad D(p_1)D(q_1) \neq D(p_2)D(q_2).$$

There are two ways to fix this bug. The first one, which we use now, goes as follows. Let us prepend the string pq by the length $l(p)$ of string p (in binary notation). This allows us to separate p and q. However, we need to find where $l(p)$ ends, so let us double all the bits in the binary representation of $l(p)$ and then put 01 as separator. More specifically, let $\mathrm{bin}(k)$ denote the binary representation of integer k, and let $\overline{x}$ be the result of doubling each bit in x. (For example, $\mathrm{bin}(5) = 101$, and $\overline{\mathrm{bin}(5)} = 110011$.) Let

$$D'(\,\overline{\mathrm{bin}(l(p))}\,01pq) = D(p)D(q).$$

Thus D' is well defined: the algorithm D' scans $\overline{\mathrm{bin}(l(p))}$ while all the digits are doubled. Once it sees 01, it determines $l(p)$, and then scans $l(p)$ digits to find p. The rest of the input is q, and the algorithm is able to compute $D(p)D(q)$.

Now we see that $C_{D'}(xy)$ is at most $2l(\mathrm{bin}(l(p))) + 2 + l(p) + l(q)$. The length of the binary representation of $l(p)$ is at most $\log_2 l(p) + 1$. Therefore, xy has a description of length at most $2\log_2 l(p) + 4 + l(p) + l(q)$ with respect to D', which implies the statement of the theorem. □

The second way to fix the bug mentioned above goes as follows. We could modify the definition of Kolmogorov complexity by requiring descriptions to be self-delimiting; we discuss this approach in detail in Chapter 4.

Note also that we can exchange p and q and thus prove that

$$C(xy) \leqslant C(x) + C(y) + 2\log_2 C(y) + c.$$

How tight is the inequality of Theorem 4? Can $C(xy)$ be much less than $C(x)+C(y)$? According to our intuition, this happens when x and y have much in common. For example, if $x = y$, we have $C(xy) = C(xx) = C(x) + O(1)$, since xx can be algorithmically obtained from x and vice versa (Theorem 3).

To refine this observation, we will define the notion of the quantity of information in x that is missing in y (for all strings x and y). This value is called the *Kolmogorov complexity of x conditional to y* (or "given y") and is denoted by $C(x\,|\,y)$. Its definition is similar to the definition of the unconditional complexity. This time the decompressor D has access not only to the (compressed) description, but also to the string y. We will discuss this notion later in Section 2. Here we mention only that the following equality holds:

$$C(xy) = C(y) + C(x\,|\,y) + O(\log n)$$

for all strings x and y of complexity at most n. The equality reads as follows: the amount of information in xy is equal to the amount of information in y plus the amount of new information in x ("new" = missing in y).

The difference $C(x) - C(x\,|\,y)$ can be considered as "the quantity of information in y about x". It indicates how much the knowledge of y simplifies x.

Using the notion of conditional complexity, we can ask questions like this: How much new information does the DNA of some organism have compared to that of another organism's DNA? If d_1 is the binary string that encodes the first DNA and d_2 is the binary string that encodes the second DNA, then the value in question is $C(d_1\,|\,d_2)$. Similarly we can ask what percentage of information has been lost when translating a novel into another language: this percentage is the fraction

$$C(\text{original}\,|\,\text{translation})/C(\text{original}).$$

The questions about information in different objects were studied before the invention of algorithmic information theory. The information was measured using the notion of Shannon entropy. Let us recall its definition. Let ξ be a random variable that takes n values with probabilities $p_1, \ldots, p_n$. Then its Shannon entropy $H(\xi)$ is defined as

$$H(\xi) = \sum p_i(-\log_2 p_i).$$

Informally, the outcome having probability p_i carries $\log(1/p_i) = -\log_2 p_i$ bits of information (=surprise). Then $H(\xi)$ can be understood as the average amount of information in an outcome of the random variable.

Assume that we want to use Shannon entropy to measure the amount of information contained in some English text. To do this, we have to find an ensemble of texts and a probability distribution on this ensemble such that the text is "typical" with respect to this distribution. This makes sense for a short telegram, but for a long text (say, a novel) such an ensemble is hard to imagine.

