Contents

What is this book about?

What is Kolmogorov complexity?

Roughly speaking, Kolmogorov complexity means "compressed size". Programs like `zip`, `gzip`, etc., compress a file (text, program, 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).

A file that has a regular structure can be compressed significantly. Its compressed size (aka Kolmogorov complexity) is small compared to its length. On the other hand, a file without regularities hardly can be compressed, and its Kolmogorov complexity 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 the recursion theory terminology, we say that a description mode is a partial computable (=partial recursive) function from Ξ to Ξ, where Ξ stands for the set of all binary strings. Let us remind that we associate with any 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. 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:

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

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 $KS_D(x)$ 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 KS_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 other strings is infinite.

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 $KS_D(x) = l(x)$.

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 $KS_D(x) = 0$.

More general, 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 K_D.

It may seem that the dependence of complexity on the choice of the decompressor makes impossible any general theory of complexity. However, it 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

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

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

$$KS_D(x) \leqslant KS_{D_1}(x) + 1$$
$$KS_D(x) \leqslant KS_{D_2}(x) + 1$$

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:

{intro-univ

Theorem 1 (Kolmogorov-Solomonoff) *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*

$$KS_D(x) \leqslant KS_{D'}(x) + c$$

for every string x.

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

$\lhd$ 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 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, goes into an infinite loop.) Then D applies the extracted program p to the rest of the input (y) and returns the obtained result. (So D is just an "universal algorithm", or "interpreter"; the only difference is that program and input are not separated and therefore we need to use self-delimiting programming language.)

Let us show that indeed D is not worse than any other description mode P. Let p be a program computing a function P and written in the chosen 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

$$KS_D(x) \leqslant KS_P(x) + l(p).$$

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

Basically, we used the same trick as in the preceding example; 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 input to that program. The program is loaded into the memory and then it decompresses the rest of the file.

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

Kolmogorov complexity

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

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

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

for all x. Sometimes this inequality is written as follows:

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

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 $KS(x)$? 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, $KS(x) < KS(y)$, 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 wonder whether Kolmogorov complexity has any sense at all. 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 novel), or 10^6 (DNA string) then the choice of the programming language is not that important.

Nevertheless one should always remember 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 consisting of 0s, 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, $KS(x) \leqslant l(x)$. This is true (up to an additive constant, as we have already said).

{intro-leng

Theorem 2 *There is a constant c such that*

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

for all strings x.

$\lhd$ Let $P(y) = y$ for all y. Then $KS_P(x) = l(x)$. By optimality, there exists some c such that

$$KS(x) \leqslant KS_P(x) + c = l(x) + c$$

for all x. $\rhd$

Usually this statement is written as follows: $KS(x) \leqslant l(x) + O(1)$. 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.)

{intro-tran

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

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

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

$\lhd$ Let D be an optimal decompressor that is used in the definition of the 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

$$KS_{D'}(A(x)) \leqslant l(p) = KS_D(x) = KS(x).$$

By optimality we obtain

$$KS(A(x)) \leqslant KS_{D'}(A(x)) + c \leqslant KS(x) + c$$

for some c and all x. $\triangleright$

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.

Let x and y be strings. How much information has their concatenation xy? 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.

{intro-pai

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

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

$\triangleleft$ Let us try first to prove the statement in a stronger form, without the term $2\log KS(x)$. 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 then we obtain $KS_{D'}(xy) \leqslant KS_D(x) + KS_D(y)$. 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 $KS_{D'}(xy)$ 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. $\triangleright$

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 Section 4.

Note also that we can exchange p and q and thus prove that $KS(xy) \leqslant KS(x) + KS(y) + 2\log_2 KS(y) + c$.

How tight is the inequality of Theorem 4? Can $KS(xy)$ be much less than $KS(x) + KS(y)$? According to our intuition, this happens when x and y have much in common. For example, if $x = y$, we have $KS(xy) = KS(xx) = KS(x) + O(1)$, 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 every strings x and y). This value is called the *the Kolmogorov complexity of x conditional to y* (or "given 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:

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

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 $KS(x) - KS(x|y)$ 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 has DNA of some organism compared to 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 $KS(d_1|d_2)$. Similarly we can ask what percentage of information has been lost when translating a novel into another language: this percentage is the fraction

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

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 follows:

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

Informally, the outcome having probability p_i carries $(-\log_2 p_i)$ 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 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 (what else can we choose?) then we obtain a ridiculously small value (less than 1000 bits).

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

{intro-carc

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

$\lhd$ 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 $k < n$). $\rhd$

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 ($000, \ldots, 111$) 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 naive 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 $l(x) - KS(x)$. 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, which 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 further 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 student P. Martin-Löf (Sweden). We discuss it in Section 3. Later L. Levin and C. Schnorr 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 *KS* and Berry's paradox

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

{intro-nobo

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, $k(x) \leqslant KS(x)$ for all x such that $k(x)$ is defined, then k is bounded: all its values do not exceed some constant.*

$\lhd$ 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 appearing 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 strings 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—a contradiction. How to find such a string? To this end we need a computable lower bound for Kolmogorov complexity.

We proceed as follows. Consider the function $B(N)$, whose argument N is a natural number and which is computed by the following algorithm:

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

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

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

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

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

which cannot happen if N is large enough. $\triangleright$

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 less 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 is, the better is 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 interpretation of his words.

We can use the same idea in more practical contexts. Assume that we design an automaton 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 means "reasonable" 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.

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 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?

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

On the other hand, we do not always reject the generator: there should be some sequence α of 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 of our complaints? What is the difference between the sequence of thousand zeros and the sequence α?

The reason is revealed when we compare 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 infinite binary sequences the limit of frequency of 1s in their initial segments equals $1/2$ (according to the the uniform Bernoulli probability distribution).

In more detail: 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 Ω_{01}, which 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 1s 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 $010101\ldots$, having period 2, satisfies the SLLN and the sequence $011011011\ldots$, having period 3, does not.

The Strong Law of Large Numbers 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 XXth century Richard von Mises suggested to use this notion (he called it in German "Kollektiv") as a basis for probability theory (at that time the measure theory approach was not developed yet). 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 the subsequent developments) in Section **??**.

Lower bounds for computational complexity. Kolmogorov complexity turned out to be a useful technical tool in 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 1960ies it is well known that (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 does copying (for all strings x) there exists $\varepsilon > 0$ 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 $bit \cdot cell/step$) 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.

A combinatorial interpretation of Kolmogorov complexity. We consider here one example of this kind. One can prove the following inequality for complexity of three strings and their combinations: {intro-trip

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

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 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 $KS(x) > n$ 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 the formal arithmetic and therefore in the set theory.)

Let us generate all the proofs (derivations) in our theory looking for the proofs of statements of the form $KS(x) > n$ 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 $KS(X) > N$ 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 $KS(x) > n$ 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 $KS(x_i) > n_i$ are provable. We assume that the theory proves only true statements, thus all the inequalities $KS(x_i) > n_i$ 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 of Kolmogorov complexity. This contradicts Theorem 6.

1 Plain complexity

1.1 The definition and main properties

{simpledf}

Let us recall the definition of Kolmogorov complexity from the Introduction. This version of complexity was defined by Kolmogorov in his pioneer paper [**?**]. In order to distinguish it from later versions we call it the *plain* Kolmogorov complexity. Later we will consider also other versions of Kolmogorov complexity, including the prefix one and the monotone one. In this section by Kolmogorov complexity we always mean the plain one.

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

$$KS_D(x) = \min\{l(y) | D(y) = x\}.$$

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 $KS_{D_1}(x) \leqslant KS_{D_2}(x) + c$ for all x and write this as $KS_{D_1}(x) \leqslant KS_{D_2}(x) + O(1)$.

A description mode is called *optimal* if it is not worse than any other description mode. By Kolmogorov–Solomonoff universality theorem (Theorem 1, p. 6) optimal description modes exist. Let us remind shortly its proof. Let U be an interpreter of a universal programming language, that is, $U(p,x)$ is the result 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

$$KS_{D'}(x) \leqslant KS_D(x) + l(\hat{p}).$$

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 $KS(x)$ (we drop the subscript) denote the complexity of x with respect to D.

As the optimal description mode is not worse than the identity function $x \mapsto x$, we obtain the inequality $KS(x) \leqslant l(x) + O(1)$ (Theorem 2, p. 8).

Let A be a partial computable function. Comparing the optimal description mode D with the description mode $y \mapsto A(D(y))$ we obtain the inequality $KS(A(x)) \leqslant KS(x) + O(1)$, which can be interpreted as the non-growth of complexity under algorithmic transformations (Theorem 3, p. 8).

Using this inequality, we can define Kolmogorov complexity of other "finite objects" like natural numbers, graphs, permutations, finite sets of strings, etc., which 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 the 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, a natural number n can be represented 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 anyway the Kolmogorov complexity of binary strings is defined up to an additive constant.

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 $\log n + O(1)$.

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 $KS(n) + O(1) \leqslant \log n + O(1)$. (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, Theorem 5, p. 11.)

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

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

We have used several times the upper bound 2^n for the number of strings x with $KS(x) < n$. 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 $KS(x) < n$ 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 Introduction, the lower bound is implied by the inequality $KS(x) \leqslant l(x) + c$: the complexity of all the strings of length less than $n-c$ is less than n.] {distributi

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 compexity is by $\sum_k k\alpha_k$ less than n. Use the inequality $\alpha_k \leqslant 2^{-k}$ and the convergence of the series $\sum k/2^k$.] {average-pl

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

{simple-upp

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

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

$\lhd$ Let us prove the theorem. First, we need to show that the set $\{\langle n,x\rangle \mid x \in S_n\} = \{\langle n,x\rangle \mid KS(x) < n\}$, 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 KS. Perform in parallel the computations of D on all the inputs. (Say, for $k = 1,2,\ldots$ 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 $\langle l(y)+2,x\rangle, \langle l(y)+3,x\rangle \ldots$ in parallel to printing of other pairs.

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

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

(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 $x_0,x_1,x_2\ldots$ are distinct.) Let y_k be the kth string of length n in the 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 has 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 $KS_{D_V}(x)\leqslant n$. Comparing D_V with the optimal description mode we see that there is a constant c such that $KS(x)<n+c$ for all $x\in V_n$. Theorem 7 is proven. $\triangleright$

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

Theorem 7 can be reformulated as follows. Let $f(x)$ be a function defined on all binary strings and taking 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

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

and

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

for all x. The values of F are natural numbers as well as $+\infty$. 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 he 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.

{simple-upp

Theorem 8 **(a)** *The function KS is upper semicomputable and* $|\{x \mid KS(x) < n\}| < 2^n$ *for all n.*

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

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

Theorem 8 allows to define Kolmogorov complexity as a minimal (up to an additive constant) upper semicomputable function K that satisfies the inequality $|\{x \mid K(x) < n\}| = O(2^n)$. One can replace the requirement of minimality in this definition by some other properties of KS. In this way we obtain the following "axiomatic" definition of Kolmogorov complexity.

{simple-axi

Theorem 9 *Let K be any function defined on binary strings and taking natural values. 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;* [complexity non-increase axiom]

(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 $K(x) = KS(x) + O(1)$*, that is, the difference* $|K(x) - KS(x)|$ *is bounded by a constant.*

$\lhd$ Theorem 8 implies that $KS(x) \leqslant K(x) + O(1)$. So we need to prove that $K(x) \leqslant KS(x) + O(1)$.

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

$$M_0 \subset M_1 \subset M_2 \subset \ldots$$

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 proved.

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 proved.

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 $K(x) = K(D(p)) \leqslant K(p) + O(1) \leqslant l(p) + O(1) = KS(x) + O(1)$. Note that we have used twice the property (b): in the proof of Lemma 2 and just now. $\triangleright$

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 the half of the regular complexity.

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 $KS(x) \leqslant \log f(n) + c$ for all n and all $x \in A_n$. {complexity}

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 $KS(y) \leqslant n - \log n + c$. [Hint. For a given natural k consider a Boolean matrix of size $k \times (2^k - 1)$ whose columns are all nonzero 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 has not the form $2^k - 1$ we can flip one of the first $2^k - 1$ bits for an appropriate k.]

1.2 Algorithmic properties

{simpleal}

The function KS is upper semicomputable. On the other hand, it is not computable and, moreover, it has no unbounded computable lower bounds (Theorem 6 on page 12).

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 $KS_D(x)$ just by trying all descriptions in the lexicographical order until we find a shortest one.

At first glance, this contradicts to our intuition: the bigger the domain of D is, the better D is. If 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' (the difference is only for one point, 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" [**?**] (by the way, in this book he also discussed the power of quantum computers long before quantum computing became fashionable).

Note that the domain of every optimal description mode is undecidable. (The set of strings is called *decidable* 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 (say $D(x) = 0$ 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, [**?**]).

Below we consider other applications of Kolmogorov complexity in Computability theory.

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 an 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 subsets.

Call a string x "simple" if $KS(x) < l(x)/2$.

{simple-set

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

$\lhd$ That set S of all "simple" strings is enumerable. Indeed, the function KS is upper semicomputable. Run an algorithm that generates all the pairs $\langle x,n\rangle$ such that $KS(x) < n$. Once we see a pair $\langle x,n\rangle$ with $n \leqslant l(x)/2$ we print x.

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 C. We can use C to obtain a computable unbounded lower bound of KS. To find a string of complexity greater than t we can generate elements of C until we find a string c_t of length greater than $2t$. As C is infinite, there is such string. The complexity of c_t is greater than t, otherwise c_t is simple. Without loss of generality we can assume that the strings c_t, $t = 1,2,\ldots$ are pairwise different. Thus the function

$c_t \mapsto t$ is a computable unbounded lower bound of KS. This contradicts to Theorem 6 (page 12). Theorem 10 is proved. $\triangleright$

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

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 KS becomes a function of a natural argument. The function $KS(m)$ 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, in $\log n + O(1)$ bits. We have seen this in the proof of Theorem 6 on page 12.