The same difficulty arises when we try to define the amount of information in the genome of some species. If we consider as the ensemble the set of the genomes of all existing species (or even all species that ever existed), then the cardinality of this set is rather small (it does not exceed 2^{1000} for sure). And if we consider all

its elements as equiprobable, then we obtain a ridiculously small value (less than 1000 bits); for the non-uniform distributions the entropy is even less.

So we see that in these contexts Kolmogorov complexity looks like a more adequate tool than Shannon entropy.

Complexity and randomness

Let us recall the inequality $C(x) \leqslant l(x) + O(1)$ (Theorem 2). For most of the strings its left-hand side is close to the right hand side. Indeed, the following statement is true:

THEOREM 5. *Let n be an integer. Then there are less than 2^n strings x such that $C(x) < n$.*

PROOF. Let D be the optimal description mode used in the definition of Kolmogorov complexity. Then only strings $D(y)$ for all y, such that $l(y) < n$, have complexity less than n. The number of such strings does not exceed the number of strings y such that $l(y) < n$, i.e., the sum

$$1 + 2 + 4 + 8 + \ldots + 2^{n-1} = 2^n - 1$$

(there are 2^k strings for each length $k < n$). □

This implies that the fraction of strings of complexity less than $n - c$ among all strings of length n is less than $2^{n-c}/2^n = 2^{-c}$. For instance, the fraction of strings of complexity less than 90 among all strings of length 100 is less than 2^{-10}.

Thus the majority of strings (of a given length) are incompressible or almost incompressible. In other words, a randomly chosen string of the given length is almost incompressible. This can be illustrated by the following mental (or even real) experiment. Toss a coin, say, 80000 times, and get a sequence of 80000 bits. Convert it into a file of size 10000 bytes (8 bits = 1 byte). One can bet that no compression software (existing before the start of the experiment) can compress the resulting file by more than 10 bytes. Indeed, the probability of this event is less than 2^{-80} for every fixed compressor, and the number of (existing) compressors is not so large.

It is natural to consider incompressible strings as "random" ones: informally speaking, randomness is the absence of any regularities that may allow us to compress the string. Of course, there is no strict borderline between "random" and "non-random" strings. It is ridiculous to ask which strings of length 3 (i.e., 000, 001, 010, 011, 100, 101, 110, 111) are random and which are not.

Another example: assume that we start with a "random" string of length 10000 and replace its bits by all zeros (one bit at a step). At the end we get a certainly non-random string (zeros only). But it would be naïve to ask at which step the string has become non-random for the first time.

Instead, we can naturally define the "randomness deficiency" of a string x as the difference $l(x) - C(x)$. Using this notion, we can restate Theorem 2 as follows: the randomness deficiency is almost non-negative (i.e., larger than a constant). Theorem 5 says that the randomness deficiency of a string of length n is less than d with probability at least $1 - 1/2^d$ (assuming that all strings are equiprobable).

Now consider the Law of Large Numbers. It says that most of the n-bit strings have frequency of ones close to $1/2$. This law can be translated into Kolmogorov complexity language as follows: the frequency of ones in every string with small

randomness deficiency is close to 1/2. This translation implies the original statement since most of the strings have small randomness deficiency. We will see later that actually these formulations are equivalent.

If we insist on drawing a strict borderline between random and non-random objects, we have to consider infinite sequences instead of strings. The notion of randomness for infinite sequences of zeros and ones was defined by Kolmogorov's student P. Martin-Löf (he came to Moscow from Sweden). We discuss it in Section 3. Later C. Schnorr and L. Levin found a characterization of randomness in terms of complexity: an infinite binary sequence is random if and only if the randomness deficiency of its prefixes is bounded by a constant. This criterion, however, uses another version of Kolmogorov complexity called *monotone* complexity.

Non-computability of C and Berry's paradox

Before discussing applications of Kolmogorov complexity, let us mention a fundamental problem that reappears in any application. Unfortunately, the function C is not computable: there is no algorithm that given a string x finds its Kolmogorov complexity. Moreover, there is no computable non-trivial (unbounded) lower bound for C.