In this section, we study in detail the rate of convergence of KS to infinity. For every natural n consider the largest number $B(n)$ whose complexity is at most n:

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

The function $n \mapsto B(n)$ may be called the "regulator" of the convergence of $KS(m)$ to infinity. Indeed, $K(x) > n$ for all $x > B(n)$. It can happen, for small values of n, that $KS(m) > n$ for all m. In this case we let $B(n) = -1$.

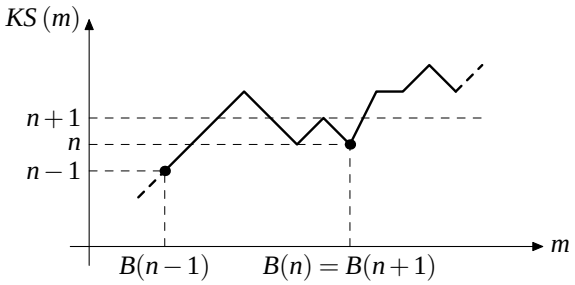

Figure 1: The definition of $B(n)$: the value $KS(m)$ does not exceed $n-1$ for $m = B(n-1)$ (the case when $KS(B(n-1)) = n-1$ is shown), and $KS(m) \geqslant n$ for all $m > B(n-1)$. At the point $m = B(n)$ the value of KS does not exceed n (the case when $KS(B(n)) = n$ is shown), and $KS(m) > n$ for all $m > B(n)$. The case when $KS(m)$ 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 $KS_{\geqslant}(m)$ is equal to n.

The function B is in a sense an inverse function to the function $KS_{\geqslant}(N) = \min\{KS(m) \mid m \geqslant N\}$. The function $KS_{\geqslant}$ increases very slowly. It takes the value n on the segment $(B(n-1), B(n)]$. The slow increase of $KS_{\geqslant}$ corresponds to the fast increase of B. The latter can be illustrated by the following

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.

$\lhd$ As algorithmic transformations do not increase complexity, for some constant c for all n we have

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

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

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

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

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

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

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.*

$\lhd$ 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., $B(n+c)$. $\rhd$

This observation is useful in the following particular case. Let M be an algorithm and X 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 show the following:

{b-versus-b

Theorem 14 *Let $BB(n)$ denote the largest 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.

$\lhd$ 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. The list of all strings of length at most n in the domain of D, and hence the number $BB(n)$, can be found given n and α_n. This information can be encoded in one string of length $n+1$, the string $0\ldots01\alpha_n$ (there are $n-l(\alpha_n)$ zeros in the beginning). Thus Kolmogorov complexity of $BB(n)$ is at most $n+O(1)$, and therefore $BB(n) \leqslant B(n+c)$ for some c and all n.

Let us prove the second inequality of the theorem. Given any $t > BB(n)$ and n we can find a string of complexity bigger than n: we run D on all inputs of length at most n within t steps to find the list of all strings of complexity at most n and then take the first string that is not listed. So $KS(t)$ bits plus $2\log n$ bits for self-delimiting description of n are enough to specify a string of complexity greater than n. Therefore, $KS(t) \geqslant n - 2\log n - O(1)$ for all $t > BB(n)$. This implies that $BB(n) \geqslant B(n - 2\log n - c)$.

This inequality is weaker than claimed $BB(n) \geqslant B(n-c)$. To get rid of the term $2\log n$ note that actually we do not need to know n exactly. It is enough to know any $n' \geqslant n$. Indeed, we can run D on all inputs of length at most n' within t steps and then take the first string that has not appeared as the output of D.

As such n' we can take t itself. Indeed, we have $t > BB(n) > B(n - 2\log n - c) \gg n$ provided n is large enough. Thus given every $t > BB(n)$ we can find a string of complexity bigger than n, therefore the complexity of t is at least $n - O(1)$. Hence $BB(n) > B(n - O(1))$. $\rhd$

This theorem shows that, within to 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 is defined as the maximal number of 1s 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) the number $B(n)$;

(d) the number $BB(n)$;

(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 description of length at most n;

(h) the graph T_n of the function $KS(x)$ 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.

{quasi-omeg

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

◁ 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 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 ((a)→(b)) and the largest number of complexity at most n ((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). But we still do not know 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. Given $B(n)$, we can find an upper bound for $BB(n-c)$ (for appropriate c). Thus 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 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 the 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 $n \leqslant KS(\gamma_n) \leqslant (n-c-d)+2\log(c+d)+O(1)$, hence $(c+d) = O(1)$.

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

Let X_n be one of objects (a)–(h). We have just proved 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, $KS(X_n) \leqslant KS(\gamma_{n+c})+O(1) \leqslant n+O(1)$.

To prove the lower bound of $KS(X_n)$ 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, $n-c \leqslant KS(\gamma_{n-c}) \leqslant (n-d)+2\log d+O(1)$. 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, $KS(x) \geqslant l(x)$. 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 halting problem is *Turing reducible* to the set of incompressible strings. This implies that halting problem is also reducible to the "upper graph" of KS, that is, to the set $\{\langle x,k\rangle \mid KS(x) < k\}$. Using the terminology of computability theory we can say that the set of compressible strings 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 an upper bound for the number of oracle queries (for the set $\{\langle x,k\rangle \mid KS(x) < k\}$) 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 we have $B(n+c) \geqslant f(B(n))$. [Hint: the complexity of $f(B(n))$ is at most $n+O(1)$.]

12 Call a set U *r-separable* [**?**] 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 $\{\langle x,k\rangle \mid KS(x) < k\}$ (the upper graph of KS) is an r-separable set. [Hint: assume that this set is disjoint with an enumerable set V. The set of the second components of pairs in V is finite, otherwise we get an unbounded computable lower bound for KS. 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 set does not m-reduce to the upper graph of KS). [Hint: there is a pair of disjoint enumerable inseparable sets.]

This problem shows how Kolmogorov complexity can be used to construct an enumerable undecidable set that is not m-complete.

Theorem 15 selects some very special objects among all objects of complexity n (in fact, one object up to equivalence described above). At first glance, this seems strange: our intuition says that all "random" strings of length n should be indistinguishable. (A string of length n is "random" if its complexity is close to n.) If there is a property that distinguishes a string of length n from other strings then this property can be used to compress the string. However we have found a very special random string γ_n of length n. This paradox can be explained as follows: the individual property of γ_n does allow to find a short description of 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 discussing "the number of wisdom" Ω and in Section **??** studying 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) is about n while the length of (c) and (d) grows faster than every computable function of n.

2 Complexity of pairs and conditional complexity

{conditiona

2.1 Complexity of pairs

{conditp}

As we have discussed, we can define complexity of any constructive object using (computable) encodings by strings. In this section we deal with pairs of strings. A pair x,y can be encoded, e.g., by a string $[x,y] = \bar{x}01y$; here $\bar{x}$ stands for x with doubled bits. Any other computable encoding $x,y \mapsto [x,y]$ could be used (of course, we need that $[x,y] \neq [x',y']$ if $x \neq x'$ or $y \neq y'$). Any two encodings of this type are equivalent (there are translation algorithms in both directions), so Theorem 3 (p. 8) guarantees that complexities of the different encoding of the same pair differ by $O(1)$.

So let us fix some encoding $[x,y]$. *Kolmogorov complexity of a pair* x,y is defined as $KS([x,y])$. Notation: $KS(x,y)$. Here are some evident properties:

- $KS(x,x) = KS(x) + O(1)$;
- $KS(x,y) = KS(y,x) + O(1)$;
- $KS(x) \leqslant KS(x,y) + O(1)$; $KS(y) \leqslant KS(x,y) + O(1)$.

The following theorem gives an upper bound for the complexity of a pair in terms of complexities of its components:

{condit-pai

Theorem 16

$$KS(x,y) \leqslant KS(x) + 2\log KS(x) + KS(y) + O(1);$$
$$KS(x,y) \leqslant KS(x) + \log KS(x) + 2\log\log KS(x) + KS(y) + O(1);$$
$$KS(x,y) \leqslant KS(x) + \log KS(x) + \log\log KS(x) + 2\log\log\log KS(x) + KS(y) + O(1);$$
$$\ldots$$

(We can continue this sequence of inequalities indefinitely. Also, one can exchange x and y.)

$\lhd$ This proof (for the first inequality) was already explained in the Introduction (Theorem 4, p. 9). The only difference is that we considered the concatenation xy instead of a pair. Let us repeat it for pairs.

A computable mapping $x \mapsto \hat{x}$ (here x and $\hat{x}$ are binary strings) is called a *prefix-free encoding*, if for any two different string x and y the string $\hat{x}$ is not a prefix of the string $\hat{y}$. (In particular, $\hat{x} \neq \hat{y}$ if $x \neq y$.) This guarantees that both x and y can be uniquely reconstructed from $\hat{x}y$.

An example $x \mapsto \bar{x}01$, where $\bar{x}$ stands for x with doubled bits, is a prefix-free encoding. Here the block 01 are used as a delimiter. However, this encoding is not the most space-efficient one, since it doubles the length. A better prefix-free encoding:

$$x \mapsto \hat{x} = \overline{\mathrm{bin}(l(x))}\,01x$$

($\mathrm{bin}(l(x))$ is the binary representation of the length $l(x)$ of the string x). Now

$$l(\hat{x}) = l(x) + 2\log l(x) + O(1).$$

This trick can be "iterated": for any prefix-free encoding $x \mapsto \hat{x}$ we can construct a new (and also prefix-free) encoding

$$x \mapsto \widehat{\mathrm{bin}(l(x))}x.$$

Indeed, if a string $\widehat{\mathrm{bin}(l(x))}x$ is a prefix of $\widehat{\mathrm{bin}(l(y))}y$, then one of the strings $\widehat{\mathrm{bin}(l(x))}$ and $\widehat{\mathrm{bin}(l(y))}$ is a prefix of the other one, and therefore $\mathrm{bin}(l(x)) = \mathrm{bin}(l(y))$. Therefore x is a prefix of y, and $l(x) = l(y)$, so $x = y$. (In other terms, we uniquely determine the length of the string, since a prefix-free code is used for it, and then get the string itself knowing where it ends.)

In this way we get a prefix-free encoding such that

$$l(\hat{x}) = l(x) + \log l(x) + 2\log\log l(x) + O(1),$$

then (one more iteration)

$$l(\hat{x}) = l(x) + \log l(x) + \log\log l(x) + 2\log\log\log l(x) + O(1)$$

etc.

Now we return to the proof. Let D be the optimal decompressor used in the definition of Kolmogorov complexity. Consider a decompressor D' defined as follows:

$$D'(\hat{p}q) = [D(p), D(q)],$$

where $\hat{p}$ is a prefix-free encoding and $[\cdot,\cdot]$ is the encoding of pairs (used in the definition of pairs complexity). Since $\hat{p}$ is a prefix-free encoding, D' is well defined (we can uniquely extract $\hat{p}$ out of $\hat{p}q$).

Let p and q be the shortest descriptions of x and y. Then $\hat{p}q$ is a description of $[x,y]$, and its length is exactly as we need in our theorem. (The more iteration we use for the prefix-free encoding, the better bound we have.) $\triangleright$

Theorem 16 implies that

$$KS(x,y) \leqslant KS(x) + KS(y) + O(\log n)$$

for strings x and y of length at most n: one may say that the complexity of a pair does not exceed the sum of the complexities of its component with logarithmic precision.

A natural question arises: is it true that $KS(x,y) \leqslant KS(x) + KS(y) + O(1)$?

A simple argument shows that this is not the case. Indeed, this inequality would imply $KS(x,y) \leqslant l(x) + l(y) + O(1)$. Consider some N. For each $n = 0, 1, 2, \ldots, N$ we have 2^n strings x of length n and 2^{N-n} strings y of length $N-n$. Combining them, we (for a given n) obtain 2^N different pairs $\langle x,y\rangle$. The total number of pairs (all $n = 0, 1, \ldots, N$ give different pairs) is $(N+1)2^N$.

Assume that indeed $K(x,y) \leqslant l(x) + l(y) + O(1) = N + O(1)$ for all these pairs. Then we get $(N+1)2^N$ different strings $[x,y]$ of complexity at most $N + O(1)$, but this is impossible (Theorem 7, p. 19, gives the upper bound $O(2^N)$).

13 Prove that there is no constant c such that {conditp-nc

$$KS(x,y) \leqslant KS(x) + \log KS(x) + KS(y) + c$$

for all x and y. [Hint: Replace KS in the right hand side by l and count the number of corresponding pairs.]

14 Give a (natural) definition of complexity for triples of strings (instead of pairs). Prove that $KS(x,y,z) \leqslant KS(x) + KS(y) + KS(z) + O(\log n)$ for strings x,y,z of length at most n.

15 (a) Prove that

$$\sum_{x \in \Xi} 2^{-l(\hat{x})} \leqslant 1$$

for any prefix-free encoding $x \mapsto \hat{x}$ (here Ξ is the set of all binary strings).

(b) Prove that if a prefix-free encoding increases the length at most by $f(n)$ (where n is the initial length), i.e., if $l(\hat{x}) \leqslant l(x) + f(l(x))$, then $\sum_n 2^{-f(n)} < \infty$.

This problem explains why a coefficient 2 appears in the Theorem 16 (p. 30): the series

$$\sum \frac{1}{n^2}, \quad \sum \frac{1}{n(\log n)^2}, \quad \sum \frac{1}{n \log n (\log\log n)^2}, \ldots$$

are convergent, while the series

$$\sum \frac{1}{n}, \quad \sum \frac{1}{n \log n}, \quad \sum \frac{1}{n \log n \log\log n}, \ldots$$

are divergent.

16 Prove that all the inequalities of Theorem 16 become false if the coefficient 2 is replaced by 1, but remain true with the coefficient $1+\varepsilon$ for any $\varepsilon > 0$. [Hint: the first inequality was considered in Problem 13.]

17 Prove that

$$KS(x,y) \leqslant KS(x) + \log KS(x) + KS(y) + \log KS(y) + O(1).$$

18 (Continued) Prove a stronger inequality:

$$KS(x,y) \leqslant KS(x) + KS(y) + \log(KS(x) + KS(y)) + O(1).$$

(note that $KS(x) + KS(y)$ can be replaced by $\max(KS(x), KS(y))$, this gives a factor at most 2, which makes $O(1)$ after taking logarithms).

19 Prove that $KS(x, KS(x)) = KS(x) + O(1)$. [Hint: Obviously, $KS(x, KS(x)) \geqslant KS(x) + O(1)$. On the other hand, the shortest description of x determines both x and $KS(x)$.] {complexity

2.2 Conditional complexity

{condit-c}

When transmitting a file, one could try to save communication charges by compressing it. The transmission could be made even more effective if and old version of the same file already exists at the other side. In this case we need only to describe the changes made. This could be considered as a kind of motivation for the definition of conditional complexity of a given string x relative to (known) string y.

A *conditional decompressor* is any computable function D of two arguments (both arguments and the value of D are binary strings). If $D(y,z) = x$ we say that y is a (conditional) *description of x when z is known* (or *relative to z*) The complexity $KS_D(x|z)$ is then defined as the length of the shortest conditional description:

$$KS_D(x|z) = \min\{l(y) \mid D(y,z) = x\}.$$

We say that (conditional) decompressor D_1 is *not worse* than D_2 if

$$KS_{D_1}(x|z) \leqslant KS_{D_2}(x|z) + c$$

for some constant c and for all x and z. A conditional decompressor is *optimal* if it is not worse than any other conditional decompressor.

{condit-uni

Theorem 17 *There exist optimal conditional decomressors.*

$\lhd$ This "conditional" version of Kolmogorov–Solomonoff theorem can be proved in the same way as the unconditional one (Theorem 1, p. 6).

Fix some programming language where one can write programs for computable functions of two arguments, and let

$$D(\hat{p}y,z) = p(y,z),$$

where $p(y,z)$ is the output of program p on inputs y and z, and $\hat{p}$ is the prefix-free encoding of p.

It is easy to see now that if D' is a conditional decompressor and p is a program for D', then

$$KS_D(x|z) \leqslant KS_{D'}(x|z) + l(\hat{p}).$$

Theorem is proved. $\rhd$

Again, we fix some optimal conditional decompressor D and omit index D in the notation.

Several easy facts about conditional Kolmogorov complexity:

{condit-bas

Theorem 18

$$\begin{aligned} KS(x|y) &\leqslant KS(x) + O(1); \\ KS(x|x) &= O(1); \\ KS(f(x,y)|y) &\leqslant KS(x|y) + O(1); \\ KS(x|y) &\leqslant KS(x|g(y)) + O(1). \end{aligned}$$

Here g and f are arbitrary computable functions (of one and two arguments, respectively); the inequalities are valid if $f(x,y)$ and $g(y)$ are defined.

$\lhd$ First inequality: any unconditional description mode can be considered as a conditional mode that ignores the second argument.

The second inequality: consider the conditional description mode D such that $D(p,z) = z$.

Third inequality: Let D be the optimal conditional description mode used in the complexity definition. Consider another description mode D' such that

$$D'(p,y) = f(D(p,y),y)$$

and apply the optimality property.

Similar argument works for the last inequality, but D' should be defined in a different way:

$$D'(p,y) = D(p,g(y)).$$

Theorem is proved. $\rhd$

20 Prove that conditional complexity is "continuous as a function of its second argument": $KS(x|y0) = KS(x|y) + O(1)$; $KS(x|y1) = KS(x|y) + O(1)$.

21 Prove that for any fixed y the function $x \mapsto KS(x|y)$ differs from KS at most by $2KS(y) + O(1)$.

22 Prove that $KS([x,z]|[y,z]) \leqslant KS(x|y) + O(1)$ for any strings x,y,z (here $[\cdot,\cdot]$ stands for the computable encoding of pairs).

23 Fix some "reasonable" programming language. (Formally, we require the corresponding universal function to be a Gödel one; this means that translation algorithm exists for any other programming language, see, e.g., [**?**].) Show that conditional complexity $KS(x|y)$ is equal (up to $O(1)$ additive term) to the minimal complexity of a program that produces output x on input y. [Hint: Let D be an optimal conditional decompressor. If we fix its first argument p, we get a program of complexity at most $l(p) + O(1)$. On the other hand, if program p maps y to x, then $KS(x|y) = KS(p(y)|y) \leqslant KS(p) + O(1)$.] {conditiona

This interpretation of conditional complexity as a minimal complexity of a program with some property will be considered in Chapter **??**.

Many properties of unconditional complexity have conditional counterparts with essentially the same proofs. Here are some of these counterparts:

- Function $KS(\cdot|\cdot)$ is upper semicomputable (this means that the set of triples $\langle x,y,n\rangle$ such that $KS(x|y) < n$ is enumerable).
- For any y and n the set of all strings x such that $KS(x|y) < n$ has cardinality less then 2^n.
- For any y and n there exists a string x of length n such that $KS(x|y) \geqslant n$.

24 Prove that for any strings y and z and for any number n there exists a string x of length n such that $KS(x|y) \geqslant n-1$ $KS(x|z) \geqslant n-1$. [Hint: both requirements are violated by a minority of strings.]

{condit-ub}

Theorem 19 *Let $\langle x,y\rangle \mapsto K(x|y)$ be an upper semicomputable function such that the set*

$$\{x \mid K(x|y) < n\}$$

has cardinality less than 2^n for any string y and integer n. Then $KS(x|y) \leqslant K(x|y) + c$ for some c and for all x and y.

Using conditional complexity, we get a stronger inequality for the complexity of pairs (compared with Theorem 16, p. 30):

{condit-pai

Theorem 20

$$KS(x,y) \leqslant KS(x) + 2\log KS(x) + KS(y|x) + O(1)$$

$\lhd$ Let D_1 be an optimal unconditional decompressor and let D_2 be an optimal conditional decompressor. Construct a new unconditional decompressor D' as follows:

$$D'(\hat{p}q) = [D_1(p), D_2(q, D_1(p))].$$

Here $\hat{p}$ stands for the prefix-free encoding of p, and $[\cdot,\cdot]$ is a computable encoding of pairs used in the definition of pair complexity. Let p be the shortest D_1-description of x and q be the shortest D_2-description of y conditional to x. Then the string $\hat{p}q$ is a D_3-description of $[x,y]$. Therefore,

$$KS(x,y) \leqslant KS_{D'}(x,y) + O(1) \leqslant l(\hat{p}) + l(q) + O(1).$$

As we have seen, one can choose a prefix-free encoding in such a way that $l(\hat{p}) \leqslant l(p) + 2\log l(p) + O(1)$ (see the proof of Theorem 16, p. 30), and we get a desired inequality. $\rhd$

As before, we can improve the bound by replacing $2\log KS(x)$ by $\log KS(x) + 2\log\log KS(x)$ etc. We also can use conditional complexity in the additional term and write

$$KS(x,y) \leqslant KS(x) + KS(y|x) + 2\log KS(y|x) + O(1).$$

(In the proof we should replace $D'(\hat{p}q)$ by $D'(\hat{q}p)$.)

25 Prove that

$$KS(x|z) \leqslant KS(x|y) + 2\log KS(x|y) + KS(y|z) + O(1)$$

for all x,y,z (a sort of a "triangle inequality").

If we are not interested in the exact form of the additional logarithmic term, the statement of Theorem 20 can be reformulated as follows:

$$KS(x,y) \leqslant KS(x) + KS(y|x) + O(\log n).$$

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

It turns out (and this is the first nontrivial statement in this chapter) that this inequality is in fact an equality:

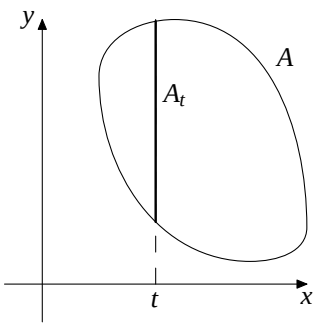

Figure 2: The section A_t of the set A of all simple pairs.

{condit-c.1

{condit-pai

Theorem 21 (Kolmogorov – Levin)

$$KS(x,y) = KS(x) + KS(y|x) + O(\log n).$$

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

$\lhd$ Since we already have one inequality, we need to prove only that $KS(x,y) \geqslant KS(x) + KS(y|x) + O(\log n)$ for all x and y of length at most n.

Let x and y be some strings of length at most n. Let a be the complexity $KS(x,y)$ of the pair $\langle x,y\rangle$. Consider the set A of all pairs whose complexity does not exceed a. Then A is a set of cardinality $O(2^a)$ (in fact, at most 2^{a+1}) and $\langle x,y\rangle$ is one of its elements.

For each string t consider the "vertical section" A_t of A:

$$A_t = \{u \mid \langle t,u\rangle \in A\}$$

(Fig. 2). The sum of the cardinalities of all A_t (over all strings t) is the cardinality of A and does not exceed $O(2^a)$. Therefore there are few "large" sections A_t, and this is the basic argument we need for the proof.

Let m be equal to $\lfloor \log_2 |A_x| \rfloor$ where x is the first component of the pair $\langle x,y\rangle$ we started with. In other terms, assume that cardinality of A_x is between 2^m and 2^{m+1}. Let us prove that

(1) $KS(y|x)$ does not exceed m significantly;

(2) $KS(x)$ does not exceed $a-m$ significantly.

We start with (1). Knowing a, we can enumerate the set A. If we know also x, we can select only pairs whose first component equals x. In this way we get an enumeration of A_x. To specify y, it is enough to tell the ordinal number of y in this enumeration (of A_x). This ordinal number takes $m + O(1)$ bits, and together with a we get $m + O(\log n)$ bits for the conditional description of y given x. Note that $a = KS(x,y)$ does not exceed $O(n)$ for strings x and y of length n. Therefore, we need only $O(\log n)$ to specify a and n, and

$$KS(y|x) \leqslant m + O(\log n).$$

Now let us prove (2). Consider the set B of all strings t such that cardinality of A_t is at least 2^m. The cardinality of B does not exceed $2^{a+1}/2^m$, otherwise the sum $|A| = \sum |A_t|$ would be greater

than 2^{a+1}. We can enumerate B if we know a and m. Indeed, we should enumerate A and group together the pairs with the same first coordinate; if we find 2^m pairs with the same value of the first coordinate, we put this value into B. Therefore the string x (as well as any element of B) can be specified by $(a-m)+O(\log n)$ bits: $a-m+1$ bits are needed for ordinal number of x in the enumeration of B, and $O(\log n)$ are used to specify a and m. So we get

$$KS(x) \leqslant (a-m)+O(\log n),$$

and it remains to add this inequality and the preceding one. $\triangleright$

This theorem can be considered as the complexity counterpart of the following combinatorial statement. Let A be a finite set of pairs, and p and q be some numbers such that cardinality of A does not exceed pq. Then we can split A into parts P and Q with the following properties: the projection of P onto the first coordinate has at most p elements, while all the sections Q_x of Q (the first coordinate equals x) have at most q elements. (Indeed, let P be the union of all sections that have more than q elements. The number of such sections do not exceed p. Remaining elements form Q.) We return to this combinatorial translation in Chapter **??**.

Note that in fact we have not used the lengths of x and y, only their complexities. So we have proved the following statement:

{condit-pai

Theorem 22 (Kolmogorov – Levin, complexity version)

$$KS(x,y) = KS(x)+KS(y|x)+O(\log KS(x,y))$$

for all strings x and y.

26 Give a more detailed analysis of the additive terms in the proof and show that

$$KS(x)+KS(y|x) \leqslant KS(x,y)+3\log KS(x,y)+O(\log\log KS(x,y)).$$

27 Show that $O(\log n)$ terms are unavoidable in Kolmogorov–Levin theorem in both directions: for each n there exist strings x and y of length at most n such that

$$KS(x,y) \geqslant KS(x)+KS(y|x)+\log n-O(1),$$

as well as strings x and y of length at most n such that

$$KS(x,y) \leqslant KS(x)+KS(y|x)-\log n+O(1).$$

[Hint: For the first inequality we can refer to the remark after Theorem 16 (p. 30). For the second one we can take as x some number between $n/2$ and n such that $KS(x) = \log n+O(1)$ and the let y be a string of length x such that $KS(y|x) = x+O(1)$.]

28 Prove that changing one bit in a string of length n changes its complexity at most by $\log n+O(\log\log n)$.

29 Fix some unconditional decompressor D. Prove that for some constant c and for all integers n and k the following statement is true: if some string x has at least 2^k descriptions of length at most n, then $KS(x|k) \leqslant n-k+c$. [Hint: Fix some k. For each n consider all strings x that have at least 2^k descriptions of length at most n. The number of these strings does not exceed 2^{n-k}, and we can apply Theorem 19, p. 35.] {number-of-

Using this problem, we can prove the following statement about unconditional complexity:

30 Let D be some optimal unconditional decompressor. Then there exists some constant c such that for any string x the number of shortest D-descriptions of x does not exceed c. [Hint: The previous problems show that $KS(x) \leqslant n-k+2\log k+O(1)$, so for $KS(x)=n$ we get an upper bound for k.]

31 Prove that there exists a constant c with the following property: if for some x and n the probability of the event $KS(x|y) \leqslant k$ (all strings y of lengths n are considered as equiprobable here) is at least 2^{-l}, then $KS(x|n,l) \leqslant k+l+c$. [Hint: Connect each string y of length n to all strings x such that $KS(x|y) \leqslant k$. We get a bipartite graph that has $O(2^{n+k})$ edges. In this graph the number of vertices x that have degree at least 2^{n-l} does not exceed $O(2^{k+l})$. Note that $KS(x|n,l)$ does not include k—this is not a typo!]

This problem could help us to find the average value of $KS(x|y)$ for given x and all string y of some length n. It is evident that $KS(x|y) \leqslant K(x|n)+O(1)$ since $n=l(y)$ is determined by y. It turns out that for most strings y (of given length) this inequality is close to an equality:

32 Prove that there exists some constant c such that for any string x and for all natural numbers n and d the fraction of strings y such that $KS(x|y) < KS(x|n)-d$ (among all strings of length n) does not exceed $cd^2/2^d$. Using this statement, prove that the average value of $KS(x|y)$ taken over all strings y of a given length n equals $KS(x|n)+O(1)$ (the constant in $O(1)$ does not depend on x and n).