THEOREM 6. *Let k be a computable (not necessarily total) function from Ξ to $\mathbb{N}$. (In other words, k is an algorithm that terminates on some binary strings and returns natural numbers as results.) If k is a lower bound for Kolmogorov complexity, that is, $k(x) \leqslant C(x)$ for all x such that $k(x)$ is defined, then k is bounded: all its values do not exceed some constant.*

The proof of this theorem is a reformulation of the so-called *Berry's paradox*. This paradox considers

> *the minimal natural number that cannot be defined by at most fourteen English words.*

This phrase has exactly fourteen words and defines that number. Thus we get a contradiction.

Following this idea, consider the *first binary string whose Kolmogorov complexity is greater than a given number N*. By definition, its complexity is greater than N. On the other hand, this string has a short description that includes some fixed amount of information plus the binary notation of N (which requires about $\log_2 N$ bits), and the total number of bits needed is much less than N for large N. That would be a contradiction if we knew how to effectively find this string given its description. Using the computable lower bound k, we can convert this paradox into the proof.

PROOF. Consider the function $B(N)$ whose argument N is a natural number. It is computed by the following algorithm:

> perform in parallel the computations $k(\Lambda)$, $k(0)$, $k(1)$, $k(00)$, $k(01)$, $k(10)$, $k(11)$, ... until some string x such that $k(x) > N$ appears; then return x.

If the function k is unbounded, then the function B is total and $k(B(N)) > N$ by construction for every N. As k is a lower bound for K, we have $C(B(N)) > N$. On the other hand $B(N)$ can be computed given the binary representation $\mathrm{bin}(N)$

of N, therefore

$$C(B(N)) \leqslant C(\mathrm{bin}(N)) + O(1) \leqslant l(\mathrm{bin}(N)) + O(1) \leqslant \log_2 N + O(1)$$

(the first inequality is provided by Theorem 3 and the second one is provided by Theorem 2; term $O(1)$ stands for a bounded function). So we obtain

$$N < C(B(N)) \leqslant \log_2 N + O(1),$$

which cannot happen if N is large enough. □

Some applications of Kolmogorov complexity

Let us start with a disclaimer: the applications we will talk about are not real, practical applications; we just establish relations between Kolmogorov complexity and other important notions.

Occam's razor. We start with a philosophical question. What do we mean when we say that a theory provides a good explanation for some experimental data? Assume that we are given some experimental data and there are several theories to explain the data. For example, the data might be the observed positions of planets in the sky. We can explain them as Ptolemy did, with epicycles and deferents, introducing extra corrections when needed. On the other hand, we can use the laws of the modern mechanics. Why do we think that the modern theory is better? A possible answer: the modern theory can compute the positions of planets with the same (or even better) accuracy given fewer parameters. In other words, Kepler's achievement is a shorter description of the experimental data.

Roughly speaking, experimenters obtain binary strings and theorists find short descriptions for them (thus proving upper bounds for complexities of those strings); the shorter the description, the better the theorist.

This approach is sometimes called "Occam's razor" and is attributed to the philosopher William of Ockham who said that entities should not be multiplied beyond necessity. It is hard to judge whether he would agree with such an interpretation of his words.

We can use the same idea in more practical contexts. Assume that we design a machine that reads handwritten zip codes on envelopes. We are looking for a rule that separates, say, images of zeros from images of ones. An image is given as a Boolean matrix (or a binary string). We have several thousands of images and for each image we know whether it means 0 or 1. We want to find a reasonable separating rule (with the hope that it can be applied to the forthcoming images). What does "reasonable" mean in this context? If we just list all the images in our list together with their classification, we get a valid separation rule—at least it works until we receive a new image—however, the rule is way too long. It is natural to assume that a reasonable rule must have a short description, that is, it must have low Kolmogorov complexity.

Often an explanation for experimental data is only a tool to predict the future elements of the data stream. This aspect was the main motivation for Solomonoff [**187**]; it is outside the scope of our book and is considered in the book of M. Hutter [**68**].