33 Prove that $KS(x) = KS(x|\,KS(x)) + O(1)$. [Hint: Assume that x has a conditional description q with condition $KS(x)$ that is shorter than $KS(x)$. Then one can specify x by providing q and the difference $KS(x)-l(q)$, and we get a description of x that is shorter than $KS(x)$—a contradiction.]

34 Prove that for some constant c for any string x and for every number n there exists a string y of length n such that {increasing

$$KS(xy) \geqslant KS(x|n)+n-c.$$

[Hint: For a given n the number of strings x such that $KS(xy) < k$ for any y of length n, does not exceed $2^k/2^n$, and this property is enumerable. So we can apply Theorem 19 (p. 35).]

35 Prove that an infinite sequence $x_0x_1x_2\ldots$ of zeros and ones is computable if and only if the set $KS(x_0\ldots x_{n-1}|n)$ (the complexities of its prefixes conditional to their lengths) is bounded by a constant.

[Hint: Consider an infinite binary tree. Let S be the enumerable set of vertices (binary strings) that have conditional complexity (w.r.t. their length) less than some constant c. The "horizontal" sections of S have cardinality $O(1)$. We need to derive from this that each infinite path that lies

entirely inside S, is computable. We may assume that S is a subtree (only strings whose prefixes are in B, remain in S).

Let ω be an infinite path that goes through S only. At each level n we count vertices in S on the left of ω (l_n vertices) and on the right of ω (r_n vertices). Let $L = \limsup l_n$ and $R = \limsup r_n$. Let N be the level such that L and R are never exceeded after this level. Knowing L, R and N we can compute arbitrarily large prefixes of ω. We should look for a path π in a tree such that at some level above N there are at least L elements of S on the left of π, and at some (possibly other) level above N there are at least R elements on the right of π. When such a path π is found, we can be sure that its initial segment (up to the first of those two levels) coincides with ω.]

36 Prove that in the previous problem a weaker assumption is sufficient: instead of $KS(x_0\ldots x_{n-1}|n) = O(1)$ we can require that $KS(x_0\ldots x_{n-1}) \leqslant \log n + c$ for some c and for all n. [Hint: In this case we get an enumerable set S of strings (=tree vertices) with the following property: the number of vertices on all levels below N is $O(N)$. This means that the average number of vertices per level is bounded by a constant. To use the previous problem, we need a bound for all levels and not for the average value. We can achieve this if we consider only vertices $x \in S$ that have a extension of length $2l(x)$ that goes entirely inside S.]

37 Consider strings of length n that have complexity at least n (*incompressible* strings).

(**a**) Prove that the number of incompressible strings of length n is between 2^{n-c} and $2^n - 2^{n-c}$ (for some c and for all n)

(**b**) Prove that the cardinality of the set of incompressible strings of length n has complexity $n - O(1)$ (note that this implies the statement (a));

(**c**) Prove that if the string x of length $2n$ is incompressible, then its halves x_1 and x_2 (of length n) have complexity $n - O(1)$.

(**d**) Prove that if a string x of length n is incompressible, then each its substring of length k has complexity at least $k - O(\log n)$.

(**e**) Prove that for any constant $c < 1$ all incompressible strings of sufficiently large length n contain a substring of $\lfloor c \log_2 n \rfloor$ zeros.

[Hints: (a) There is at most $2^n - 1$ descritions of length les than n. and part of them is used for shorter strings: any string of length $n - d$ (for some d) has complexity less than n. This gives a lower bound for the number of uncompressible strings. To prove the upper bound, note that strings of length n that have prefix of k zeros, could be described by $2\log k + (n - k)$ bits.

(b) Let t be the number of incompressible string written in binary. If t has $n - k$ bits, then knowing t and $\log k$ additional bits we can reconstruct first n and then the list of all incompressible strings of length n, so the first incompressible string has complexity less than n, which is impossible.

(c) If one part of the string is has a short description, the entire string has a short description that starts with prefix-free encoding of the difference between the length and complexity of the compressible part.

(d) If a string has a simple substring, then the entire string can be compressed (we need to specify the substring, its position and the rest of the string).

(e) Let us count the number of strings of length n that do not contain k zeros in a row; a recurrent relation shows that this number grows like a geometric sequence whose base is the maximal real

root of the equation $x = 2 - (1/x^k)$, and we can get a bound for complexity of strings that do not have k zeros in a row.]

38 Prove that (for some constant c) for any infinite sequence $x_0x_1x_2\ldots$ of zeros and ones there exist infinitely many n such that $KS(x_0x_1\ldots x_{n-1}) \leqslant n - \log n + c$. {condit-c-s

Prove that there is a constant c and the sequence $x_0x_1x_2\ldots$ such that $KS(x_0x_1\ldots x_{n-1}) \geqslant n - 2\log n - c$ for all n. [Hint: The series $\sum 1/n$ diverges while the series $\sum(1/n^2)$ converges. For details see Theorem 87 and **??**.]

39 For a string x of length n let us define $d(x)$ and $d_c(x)$ as follows: $d(x) = n - KS(x)$ and $d_c(x) = n - KS(x|n)$. Show that they are rather close to each other: {conditiona

$$d_c(x) - 2\log d_c(x) - O(1) \leqslant d(x) \leqslant d_c(x) + O(1).$$

[Hint: We need to show that if $KS(x|n) = n - d$, then $KS(x) \leqslant n - d + 2\log d + O(1)$. Indeed, let us take the conditional description of x of length $n - d$ and put it after the self-delimiting description of d that has size $2\log d + O(1)$. Knowing this string, we can reconstruct d, then n and finally x.]

(The intuitive meaning of the difference between the length of a string and its complexity is discussed in Chapter 5 and Chapter **??**.)

2.3 Complexity as the amount of information

{conditi}

As we know (Theorem 18), the conditional complexity $KS(y|x)$ does not exceed the unconditional one $KS(y)$ (up to a constant). The difference $KS(y) - KS(y|x)$ tells us how much the knowledge of y makes x easier to describe. So this difference can be called the *amount of information in x about y*. Notation: $I(x:y)$.

Theorem 18 says that $I(x:y)$ is non-negative (up to a constant): there exists some c such that $I(x:y) \geqslant c$ for all x and y.

Recall that

$$KS(x,y) = KS(x) + KS(y|x) + O(\log KS(x,y)),$$

(Theorem 22, p. 37). This allows us to express conditional complexity in terms of unconditional one: $KS(y|x) = KS(x,y) - KS(x) + O(\log KS(x,y))$. Then we get the following expression for the information:

$$I(x:y) = KS(y) - KS(y|x) = KS(x) + KS(y) - KS(x,y) + O(\log KS(x,y)).$$

This expression immediately implies the following theorem:

{condit-sym

Theorem 23 (information symmetry)

$$I(x:y) = I(y:x) + O(\log KS(x,y))$$

So the difference between $I(x:y)$ and $I(y:x)$ is logarithmically smaller than $KS(x,y)$. The following problem shows that at the same time this difference could be comparable with the values $I(x:y)$ and $I(y:x)$ if they are much less than $KS(x,y)$.

40 Let x be a string of length n such that $KS(x|n) \geqslant n$. Show that $I(x:n) = KS(n) + O(1)$ and $I(n:x) = O(1)$.

The property of information symmetry (up to a logarithmic term) explains why $I(x:y)$ (or $I(y:x)$) is sometimes called *mutual information* in two strings x and y. The connection between mutual information, conditional and unconditional complexities and pair complexity can be illustrated by a (rather symbolic) picture (Fig. 3).

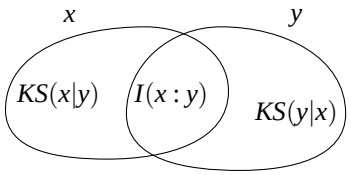

Figure 3: Mutual information and conditional complexity {condit-i.1

It shows that strings x and y have $I(x:y) \approx I(y:x)$ bits of mutual information. Adding $KS(x|y)$ bits (information that is present in x but absent in y, the left part), we obtain

$$I(y:x) + KS(x|y) \approx (KS(x) - KS(x|y)) + KS(x|y) \approx KS(x)$$

bits (matching the complexity of x). Similarly, the central part together with $KS(y|x)$ (the right part) give $KS(y)$. Finally, all three parts together give us

$$KS(x|y) + I(x:y) + KS(y|x) = KS(x) + KS(y|x) = KS(x|y) + KS(y) = KS(x,y)$$

bits (all equalities are true up to $O(\log n)$ for strings x and y of length at most n).

In some cases this picture can be understood quite literally. Consider, for instance, an incompressible string $r = r_1 \ldots r_n$ of length n such that $KS(r_1 \ldots r_n) \geqslant n$. Then any substring u of x has complexity $l(u)$ up to $O(\log n)$ terms. Indeed, since u is a substring of r, we have $r = tuv$ for some strings t, v. Then $l(r) = KS(r) \leqslant KS(t) + KS(u) + KS(v) \leqslant l(t) + l(u) + l(v) = l(r)$ (up to a logarithmic error) and therefore all the inequalities are equalities (with the same logarithmic precision).

Now take two overlapping substrings x and y (Fig. 4). Then $KS(x)$ is the length of x, $KS(y)$ is the length of y (up to $O(\log n)$).

The complexity $KS(x,y)$ is equal to the length of the union of segments (since the pair $\langle x,y\rangle$ is equivalent to this union plus information about lengths, which is of size $O(\log n)$).

Therefore, conditional complexities $KS(x|y)$, $KS(y|x)$ and the mutual information $I(x:y)$ are equal to the lengths of the corresponding segments (up to $O(\log n)$).

However, not always the mutual information can be extracted in form of some string (like it happened in our example, where this common information in just the intersection of strings x and y). As we will see in Chapter **??**, there exist two strings x and y that have large mutual information

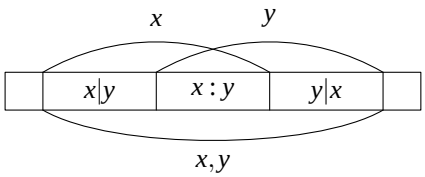

Figure 4: Common information in overlapping substrings {condit-i.2

$I(x:y)$ but there is no string z that represents ("materializes") this information in the following sense: $K(z|x) \approx 0$, $KS(z|y) \approx 0$ (all information that is present in z is also present both in x and in y) and $KS(z) \approx I(x:y)$ (all mutual information is extracted). In our last example we can take the intersection substring for z.

41 Prove that for any string x of length at most n the expected value of the mutual information $I(x:y)$ in x and random string y of length n is $O(\log n)$.

Now we move to triples of strings instead of pairs. Here we have an important tool that can be called *relativization*: most of the results proved for unconditional complexities remain valid for conditional complexities (and proofs remain valid with minimal changes). Let us give some example of this type.

A theorem about the complexity of pairs (p. 30) says that $KS(x,y) \leqslant KS(x) + 2\log KS(x) + KS(y) + O(1)$. Replacing all complexities by conditional ones (with the same condition z in all cases), we get the following inequality:

$$KS(x,y|z) \leqslant KS(x|z) + 2\log KS(x|z) + KS(y|z) + O(1),$$

By conditional complexity of a pair x,y relative to z we mean, as one can expect, the conditional complexity of its encoding: $KS(x,y|z) = KS([x,y]|z)$. As for unconditional complexity, the choice of encoding is not important (the complexity changes by $O(1)$).

The proof of this relativized inequality repeats the proof of the unrelativized one: we combine description p for x (with condition z) and description q for y (with condition z) into a string $\hat{p}q$ which is a description of $[p,q]$ (with condition z) relative to some suitable conditional decompressor.

So this is nothing really new. However, we may express all the conditional complexities in terms of unconditional ones: recall that $KS(x,y|z) = KS(x,y,z) - KS(z)$ and $KS(x|z) = KS(x,z) - KS(z)$, $KS(y|z) = KS(y,z) - KS(z)$ (with logarithmic precision). Then we get the following theorem:

{condit-bas

Theorem 24

$$KS(x,y,z) + KS(z) \leqslant KS(x,z) + KS(y,z) + O(\log n)$$

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

Sometimes this inequality is called the *basic* inequality for complexities.

The same relativization can be applied to Theorem 21 (p. 36) that relates the complexity of a pair and conditional complexity. Then we get the following statement:

Theorem 25

$$KS(x,y|z) = KS(x|z) + KS(y|x,z) + O(\log n),$$

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

$\triangleleft$ We can follow the proof of theorem 21, replacing unconditional descriptions by conditional ones (with z as the condition). Doing this, we replace $KS(y|x)$ by $KS(y|x,z)$. One can say that now we work in three-dimensional space with coordinates x,y,z and apply the same arguments simultaneously in all planes parallel to xy plane.

If this argument does not look convincing for you, there is a more formal one. Express all the conditional complexities in terms of unconditional ones:

$$KS(x,y|z) = KS(x,y,z) - KS(z),$$

and for the right-hand side

$$KS(x|z) + KS(y|x,z) = KS(x,z) - KS(z) + KS(y,x,z) - KS(x,z).$$

We see that both sides coincide (up to $O(\log n)$). (A pedantic reader may note that this simplified argument gives larger hidden constants in $O(\log n)$-notation.) $\triangleright$

42 Proof that in Theorem 25 a weaker assumption "$KS(x|z)$ and $KS(y|x,z)$ do not exceed n" is sufficient.

We also relativize the definition of mutual information and let $I(x:y|z)$ be the difference $KS(y|z) - KS(y|x,z)$. As for the case of (unconditional) information, this quantity is non-negative (up to $O(1)$ precision). Replacing conditional complexities by the expressions involving unconditional ones (with logarithmic precision), we can rewrite the inequality $I(x:y|z) \geqslant 0$ as follows:

$$KS(y|z) - KS(y|x,z) = KS(y,z) - KS(z) - KS(y,x,z) + KS(x,z) \geqslant 0.$$

So we get the basic inequality of Theorem 24 again.

In fact, almost all known equalities and inequalities that involve complexities (unconditional and conditional) and information (and have logarithmic precision) are immediate consequences of Theorems 21 and 24. Let us give two examples of this type.

Independent strings. We say that strings x and y are "independent" if $I(x:y) \approx 0$. We need to specify what we mean by "$\approx$", but we always ignore the terms of order $O(\log n)$ where n is the maximal length (or complexity) of the strings involved.

Independent strings could be considered as some counterpart of the notion of independent random variables, which is crucial in the probability theory. There is a simple observation: if a random variable ξ is independent with the pair of random variables $\langle \alpha, \beta \rangle$, then ξ is independent with α and with β (separately).

The Kolmogorov complexity counterpart of this statement (if a string x is independent with a pair $\langle y,z \rangle$, then x is independent with y and x is independent with z) can be expressed as an inequality:

$$I(x:\langle y,z \rangle) \geqslant I(x:y)$$

(and the similar inequality for z instead of y). This inequality is indeed true (with logarithmic precision), and it is easy to see if we rewrite it in terms of unconditional complexities:

$$KS(x) + KS(y,z) - KS(x,y,z) \geqslant KS(x) + KS(y) - KS(x,y),$$

which after cancellation of similar terms gives a basic inequality (Theorem 24).

Complexity of pairs and triples. On the other hand, to prove the following theorem (which we have already mentioned on p.15), it is convenient to replace unconditional complexities by conditional ones:

{condit-tri

Theorem 26

$$2KS(x,y,z) \leqslant KS(x,y) + KS(x,z) + KS(y,z) + O(\log n),$$

for all strings x,y,z of complexity at most n

$\lhd$ Moving $KS(x,y)$ $KS(x,z)$ to the left-hand side and replacing $KS(x,y,z) - KS(x,y)$ and $KS(x,y,z) - KS(x,z)$ by conditional complexities $KS(z|x,y)$ $KS(y|x,z)$, we get the following inequality:

$$KS(z|x,y) + KS(y|x,z) \leqslant K(y,z) + O(\log n).$$

It remains to rewrite the right-hand side as $KS(y) + KS(z|y)$, and note that $KS(z|x,y) \leqslant KS(z|y)$ and $KS(y|x,z) \leqslant KS(y)$. $\rhd$

Instead we could just add two inequalities (the basic one and the inequality for the complexity of a pair):

$$KS(x,y,z) + KS(y) \leqslant KS(x,y) + KS(y,z) + O(\log n),$$
$$KS(x,y,z) \leqslant KS(y) + KS(x,z) + O(\log n),$$

and then cancel $KS(y)$ in both sides. (This proof, as well as the previous one, have an important esthetic problem: both treat x,y,z in a non-symmetric way while the statement of the theorem is symmetric.)

We return to the inequality of Theorem 26 and to its geometric consequences in Chapter **??**.

We can provide a more systematic treatment of the different complexity quantities related to three strings as follows. There are seven basic quantities: three of them are complexities of individual strings, another three are complexities of pairs and one more is the complexity of the entire triple. Other quantities such that conditional complexity and mutual information can be expressed in terms of these seven complexities. To understand better what conditions these seven quantities should satisfy, let us make a linear transformation in the 7-dimensional space and switch to new coordinates. Consider seven variables $a_1, a_2, \ldots, a_7$ that correspond to 7 regions shown in Fig. 5.

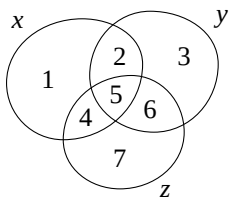

Figure 5: New coordinates $a_1, a_2, \ldots, a_7$. {condit-i.3

Formally, the coordinate transformation is given by the following equations:

$$\begin{aligned}
KS(x) &= a_1 + a_2 + a_4 + a_5,\\
KS(y) &= a_2 + a_3 + a_5 + a_6,\\
KS(z) &= a_4 + a_5 + a_6 + a_7,\\
KS(x,y) &= a_1 + a_2 + a_3 + a_4 + a_5 + a_6,\\
KS(x,z) &= a_1 + a_2 + a_4 + a_5 + a_6 + a_7,\\
KS(y,z) &= a_2 + a_3 + a_4 + a_5 + a_6 + a_7,\\
KS(x,y,z) &= a_1 + a_2 + a_3 + a_4 + a_5 + a_6 + a_7.
\end{aligned}$$

Indeed, it is easy to see that these equations determine an invertible linear transformation of $\mathbb{R}^7$: each 7-tuple of complexities corresponds to unique value of variables $a_1, \ldots, a_7$.

Conditional complexities and mutual informations could be expressed in terms of complexities and therefore could be rewritten in new coordinates. For example, $I(x:y) = KS(x) + KS(y) - KS(x,y) = a_2 + a_5$ and $KS(x|y) = KS(x,y) - KS(y) = a_1 + a_4$.

What is the intuitive meaning of these new coordinates? It is easy to see that $a_1 = KS(x|y,z)$ (with logarithmic precision). The meaning of a_3 (and a_7) is similar. The coordinate a_2 is (with the same precision) $I(x:y|z)$; coordinates a_4 and a_6 have similar meaning (see Fig. 6). In particular, we conclude that for any strings x,y,z the corresponding values of coordinates $a_1, a_2, a_3, a_4, a_6, a_7$ are non-negative (up to $O(\log n)$ for strings x,y,z of complexity at most n).

The coordinate a_5 is more delicate. Informally, we would like to understand it as the "amount of common information in three strings x,y,z". Sometimes the notation $I(x:y:z)$ is used. However, the meaning of this expression is not quite clear, especially if we take into accout that a_5 can be negative.

Consider the following example where $a_7 < 0$. Let x and y be two halves of an incompressible strings of length $2n$ Then $KS(x) = n$, $KS(y) = n$, $KS(x,y) = 2n$ and $I(x:y) = 0$ (up to $O(\log n)$). Consider a string z of length n which is a bitwise sum modulo 2 of x and y. Then each of the strings x,y,z can be reconstructed if two others are known; therefor the complexities of all pairs $KS(x,y), KS(y,z)$, $KS(x,z)$ are equal to $2n$ (again up to $O(\log n)$), and the complexity $KS(x,y,z)$ is also n. The complexity of z is equal to n (it can not be larger, since the length is n; on the other hand, it cannot be smaller, since z and y form a pair of complexity $2n$).

So we get the following values of $a_1, \ldots, a_7$ for this example (Fig. 7):

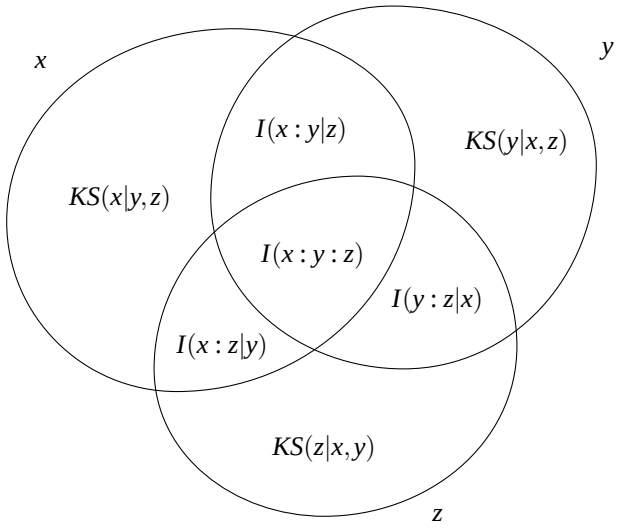

Figure 6: The complexity interpretation of new coordinates. {condit-i.5

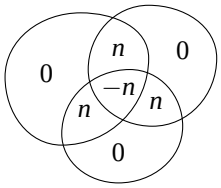

Figure 7: Two independent incompressible strings of length n and their bitwise sum mod 2. {condit-i.4

Note that even if a_5 is negative, the sums a_5+a_2, a_5+a_4 and a_5+a_6, being mutual informations for pairs, are non-negative. (In our examples these sums are equal to 0.)

This example corresponds to the simple case of secret sharing of secret z between two people: if one of them knows x and the other one knows y, then none of them has any information about z in isolation (since $I(x:z)\approx 0$ and $I(y:z)\approx 0$)), but together they can reconstruct z as a bitwise sum of x and y.

One can check that we have already given a full list of inequalities that are true for complexities of three strings and their combinations (all a_i, except for a_5, are non-negative, as well as three sums mentioned abovew). We return to this question in Chapter **??**.

Our diagram is a good mnenonic tool. For example, consider again the inequality

$$KS(x,y,z)\leqslant KS(x,y)+KS(x,z)+KS(y,z).$$

In our new variables it can be rewritten as $a_2+a_4+a_5+a_6\geqslant 0$ (you can easily check it by counting the multiplicity of each a_i in both sides of the inequality). It remains to note that $a_2+a_5\geqslant 0$, $a_4\geqslant 0$ and $a_6\geqslant 0$. (Alas, the symmetry is broken again!)

43 Prove that $I(xy:z)=I(x:z)+I(y:z|x)+O(\log n)$ for strings x,y,z of complexity at most n. [Hint: Use the diagram.]

This problem shows that infromation in xy about z can be somehow split into two parts: infromation in x about z and information in y about z (when x is known). This is somehow similar to the equality $KS(x,y) = KS(x) + KS(y|x)$, but now complexity is replaced by the quantity of information about z. As a corollary we immediately get that if xy is independent with z then x in independent with z and, at the same time y is independent with z when x is known. (Here independence means that mutual information is negligible.) A symmetric argument shows that x is independent with y and x is independent with z when y is known.

44 Show that properties "x is independent with y" and "x is independent with y when z is known" are quite different: any one of them can be true when the other one is false.

45 We say that strings x,y,z,t form a *Markov chain* (a well known notion in the probability theory now transferred to the algorithmic information theory) if $I(x : z|y)$ and $I(\langle x,y\rangle : t|z)$ are negligible. (Of course, we need to specify what is "negligible" to get a formal definition.) Show that the reversed sequence of strings also forms a Markov chain, i.e., that $I(t : y|z)$ and $I(\langle t,z\rangle : x|y)$ are negligible. [Hint: Since $I(\langle x,y\rangle : t|z) = I(y : t|z) + I(x : t|y,z)$, the left-hand side in this equality is zero if and only if both terms on the right-hand side are zero; and the second term in the right-hand side does not change when the order of x,y,z,t is reversed.]

3 Martin-Löf randomness

{random}

Here we interrupt the exposition of Kolmogorov complexity and its properties and define another basic notion used in the algorithmic information theory, i.e., the notion of Martin-Löf random (or typical) sequence. This chapter does not refer to the preceding one and is not used until Chapter 5 where we characterize randomness in terms of Kolmogorov complexity.

Let us remind some basic facts of measure theory for the case of infinite sequences of zeros and ones.

3.1 Measures on Ω

{randomcl}

Consider the set Ω whose elements are infinite sequences of zeros and ones. This set is called *Cantor space*. For a binary string x we consider a set Ω_x of all infinite sequences that have initial segment x. For example, Ω_{00} is the set of all sequences that start with two zeros, and $\Omega_\Lambda = \Omega$ (where Λ is an empty binary string).

The sets Ω_x are called *intervals*. All intervals and all unions of arbitrary families of intervals are called *open* subsets of Ω. In this way we get a topology on Ω, and this topology corresponds to a standard distance function on Ω defined as follows: the longer common prefix two sequences have, the smaller the distance between them is:

$$d(\omega,\omega') = 2^{-n},$$

where n is the smallest index such that $\omega_n \neq \omega'_n$. (Here ω_n stands for the nth term of the sequence $\omega = \omega_0\omega_1\omega_2\ldots$)

46 Prove that topological space Ω is homeomorphic to the Cantor set on the real line. (This set is obtained from $[0,1]$ by deleting the middle third, then the middle thirds of two remaining segments and so on.)

However, we are interested in measure theory rather than topology. A family of subsets of Ω is called a *σ-algebra* if it is closed under finite or countable unions and intersections and negation (taking the complement).

A minimal σ-algebra that contains all intervals Ω_x (and therefore all open sets) is called the algebra of *Borel* sets.

Consider an arbitrary σ-algebra that contains all intervals. Let μ be a function that maps every set in this σ-algebra into a non-negative real number, and has the following property (called *σ-additivity*):

> if a set A is a union of a countable or finite family of disjoint sets $A_0, A_1, A_2, \ldots$ that belong to the σ-algebra on which μ is defined, then
>
> $$\mu(A) = \mu(A_0) + \mu(A_1) + \mu(A_2) + \ldots$$
>
> (the right-hand side is a finite sum or a converging series with non-negative terms).

Then μ is called a *measure* on Ω, and the value $\mu(A)$ is called the measure of the set A.

A measure μ such that $\mu(\Omega) = 1$ is called a *probability distribution* on Ω. Elements of the σ-algebra that is the domain of μ are called *events*, and $\mu(A)$ is called the *probability* of the event A.

Any measure in monotone ($A \subset B$ implies $\mu(A) \leqslant \mu(B)$). Indeed, $\mu(B) - \mu(A) = \mu(B \setminus A) \geqslant 0$.

Another important property of measures is continuity: if a set B is a union of increasing sequence of sets

$$B_0 \subset B_1 \subset B_2 \subset \ldots,$$

then $\mu(B_n)$ tends to $\mu(B)$ as $n \to \infty$. (Indeed, let us apply the additivity property to all sets $A_i = B_i \setminus B_{i-1}$ and then to all sets A_i such that $i \leqslant n$.) The similar property holds for decreasing sequences of sets.

For any measure μ on Ω let us consider a function p defined of binary strings as follows:

$$p(x) = \mu(\Omega_x).$$

This function has non-negative real values and satisfies the following *additivity* property:

$$p(x) = p(x0) + p(x1)$$

for any string x. (Indeed, the interval Ω_x is the union of its two halves Ω_{x0} and Ω_{x1}, which are disjoint sets.)

As we know from measure theory (Lebesgue theorem), an inverse transition is possible. Namely, for any additive function p on binary strings that has non-negative real values, Lebesgue theorem provides a measure μ such that $\mu(\Omega_x) = p(x)$ for all binary strings x.