Foundations of probability theory. The probability theory itself, being currently a part of measure theory, is mathematically sound and does not need any extra "foundations". The difficult questions arise, however, if we try to understand

why this theory could be applied to the real-world phenomena and how it should be applied.

Assume that we toss a coin a thousand times (or test some other hardware random number generator) and get a bit string of length 1000. If this string contains only zeros or equals 0101010101 ... (zeros and ones alternate), then we definitely will conclude that the generator is bad. Why?

The usual explanation: the probability of obtaining a thousand zeros is negligible (2^{-1000}) provided the coin is fair. Therefore, the conjecture of a fair coin is refuted by the experiment.

The problem with this explanation is that we do not always reject the generator: there should be some sequence α of a thousand zeros and ones which is consistent with this conjecture. Note, however, that the probability of obtaining the sequence α as a result of fair coin tossing is also 2^{-1000}. So what is the reason behind our complaints? What is the difference between the sequence of a thousand zeros and the sequence α?

The reason is revealed when we compare the Kolmogorov complexities of these sequences.

Proving theorems of probability theory. As an example, consider the Strong Law of Large Numbers. It claims that for almost all (according to the the uniform Bernoulli probability distribution) infinite binary sequences, the limit of frequencies of ones in their initial segments equals 1/2.

More formally, let Ω be the set of all infinite sequences of zeros and ones. The uniform Bernoulli measure on Ω is defined as follows. For every finite binary string x, consider the set Ω_x consisting of all infinite sequences that start with x. For example, $\Omega_\Lambda = \Omega$. The measure of Ω_x is equal to $2^{-l(x)}$. For example, the measure of the set Ω_{01}, that consists of all sequences starting with 01, equals 1/4.

For each sequence $\omega = \omega_0\omega_1\omega_2\ldots$ consider the limit of the frequencies of ones in the prefixes of ω, that is,

$$\lim_{n\to\infty} \frac{\omega_0 + \omega_1 + \ldots + \omega_{n-1}}{n}.$$

We say that ω satisfies the Strong Law of Large Numbers (SLLN) if this limit exists and is equal to 1/2. For instance, the sequence 010101 ..., having period 2, satisfies the SLLN, and the sequence 011011011 ..., having period 3, does not.

The SLLN says that the set of sequences that do not satisfy SLLN has measure 0. Recall that a set $A \subset \Omega$ has measure 0 if for all $\varepsilon > 0$ there is a sequence of strings $x_0, x_1, x_2, \ldots$ such that

$$A \subset \Omega_{x_0} \cup \Omega_{x_1} \cup \Omega_{x_2} \cup \ldots$$

and the sum of the series

$$2^{-l(x_0)} + 2^{-l(x_1)} + 2^{-l(x_2)} + \ldots$$

(the sum of the measures of Ω_{x_i}) is less than ε.

One can prove SLLN using the notion of a Martin-Löf random sequence mentioned above. The proof consists of two parts. First, we show that every Martin-Löf random sequence satisfies SLLN. This can be done using Levin–Schnorr randomness criterion (if the limit does not exist or differs from 1/2, then the complexity of some prefix is less than it should be for a random sequence).

The second part is rather general and does not depend on the specific law of probability theory. We prove that the set of all Martin-Löf non-random sequences

has measure zero. This implies that the set of sequences that do not satisfy SLLN is included in a set of measure 0 and hence has measure 0 itself.

The notion of a random sequence is philosophically interesting in its own right. In the beginning of twentieth century Richard von Mises suggested using this notion (he called it in German *Kollektiv*) as a basis for probability theory (at that time the measure theory approach had not yet been developed). He considered the so-called "frequency stability" as a main property of random sequences. We will consider von Mises' approach to the definition of a random sequence (and subsequent developments) in Chapter 9.

Lower bounds for computational complexity. Kolmogorov complexity turned out to be a useful technical tool when proving lower bounds for computational complexity. Let us explain the idea using the following model example.

Consider the following problem. Initially, a string x of length n is located in the n leftmost cells of the tape of a Turing machine. The machine has to copy x, that is, to get xx on the tape (the string x is intact and its copy is appended), and halt.