The measure provided by Lebesgue theorem has the following additional property: if $\mu(A) = 0$ for some set A and $B \subset A$, then $\mu(B)$ is defined (and therefore $\mu(B) = 0$). In the sequel we consider only measures that have this additional property.

We do not explain Lebesgue's construction here and refer the reader to any textbook in measure theory, e.g., [**?**, **?**]. However, let us recall the definition of sets having measure 0, since Martin-Löf definition of randomness uses its effective version.

Let p be an additive nonnegative real-valued function on strings. We call $p(x)$ the measure of the interval Ω_x. A subset $A \subset \Omega$ is a *null set* (a set *of measure* 0) if for any $\varepsilon > 0$ there exist a finite or countable family of intervals that cover A and have total measure at most ε.

In other words, a set A is a null set if there exists a function $\langle \varepsilon, i \rangle \mapsto x(\varepsilon, i)$ (first argument is a positive real, the second argument is a non-negative integer; values are binary strings) such that

- $A \subset \Omega_{x(\varepsilon,0)} \cup \Omega_{x(\varepsilon,1)} \cup \Omega_{x(\varepsilon,2)} \ldots$
- $p(x(\varepsilon,0)) + p(x(\varepsilon,1)) + p(x(\varepsilon,2)) + \ldots \leqslant \varepsilon$

for any positive ε. Note that the family of intervals can be finite, since we do not require the function x to be total (undefined values are skipped both in the union and in the sum).

Here are some simple but useful observations:

- The definition does not change if we restrict ourselves to rational values of ε (or even let $\varepsilon = 2^{-k}$ for integer k).

- Any subset of a null set is a null set.
- A finite or countable union of null sets is a null set. (Indeed, to cover the union by the family of intervals of total measure less than ε, we combine the coverings of its parts of measure less than $\varepsilon/2, \varepsilon/4, \varepsilon/8$ etc.).
- Assume that p is chosen in such a way that any singleton is a null set (it is equivalent to the following property: for any infinite sequence $\omega = \omega_0\omega_1\omega_2\ldots$ the limit of $p(\omega_0\ldots\omega_n)$ (as $n \to \infty$) equals 0). Then any finite or countable set is a null set.

A *uniform measure* on Ω assigns to each interval Ω_x the number $2^{-l(x)}$:

$$p(x) = 2^{-n} \text{ for all strings } x \text{ of length } n.$$

The uniform measure is closely related to the standard measure on $\mathbb{R}$ (or, more precisely, on $[0,1]$). Formally, the measure of a set $A \subset \Omega$ is equal to the measure of the set of reals whose binary expansions are elements of A. (In fact, the correspondence between binary fractions and reals in $[0,1]$ is not a bijection, since numbers of the form $k/2^l$ for integer k and l have two representations: e.g., $0.01111\ldots = 0.10000\ldots$. But this happens only for a countable family of reals and measure theory ignores this.)

Indeed, the reals whose binary expansions start with x, form an interval, and the length of this interval is just 2^{-n} where n is the length of x. This implies that for any interval $I \subset [0,1]$ the uniform measure of the sequences that represent reals in I is equal to the length of the interval I.

Probability theory describes the uniform distribution as the probability distribution for the outcomes of independent fair coin tossing. Indeed, for n independent fair coins all 2^n binary strings of length n appear with the same probability 2^{-n}. The set Ω_x is the event "a random sequence of zeros and ones starts with x", and this event has probability $2^{-l(x)}$.

Similarly, we can consider a biased coin, where coin tossing are still independent. The corresponding measure (probability distribution) is called *Bernoulli measure* (or *Bernoulli distribution*) with parameters q, p (probabilities of 0 and 1 respectively; we assume that $p, q \geqslant 0$ and $p+q=1$). {nonuniform

With respect to this distribution, the event "sequence ω starts with a string x" has probability $q^u p^v$ where u and v are the numbers of zeros and ones in x. In other terms, we consider a function

$$x \mapsto q^{u(x)} p^{v(x)}$$

where $u(x)$ and $v(x)$ stand for the numbers of zeros and ones in x, respectively. (It is easy to check that this function has the additivity property.)

3.2 The Strong Law of Large Numbers

{random-llr

To see all these notions in action, let us state and prove the so-called *Strong Law of Large Numbers*.

Fix some $p, q \geqslant 0$ such that $p+q=1$. Let A_p be the set of all infinite sequences $\omega_0\omega_1\omega_2\ldots$ of zeros and ones such that limit frequency of ones exists and is equal to p, i.e.,

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

Theorem 27 *The set A_p has measure 1 with respect to Bernoulli distribution with parameters p and q.*

In other terms, the complement of A_p, i.e., the set of all sequences that either do not have limit frequency at all or have a limit frequency different from p, is a null set (according to this distribution).

$\lhd$ We prove this theorem for the uniform case (i.e., for $p = q = 1/2$) by an explicit calculation. The general case is left as an exercise (see also Sect. **??**).

Let us consider first a finite number of coin tossings and fix some n. All binary strings of length n have the same probability. We claim that most of them have approximately $n/2$ ones. Assume that some threshold ε is fixed. How many sequences have more than $(1/2+\varepsilon)n$ ones? The answer can be found using the Pascal triangle: we have to sum up all the terms in the nth row starting from some point that is slightly on the right of the midpoint. In this part we have a decreasing sequence of less than n terms, so the sum in question is bounded by the first term multiplied by n. (We don't need to be very accurate in our bounds and ignore factors which are polynomial in n. So we can omit the factor n in our bound.)

The first term of the sum is the binomial coefficient

$$\frac{n!}{k!(n-k)!},$$

where k is the smallest integer not less than $(1/2+\varepsilon)n$. We use the Stirling's approximation:

$$m! = \sqrt{(2\pi + o(1))m}\left(\frac{m}{e}\right)^m,$$

where e is the base of natural logarithms. Ignoring polynomial (in n) factors and using the notation $u = k/n$, $v = (n-k)/n$, we get

$$\frac{n!}{k!(n-k)!} \approx \frac{(n/e)^n}{(k/e)^k((n-k)/e)^{n-k}} = \frac{n^n}{k^k(n-k)^{n-k}} \approx$$

$$\approx \frac{n^n}{(un)^{un}(vn)^{vn}} = \frac{1}{u^{un}v^{vn}} = 2^{H(u,v)n},$$

where

$$H(u,v) = -u\log u - v\log v.$$

The value $H(u,v)$ is called the *Shannon entropy* of the random variable that has two values whose probabilities are u v. (We study the Shannon entropy in Chapter 7.) Figure 8 shows the corresponding graph (note that $v = 1-u$). It is easy to check that $H(u,1-u)$ achieves its maximal value (equal to 1) only at $u = 1/2$.

Now we see that the number of binary strings of length n that have frequency of ones greater than $(1/2+\varepsilon)$ does not exceed $\mathrm{poly}(n)2^{H(1/2+\varepsilon,1/2-\varepsilon)n}$ and therefore is bounded by $2^{cn+o(n)}$, where c is some constant less than 1 (depending on ε). Therefore, the fraction of this strings

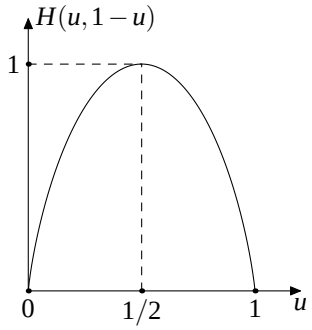

Figure 8: Shannon entropy as a function of u.

{randomcl-e

(among all strings of length n) exponentially decreases while n increases. The same is true for the strings that have frequency of ones less than $(1/2-\varepsilon)$.

Let us see where we are. For each fixed $\varepsilon > 0$ we have proved the following statement:

Lemma. The fraction of strings of length n where frequency of ones differs from $1/2$ at least by ε (among all strings of length n) does not exceed some δ_n that decreases exponentially as n increases.

This lemma (without any specific claims for the fast convergence $\delta_n \to 0$) is called the *Law of Large Numbers*. To prove the *Strong* Law of Large Numbers we need to know that the series $\sum_n \delta_n$ is convergent.

We need to prove that the set $A_{1/2}$ of all sequences that have limit frequency of ones equal to $1/2$ has measure 1. In other terms, we need to prove that the complement of this set (we denote this complement by B) is a null set.

According to the definition of limit the set B is the union (over all $\varepsilon > 0$) of the sets B_ε. Here B_ε is the set of all sequences such that frequency of ones in their prefixes exceeds $1/2+\varepsilon$ (or is less than $1/2-\varepsilon$) infinitely many times.

Evidently, we can consider only a countable set of different ε (e.g., only rational values), and the countable union of null sets is a null set. Therefore it remains to prove that the set B_ε is a null set for each ε.

The set B_ε consists of the sequences that have arbitrarily long "bad" prefixes. Here "bad" prefix is a string where the frequency of ones differs from $1/2$ more than by ε. Therefore, for each N the set B_ε is covered by the family of intervals Ω_x where x ranges over all bad strings of length at least N. The total (uniform) measure of all this intervals does not exceed

$$\delta_N + \delta_{N+1} + \delta_{N+2} + \ldots,$$

and this sum can be made small since the series $\sum_i \delta_i$ is convergent.

(Probability theorists call this argument *Borel–Cantelli lemma*. In its general form this lemma says that if the sum of measures of some sets $A_0, A_1, \ldots$ is finite, then the set of all points that belong to infinitely many A_i is a null set.) $\triangleright$

One can get a bound for the number of bad strings of length n without Stirling's approximation. We do it separately for bad strings that have too many and too few ones. For example, let us

consider the set of all "bad" strings that have frequency of ones greater than $1/2+\varepsilon$. To get a bound for the cardinality of this set, consider two distributions (measures) on the set of all strings of length n. The first one, called L, is the uniform distribution: all strings have probability 2^{-n}. The second one, called S, is biased (ones are more likely than zeros) and corresponds to n independent coin tossing where 1 appears with probability $p = 1/2+\varepsilon$. In other terms, $S(x) = q^u p^v$ for a string x that has u zeros and v ones (here $q = 1/2-\varepsilon$ is the probability of zero outcome). The ratio $S(x)/L(x)$ increases when the number of ones in x increases, and for all bad strings this ratio is at least $2^n/2^{H(p,q)n}$. Therefore, the total L-measure of all bad strings does not exceed their total S-measure divided by this expression. Recalling that the total S-measure of all bad strings does not exceed 1, we conclude that the total L-measure (i.e., the fraction) of all bad strings does not exceed $2^{H(p,q)n}/2^n$. So we get another proof of our bound, which is less technical (though more difficult to find). This proof works not only for the uniform Bernoulli measure ($p = 1/2$), but also for arbitrary p (after appropriate changes).

47 Prove the Strong Law of Large Numbers for arbitrary p. [Hint: Let p_0 and q_0 be fixed positive reals such that $p_0 + q_0 = 1$. Then the expression $-p_0 \log p - q_0 \log q$, where p, q are arbitrary positive reals such that $p+q = 1$, is minimal when $p = p_0$, $q = q_0$. See also Section **??**.] {non-unifor

People often say that "the Strong Law of Large Numbers guarantees that in any random (with respect to uniform Bernoulli measure) sequence the frequency of 1s tends to $1/2$". (The the case of nonuniform Bernoulli measures is similar.) However, in this sentence the word "random" shouldn't be understood literally: the phrase "any random sequence satisfies α" (for some condition α) is an idiomatic expression that means that the set of all sequences that do not satisfy α is a null set.

A natural question arises: can we define the notion of random sequence in such a way that this idiomatic expression can be understood literally? Let us fix some distribution on Ω, say, the uniform Bernoulli distribution. We would like to find some subset of Ω and call its elements "random sequences". Our goal would be achieved if for any condition α the following two statements were equivalent:

- all random sequences satisfy the condition α;
- the set of all sequences that does not satisfy α is a null set.

In other terms, the sets of measure 1 should be exactly those sets that contain all random sequences (and, may be, some nonrandom ones).

One more reformulation: the set of all random sequences should be the smallest (with respect to inclusion) set of measure 1, and the set of non-random sequences should be the largest (with respect to inclusion) null set. Now it easy to see that our goal cannot be achieved. Indeed, any singleton in Ω is a null set. However, the union of all these singletons is the entire space Ω.

In 1965 Per Martin-Löf (a Swedish mathematician, who was Kolmogorov's student at that time) found that we can save the game if we restrict ourselves to "effectively null sets". There exist a largest (with respect to inclusion) effectively null set, and therefore we can define the notion of a random sequence is such a way that any condition α is satisfied for all random sequences if and only if the set of all sequences that do not satisfy α is an *effectively* null set. Martin-Löf construction is explained in the next section.

3.3 Effectively null sets

{randomml}

Let a measure on Ω be fixed an let $p(x)$ be the measure of the interval Ω_x.

We say that a set $A \subset \Omega$ is an effectively null set (with respect to the given measure) if for every $\varepsilon > 0$ one can effectively find a family of intervals that cover A and whose total measure does not exceed ε.

Some details should be specified in this definition. First, we consider only rational values of ε (otherwise it is not clear how ε could be given to an algorithm). Second, we need to specify how the sequence of intervals (that cover A) is generated. We do this as follows:

Definition. A set $A \subset \Omega$ is called an *effectively null set* (with respect to a given measure) if there exists a computable function $x(\cdot,\cdot)$ whose first argument is a positive rational number, second argument is a natural number and values are binary strings, such that: {effective-

1. $A \subset \Omega_{x(\varepsilon,0)} \cup \Omega_{x(\varepsilon,1)} \cup \Omega_{x(\varepsilon,2)} \ldots$;
2. $p(x(\varepsilon,0)) + p(x(\varepsilon,1)) + p(x(\varepsilon,2)) + \ldots \leqslant \varepsilon$

for any rational $\varepsilon > 0$. Note that we do not require the function x to be total; if $x(\varepsilon,i)$ is undefined, the corresponding term (in both conditions) is omitted.

48 Show that we get an equivalent version of the definition if we consider an algorithm that gets $\varepsilon > 0$ as an input and enumerates a set of binary strings (by printing its elements with arbitrary delays between elements) such that intervals Ω_x for generated x cover A and have total measure at most ε.

49 Show that we get an equivalent definition if we consider only rational numbers of the form 2^{-k} (for integer k) instead of all rational ε. Show that the definition does not change if we replace the sign $\leqslant$ by $<$ in the second inequality.

50 Show that we get an equivalent definition if we require that for each $\varepsilon > 0$ the domain of the function $i \mapsto x(\varepsilon,i)$ is an initial segment of $\mathbb{N}$ (or $\mathbb{N}$ itself).

51 Show that we get an equivalent definition if we require that the family of intervals is decidable (instead of enumberable). [Hint: An interval can be split is small parts, so we may assume that intervals in the sequence have non-increasing length, and the family of intervals becomes decicalbe.]

Let us give some examples of effectively null subsets of Ω (with respect to the uniform measure).

A singleton whose only element is a sequence of zeros, is an effectively null set. Indeed, for every $\varepsilon > 0$ we find an integer k such that $2^{-k} < \varepsilon$, and consider a covering that consists of one interval $\Omega_{00\ldots0}$ (corresponding to the string of k zeros).

Formally speaking, $x(\varepsilon,0) = 0^k$, where 0^k stands for the sequence formed by k zeros, and k is the smallest integer such that $2^{-k} < \varepsilon$. The values $x(\varepsilon,i)$ are undefined for $i > 0$.

In this example the (identically) zero sequence can be replaced by any computable sequence of zeros and ones; we need only to consider its prefix of length k instead of 0^k.

However, we cannot replace it by any binary sequence, as the following problem shows:

52 Prove that there exists a sequence $\omega \in \Omega$ such that singleton $\{\omega\}$ is not an effectively null set. [Hint: Consider all computable functions x that satisfy the second condition of the definition of effectively null set. There are countably many such functions. For each of them consider the largest set A that satisfies the requirement (1) of the definition (i.e., the intersection of unions of coverings over all ε). This set is an (effectively) null set, and the union of a countable family of those sets is null set. Therefore, there exists a sequence ω which does not belong to this union.]

(Note that the statement of this problem is a straightforward corollary of the Martin-Löf theorem on the existence of the largest effectively null set (Theorem 28, p. 56)) proved later in this section, and the hint just follows the proof of the Martin-Löf theorem. As we see later, the set $\{\omega\}$ is an effectively null set if and only if the sequence ω is not "Martin-Löf random".)

It is easy to construct a non-computable sequence ω such that the singleton $\{\omega\}$ is an effective null set. Indeed, consider any sequence of the form $\omega = 0?0?0?0\ldots$ (each second term is zero, the rest is arbitrary). Let us show that $\{\omega\}$ is indeed an effectively null set. To find a covering with total measure 2^{-n}, consider all strings of length $2n$ that are formed by n arbitrary bits interleaved with n zeros (as in ω). There are 2^n strings of this form, and each corresponds to an interval of length 2^{-2n}, so the total measure is 2^{-n}.

In fact we have proved a bit more: the set of all sequences that have only zeros at even positions, is an effectively null set. Therefore, each of its subsets (in particular, every singleton) is an effectively null set.

Let us now return to the definition of an effectively null set and separate the requirements used in this definition. We say that a computable function x is "regular" if is satisfies the requirement (2). The requirement (1) then says that for every rational $\varepsilon > 0$ the set A is a subset of the union

$$\Omega_{x(\varepsilon,0)} \cup \Omega_{x(\varepsilon,1)} \cup \Omega_{x(\varepsilon,2)} \ldots$$

Therefore, a regular function "serves" all the subsets of the set

$$\bigcap_{\varepsilon>0} (\Omega_{x(\varepsilon,0)} \cup \Omega_{x(\varepsilon,1)} \cup \Omega_{x(\varepsilon,2)} \ldots) = \bigcap_{\varepsilon>0} \bigcup_i \Omega_{x(\varepsilon,i)}$$

So for each (computable) regular function x we get an effectively null set (defined by the formula above), and effectively null sets are all these sets (for all regular functions) and all their subsets, and that's all.

Before we formulate Martin-Löf theorem, let us give the definition of a *computable measure* on the set Ω.

A real number α is called *computable* if there exists an algorithm that computes rational approximations to α with any given precision. Formally, α is a computable real if there exists a computable function $\varepsilon \mapsto a(\varepsilon)$ defined on all positive rational numbers and having rational values such that

$$|\alpha - a(\varepsilon)| < \varepsilon$$

for all rational $\varepsilon > 0$.

53 Show that we get an equivalent definition if we additionally require that all approximation given by a are approximations from below, i.e., $a(\varepsilon) < \alpha$ for all ε. [Hint: we can transform any approximation to the approximation from below losing only factor 2 in precision.]

54 Prove that the sum, difference, product and quotient of two computable reals are computable reals.

55 Prove that e (the base of natural logarithms) and π are computable.

56 Prove that elementary function (roots, sine, exponent, logarithm etc.) preserve computability, i.e., have computable values for computable arguments. (We assume, of course, that the base is computable in case of logarithm and exponent.)

A measure μ on Ω is computable if measures of all intervals are computable reals, and, moreover, we can effectively find an approximation algorithm for $\mu(\Omega_x)$ given x. Here is a formal definition:

Definition. A measure μ on the set Ω is *computable* if there exists a computable function $\langle x, \varepsilon\rangle \mapsto a(x,\varepsilon)$, defined for all strings x and all positive rational numbers ε, such that

$$|\mu(\Omega_x) - a(x,\varepsilon)| < \varepsilon$$

for all x and ε.

This definition does not assume that the measure of the entire space Ω equals 1, but in fact we will use it only in this case (i.e., for probability distributions).

{randomml-n

Theorem 28 *Let μ be a computable measure on Ω. Then there exists a largest effectively null set with respect to μ. In other words, the union of all effectively μ-null sets is an effectively μ-null set.*

$\lhd$ As we have seen, for each regular function x we get a corresponding effectively null set. Since there is countably many regular functions, we get a countably many effectively null sets and their union contains every effectively null set. Therefore, the union of all effectively null sets is a null set. (When speaking about null sets and effectively null sets we have in mind measure μ.)

However, we need more: we have to prove that this union is an *effectively* null set. To achieve this goal, we enumerate all regular functions and then use the effective version of the theorem that says that the countable union of null sets is a null set.

For technical reasons it is convenient to change a bit the definition of a regular function. Namely, we now say that a computable function $x(\cdot,\cdot)$ is regular if all the finite partial sums of the series

$$p(x(\varepsilon,0)) + p(x(\varepsilon,1)) + p(x(\varepsilon,2)) + \ldots$$

are less than ε (note the strict inequality). Here $p(x)$ stands for $\mu(\Omega_x)$. This makes our requirements for regular functions a bit stronger (if all partial sums are less than ε, the sum of the series does not exceed ε, but the reverse is not always true). However, the notion of the effectively null set is not affected, since we always can replace ε by (say) $\varepsilon/2$.

In the sequel the regular functions are understood in this modified sense (in fact, regular functions are used only locally, in the proof of Martin-Löf theorem).

The following Lemma allows us to enumerate all regular functions.

Lemma. There exists a computable (partial) function

$$\langle q, \varepsilon, i\rangle \mapsto X(q,\varepsilon,i)$$

(where q and i are natural numbers, ε is a positive rational number) such that for any fixed q we get a regular function X_q (of two remaining arguments) and all regular functions can be obtained in this way.

Proof. Let us enumerate all programs for the functions of two arguments (whether these functions are regular or not); we get a computable sequence of programs, and qth term of this sequence is called "qth program" in the rest of the proof.

Then we define $X(q,\varepsilon,i)$ as the output of the qth program on input ε,i, assuming that some conditions are met; otherwise $X(q,\varepsilon,i)$ is undefined. The conditions guarantee that all X_q are regular, and that regular functions are untouched.

To compute $X(q,\varepsilon,i)$, we apply in parallel the program number q to all pairs

$$(\varepsilon,0),(\varepsilon,1),\ldots,$$

(starting with one step of the first computation, then making two steps of the first two computations etc.)

When some computation terminates with some output, we interrupt this process to verify that strings obtained so far do not violate the regularity condition. This means that we start to compute more and more precise approximations to $p(z)$ for all these strings until we could guarantee that the sum of all these $p(z)$ is less then ε (this happens if the sum of approximations is less than ε minus the sum of approximation errors). (Since μ is computable, we can compute approximations to $p(z)$ for any z with any precision.)

It is possible that we do not return from this interrupt; this happens if the sum of measures is not less than ε.

Now $X(q,\varepsilon,i)$ is defined as the output of qth program on (ε,i) if this output appears and passes the test during the process described.

If qth program computes a regular function, the verification will never fail and X_q coincides with this function. On the other hand, for every q the function X_q is regular: if for some ε the qth program (applied to ε and all $i=0,1,2,\ldots$) generates strings whose total measure is too large, only finitely many of the strings will be let through, and their total measure is still less than ε. Lemma is proved.

57 Explain why we need to change the definition of correctness. [Answer: if the sum consists of final number of nonzero terms and their sum is exactly ε, we will newer know this.]

Now we finish the proof of Martin-Löf theorem. Let X be the function provided by the Lemma. For all $q=0,1,2,\ldots$ consider the effectively null set Z_q that corresponds to the regular function X_q. Every effectively null set by definition is a subset of Z_q for some q. It remains to show that the union $Z_0\cup Z_1\cup\ldots$ is an effective null set.

We do the same trick that is used to prove that a countable union of null sets is a null set. To find a covering of total measure less that ε for $\cup_q Z_q$, we combine the $(\varepsilon/2)$-covering for Z_0 with $(\varepsilon/4)$-covering for Z_1, etc.

More formally, we consider a function $x(\varepsilon,i)$, that is defined as follows:

$$x(\varepsilon,[q,k])=X(q,\varepsilon/2^{q+1},k).$$

Here $[q,k]$ stands for the number of pair q,k under some computable bijection between $\mathbb{N}^2$ and $\mathbb{N}$. $\triangleright$

Now we are ready to give the definition of Martin-Löf random sequence. Assume that some computable measure μ on the set Ω is fixed.

Definition. A sequence ω is called *Martin-Löf random* (*ML-random*) with respect to μ if ω does not belong to the largest effectively null set (with respect to μ) provided by Theorem 28.

Reformulation: a sequence is Martin-Löf random if it does not belong to any effectively null set.

One more version: a sequence ω is Martin-Löf random if the singleton $\{\omega\}$ is not an effectively null set.

A digression: terminology. The notion of Martin-Löf randomness is a refinement of the intuitive idea of a "typical sequence". One could say that a sequence is "typical" if it does not have any regularities or special features which separates it from most sequences. (If somebody says that "Mr. X is a typical math professor" she means that Mr. X has no special characteristics that make him different from the most math professors.) A "special feature" is a feature that is posessed only by a negligible fraction of the objects considered (sequences). For example, if a sequence ω starts with 0, this is not a special feature, since half of the sequences have start with 0 On the other hand, if each second term of ω is zero, this is indeed a special feature.

This informal idea is implemented in the Martin-Löf definition: a special feature is a feature that corresponds to an effectively null set, and therefore typical sequences are sequences that do not belong to any effectively null set, i.e., Martin-Löf random sequences.

It would be more logical to use the word "typical" for Martin-Löf's definition and reserve the word "random" for more general intuitive notion that can be formalized in different ways (and the idea of a typical sequence is one of them). However, the attempts to introduce a new, more logical, terminology often make the situation worse (authors have to confess that this can be said about their own attempts!). And there is already a lot of misunderstanding: the words "random sequence" are already used in different ways.

So we keep the term "Martin-Löf random sequence" for the definition given above (and sometimes use the name "Martin-Löf typical sequence") keeping the name "random sequence" for a vague philosophical notion of randomness that needs additional clarification to become a mathematical notion. (End of digression.)

The following statement is a trivial corollary of Martin-Löf theorem; however, it deserves a careful thinking since it looks counter-intuitive.

{randomml-r

Theorem 29 *A set $A \subset \Omega$ is an effectively null set if and only if all its element are not Martin-Löf random* (*are non-typical*).

In particular, the set of all non-typical sequences is the largest effectively null set, and the set of all typical sequences has measure 1.

$\triangleleft$ Indeed, any element of any effectively null set is non-typical by definition; on the other hand, if all elements of some set A are non-typical, then A is a subset of the largest effectively null set and therefore A is an effectively null set. $\triangleright$

What is strange here? A set A is a null set if it has "few elements"; the nature of these elements does not matter much. Any singleton $\{\omega\} \subset \Omega$ is a null set and this does not depend on the properties of the sequence ω.

On the other hand, now we see that if we replace null sets by effectively null sets, the situation changes drastically: we may put as many non-typical sequences in a set as we wish, and it would remain an effectively null set, but any one typical (ML-random) sequence added destroys this property.

For example, recall that any computable sequence forms an effectively null singleton (with respect to uniform measure). We immediately get the following corollary:

{computable

Theorem 30 *The set of all computable sequences of zeros and ones is an effectively null subset of* Ω (*with respect to the uniform measure*).

It is interesting to note that this observation was made before Martin-Löf gave the definition of randomness, while developing the constructive version of calculus ("Zaslavsky construction" used for many counterexamples; it deals with real numbers instead of bit sequences).

In the next section we explore the properties of ML-random sequences (with respect to the uniform measure). We end this section with the following nice criterion for ML-randomness which is attributed to R. Solovay in [**?**].

{solovay-cr

Theorem 31 *A sequence* ω *is no ML-random with respect to a computable measure* μ *if and only if there exists s computable sequence of intervals with finite sum of measures that covers* ω *infinitely many times, i.e., a computable sequences of binary strings* $x_0, x_1, x_2, \ldots$ *such that*

$$\sum_i \mu(\Omega_{x_i}) < \infty$$

and $\omega \in \Omega_{x_i}$ *for infinitely many i.*

$\lhd$ Assume that ω is not ML-random. Then for each ε we can effectively find a computable sequence of intervals that covers $\{\omega\}$ and has the sum of measures less than ε. Then we combine these sequences for $\varepsilon = 1, 1/2, 1/4, 1/8, \ldots$ and get a computable sequence of intervals with sum of measures not exceeding 2 that covers ω infinitely many times (at least once for each ε).).