Since the middle of the 1960s it has been well known that a (one-tape) Turing machine needs time proportional to n^2 to perform this task. More specifically, one can show that for every Turing machine M that can copy every string x, there exists some $\varepsilon > 0$ such that for all n there is a string x of length n whose copying requires at least εn^2 steps.

Consider the following intuitive argument supporting this claim. The number of internal states of a Turing machine is a constant (depending on the machine). That is, the machine can keep in its memory only a finite number of bits. The speed of the head movement is also limited: one cell per step. Hence the rate of information transfer (measured in *bit·cell/step*) is bounded by a constant depending on the number of internal states. To copy a string x of length n, we need to move n bits by n cells to the right; therefore, the number of steps should be proportional to n^2 (or more).

Using Kolmogorov complexity, we can make this argument rigorous. A string is hard to copy if it contains maximal amount of information, i.e., its complexity is close to n. We consider this example in detail in Section 8.2 (p. 233).

A combinatorial interpretation of Kolmogorov complexity. We consider here one example of this kind (see Chapter 10, p. 313, for more detail). One can prove the following inequality for the complexity of three strings and their combinations:

$$2C(xyz) \leqslant C(xy) + C(xz) + C(yz) + O(\log n)$$

for all strings x, y, z of length at most n.

It turns out that this inequality has natural interpretations that are not related to complexity at all. In particular, it implies (see [**65**]) the following geometrical fact:

Consider a body B in a three-dimensional Euclidean space with coordinate axes OX, OY, and OZ. Let V be B's volume. Consider B's orthogonal projections onto coordinate planes OXY, OXZ, and OYZ. Let S_{xy}, S_{xz}, and S_{yz} be the areas of these projections. Then

$$V^2 \leqslant S_{xy} \cdot S_{xz} \cdot S_{yz}.$$

Here is an algebraic corollary of the same inequality. For every group G and its subgroups X, Y, and Z, we have

$$|X \cap Y \cap Z|^2 \geqslant \frac{|X \cap Y| \cdot |X \cap Z| \cdot |Y \cap Z|}{|G|},$$

where $|H|$ denotes the number of elements in H.

Gödel incompleteness theorem. Following G. Chaitin, let us explain how to use Theorem 6 in order to prove the famous Gödel incompleteness theorem. This theorem states that not all true statements of a formal theory that is "rich enough" (the formal arithmetic and the axiomatic set theory are two examples of such a theory) are provable in the theory.

Assume that for every string x and every natural number n, one can express the statement $C(x) > n$ as a formula in the language of our theory. (This statement says that the chosen optimal decompressor D does not output x on any input of length at most n; one can easily write this statement in formal arithmetic and therefore in set theory.)

Let us generate all the proofs (derivations) in our theory and select those of them which prove some statement of the form $C(x) > n$ where x is some string and n is some integer (statements of this type have no free variables). Once we have found a new theorem of this type, we compare n with all previously found n's. If the new n is greater than all previous n's, we write the new n into the "records table" together with the corresponding x_n.

There are two possibilities: either (1) the table will grow infinitely, or (2) there is the last statement $C(X) > N$ in the table which remains unbeaten forever. If (2) happens, there is an entire class of true statements that have no proof. Namely, all true statements of the form $C(x) > n$ with $n > N$ have no proofs. (Recall that by Theorem 5 there are infinitely many such statements.)

In the first case we have infinite computable sequences of strings $x_0, x_1, x_2 \ldots$ and numbers $n_0 < n_1 < n_2 < \ldots$ such that all statements $C(x_i) > n_i$ are provable. We assume that the theory proves only true statements; thus, all the inequalities $C(x_i) > n_i$ are true. Without loss of generality, we can assume that all x_i are pairwise different (we can omit x_i if there exists $j < i$ such that $x_j = x_i$; every string can occur only finitely many times in the sequence $x_0, x_1, x_2 \ldots$ since $n_i \to \infty$ as $i \to \infty$). The computable function k, defined by the equation $k(x_i) = n_i$, is then an unbounded lower bound for Kolmogorov complexity. This contradicts Theorem 6.