On the other hand, assume that there exists a computable sequence of strings $x_0, x_1, x_2, \ldots$ such that the sum of measures of corresponding intervals does not exceed some constant c and infinitely many of these intervals contain ω. We may assume without loss of generality that c is a rational number. To find a covering for ω that has sum of measures less than ε, we consider the set M_N of all sequences in Ω that are covered at least N times. Here N is a positive integer such that $c/N < \varepsilon$. It is easy to see that M_N can be represented as the union if a computable sequence of disjoint intervals (while reading $x_0, x_1, \ldots$, we see more and more elements of M_N and add respective intervals when necessary). Therefore the set $\{\omega\}$ is an effectively null set and the sequence ω is not ML-random. $\rhd$

Remark. This result is a constructive version of Borel–Cantelli Lemma (if the sum of measures of sets $A_0, A_1, \ldots$ is finite, then the set of all points that belong to infinitely many A_i is a null set),

and our argument is an effective version of a classical proof of Borel–Cantelli Lemma. However, we should be careful since not any classical proof can be effectivized. The standard proof (since the series is converging, its tails could be made as small is needed) does not work here, since there is no way to find an appropriate tail given ε.

3.4 Properties of Martin-Löf randomness

{randomun}

The Strong Law of Large Numbers also provides an example of an effective null sets (with respect to the uniform measure).

{randomml-l

Theorem 32 *A set of all bit sequences that do no have limit frequency $1/2$ is an effectively null set with respect to the uniform measure.*

$\lhd$ It is enough to prove that for every rational $\varepsilon > 0$ the set of all sequences such that frequency of ones is greater than $1/2+\varepsilon$ infinitely many times (or less than $1/2-\varepsilon$ infinitely many times) is an effective null set.

Indeed, the upper bound for the measure of this set achieved in the proof of the Strong Law of Large Numbers in the previous section (Theorem 27, p. 51) is effective: the set of intervals was the set of all sufficiently long strings with large frequency deviation, and its total measure was effectively bounded by a tail of the converging geometric series. $\rhd$

The statement of this theorem can be reformulated as the property of individual ML-random sequences:

{random-llr

Theorem 33 *Let $\omega = \omega_0\omega_1\ldots$ be a ML-random sequence with respect to the uniform measure. Then*

$$\lim_{n\to\infty}\frac{\omega_0+\omega_1+\ldots+\omega_{n-1}}{n}=\frac{1}{2}.$$

The similar statement is true for arbitrary Bernoulli measure. Let p and q be computable positive reals such that $p+q=1$. Consider the Bernoulli measure with parameters q and p (the sequence of independent coin tossing with success probability p). It is easy to check that this us a computable measure (since p and q are computable).

Theorem 34 *Any ML-random sequence with respect to Bernoulli measure with computable parameters q, p has limit frequency p.*

$\lhd$ Indeed, the upper bound for the probability of large deviations (obtained by comparing the given Bernoulli measure with the other one, with shifted p, see Problem 47, p. 53), gives an explicit bound and an explicit set of intervals, so we get an effectively null set. $\rhd$

There are several other properties of typicalness (ML-randomness) with respect to the uniform measure:

Theorem 35 *Let ω be a typical (=ML-random) sequnce with respect to the uniform measure. Then any other sequence which is obtained from ω by a finite number of insertions / deletions / changes, is also typical (ML-random).*

$\triangleleft$ It is enough to show that adding a zero/one in the beginning of a typical sequence or deleting the first term of a typical sequence gives a typical sequence.

Indeed, assume that sequence ω is *not* typical, i.e., forms an effectively null singletion: for each ε one can effective construct a covering by intervals with total measure less than ε. Let us add zero at the beginning of all these intervals (i.e., the corresponding strings). We get a covering for 0ω whose measure is twice smaller. This argument shows that if ω is not typical, then 0ω is not typical either. (Similar argument works for 1ω.)

On the other hand, if we delete the first bit of all strings that form a covering for ω, we get a family of intervals of twice larger measure that covers ω' (obtained from ω by first bit deletion). Therefore, ω' is not typical. $\triangleright$

58 Prove that replacing all zeros by ones and vice versa in a typical sequence (with respect to the uniform measure) we get a typical sequence.

The folllowing problem shows that a computable subsequence of a typical sequence is typical.

59 Let $n_0, n_1, n_2, \ldots$ be a computable sequence of different integers ($n_i \neq n_j$ if $i \neq j$). Let $\omega = \omega_0\omega_1\omega_2\ldots$ be a typical (=ML-random) sequence. Then its subsequence

$$\omega|n = \omega_{n_0}\omega_{n_1}\omega_{n_2}\ldots$$

is typical (ML-random). [Hint: any interval Ω_x in a cover for $\omega|n$ produces a finite family of intervals whose union is the set of sequences whose $(n_0, n_1, \ldots, n_{i-1})$-subsequence coincides with x (here i is the length of the string x). The total measure of these intervals equals 2^{-i}, the measure of Ω_x.]

More general selection rules are consider in Chapter **??** (p. **??**) where a frequency approach to the notion of randomness (von Mises' approach) is considered.

60 Let ω be a typical (=ML-random) sequnce with respect to the uniform measure. Let us split ω into two-bit blocks and then replace blocks 00 by zeros and blocks 01, 10 and 11 by ones. Prove that the resulting sequence is typical with respect to Bernoulli measure with parameters $1/4, 3/4$. [Hint. We described a transformation of Ω into itsef. The preimage of any open set U is open, and the uniform measure of that preimage equals the $(1/4, 3/4)$-measure of the set U.]

61 (Continued.) Prove that any typical (=ML-random) sequence with respect to the $(1/4, 3/4)$-measure can be obtained in this way from a sequence that is typical (=ML-random) with respect to the uniform measure. [Hint: For any open set $B \subset \Omega$ consider the set B' of all sequences ω such that $F^{-1}(\{\omega\}) \subset B$ (the set of sequences that do not have a preimage outside B, i.e., the complement to the image of the complement of B). The image of a compact set is a compact set, therefore B' is open. Show that if B is a union of an enumerable family of intervals, then B' is also a union of enumerable family of intervals, and Bernoulli measure of B' does not exceed the uniform measure of B. See also the proof of a more general statement (Theorem 99, p. 142).]

What can be said about the "complexity" of a ML-random sequence (with respect to the uniform measure) from the viewpoint of the recursion theory? We know already that ML-random sequence is not computable. It also cannot be a characteristic function of an enumerable (recursively enumerable, computably enumerable) set.

Theorem 36 *Let A be an enumerable set of natural number. Consider its characteristic sequences $a_0a_1a_2\ldots$ ($a_i = 0$ for $i \notin A$ and $a_i = 1$ for $i \in A$). This sequence is not ML-random.*

$\lhd$ Let k be an arbitrary natural number. Let us enumerate the set A and see what happens with k first bits of its characteristic sequences. As (current version of) A increases, we get more and more ones in this k-bit prefix. In this way we get at most $k+1$ candidates; at some point we come to a final (true) one, but we never know that this happened already. Anyway, the set of candidates is enumrable and the number of candidates does not exceed $k+1$ (since k-bit prefix can have $0\ldots k$ ones). The total measure of these intervals is $(k+1)/2^k$ and therefore can be made arbitrarily small. (Note that the definition of the effectively null set allows us to enumerate the intervals that form a covering, and this is exactly what we can do in our case.) $\rhd$

A natural question arises: in what sense one can provide explicitly a ML-random sequence? As we have seen, neither computable nor characteristic sequences of enumerable sets are random. If you are familiar with the basics of the recursion theory (see, e.g., [**?**]), you may appreciate the following result: there exist a ML-random sequence that belongs to the class $\Sigma_2 \cap \Pi_2$ of the arithmetic hierarchy (this class can be also described as the class of all $\mathbf{0}'$-computable sequences).

Theorem 37 *There exists a $\mathbf{0}'$-computable sequence that is ML-random with respect to the uniform measure.*

$\lhd$ It is enough to show that for any enumerable set of strings $\{x_0, x_1, \ldots\}$ such that $\sum 2^{-l(x_i)} < 1/2$ there exists a $\mathbf{0}'$-computable sequence that does not have any of x_i as a prefix. (Indeed, the largest effective null set has such a covering with total measure less than $1/2$, and any sequence that is not covered is ML-random.)

The intervals Ω_{x_i} are divided into two groups: some of them belong to the left half of Ω (i.e., x_i starts with 0) and some belong to the right half. Total measure of both groups at most $1/2$. Therefore, at least one of the group has total measure at most $1/4$. However, looking at the sequence x_i, we cannot find out which half has this property (since at any moment new large interval can arrive).

However, $\mathbf{0}'$-oracle allows us to make this choice, since the event "measure exceeds 1/4" is enumerable. Then we divide this half into two parts of size $1/4$ each and choose one of them where the total measure of corresponding intervals does not exceed $1/8$, and so on.

In this way we get a $\mathbf{0}'$-computable sequence with the following property: each its prefix is at most half-covered by our intervals. In particular, no prefix of this sequence can appear in the sequence x_i, and this is what we need. $\rhd$

(See Section 5.7 (p. 135) for an alternative proof.)

In fact our argument uses a relativized version of the following result:

62 Assume that $x_0, x_1, x_2, \ldots$ is a computable sequence of binary strings and the sum {schnorr-nc

$$\sum_i 2^{-l(x_i)}$$

is less than 1 and is a computable real number. Then there exists a computable sequence of zeros and ones that has neither of x_i as its prefix.

[Hint: Let this sum be less than some rational $S < 1$. By induction construct a computable sequence $\omega_0\omega_1\omega_2\ldots$ with the following property: the fraction of the set $U = \cup\Omega_{x_i}$ among the sequences that have prefix $\omega_0\ldots\omega_k$ is less than S.]

This problem is related to the definition of randomness suggested by C. Schnorr [**?**]. He gave a more restrictive definition of an effectively null set. The additional requirement: for every (rational) $\varepsilon > 0$ the total measure of corresponding intervals is not only less than ε but also is a computable real (and the approximation algorithm computably depends on ε). This requirement is equivalent to the following one: for every $\varepsilon > 0$ and $\delta > 0$ one can effectively find out how many terms in the series $\sum_i p(x(\varepsilon,i))$ are needed to make the tail less than δ. (For a series with non-negative terms the computability of sum is equivalent to computable convergence.)

By *Schnorr effectively null sets* we mean the effectively null sets according to this modified definition. (Schnorr calls them *total rekursive Nullmenge*, see Definition 8.1 in [**?**]; effectively null sets are called *rekursive Nullmenge*, see Definition 4.1.)

63 Let us change the definition of an effectively null set in another way: now we require that the total measure of all intervals in the covering is *exactly* ε. Show that this definition is equivalent to the definition of Schnorr effectively null set. (One can also consider the measure of the union of all intervals instead of the sum of measures.)

Problem 62 shows that for every Schorr effectively null set there exists a computable sequence outside this set. (For simplicity let us consider the case of uniform measure.) On the other hand, every computable sequence (i.e., the singleton made of it) is a Schnorr effectively null set. Therefore, none of the Schnorr effectively null sets is the largest one in the class (in other words, the union of all Schnorr effectively null sets is not a Schnorr effectively null set). Nevertheless we can call a sequence which does not belong to any Schnorr null set a *Schnorr random sequence* or *Schnorr typical sequence*.

Since now we have less effectively null set, we may get the broader class of random sequences, and it is indeed the case. The following problem (together with the results of Chapter 5) guarantees that there exist Schnorr random sequences that are not Martin-Löf random.

64 Prove that there exists a Schnorr random sequences $\omega = \omega_0\omega_1\omega_2\ldots$ whose prefixes have logarithmic complexity, i.e., $KS(\omega_0\ldots\omega_{n-1}) = O(\log n)$.

[Hint: The previous problem shows how one can construct a computable sequence that does not belong to a given Schnorr effectively null set. At some point of this construction we can take into account another Schnorr effectively null set and get a computable sequence that does not belong to both. (Indeed, we need to take a sufficiently small covering for the second set that does not go out of the safety margin in the construction form the first set.) Moreover, we can consider infinitely many Schnorr effectively null sets in this way (adding them one after another). This will not give us a computable Schnorr random sequnce (it does not exist at all), because we need additional information that says us which algorithms correspond to Schnorr effectively null set and which do not. But if we postpone the introduction of a new algorithm to the moment when the constructed prefix of our sequence is rather long, this additional information is logarithmic compared to the prefix length.]

We return to Schnorr definition of randomness in Section **??** where it is reformulated in terms of computable martingales.

65 Prove that a sequence ω is not Schnorr random if ans only if there exists a computable sequence of strings $x_0, x_1, \ldots$ such that the series $\sum_i p(x_i)$ computably converges (has a computable sum) and infinitely many of x_i are prefixes of ω (this is a version of Theorem 31 statement for Schnorr randomness). [Hint: In this case even the standard proof of Borel–Cantelli lemma works.] {schnorr-sc

4 A priori probability and prefix complexity

{prefix}

4.1 Randomized algorithms and semi-measures on $\mathbb{N}$

{prefix-pp}

In this section we consider algorithms (=programs, machines) equipped with a random number generator. That is, algorithms may perform instructions of the following form:

$$b := \mathit{random}.$$

This instruction assigns to the variable (memory cell) b a random bit (0 or 1), both values are assigned with equal probabilities. To perform this instruction we toss a fair coin and write its outcome in the memory cell b. Algorithms including such instructions are called *randomized* or *probabilistic*.

The result (output) produced by a randomized algorithm depends not only on its input but also on the result of the coin tossing. That is, for every fixed input, the output of a randomized algorithm is a random variable.

Speaking formally, the probability that a randomized algorithm A prints a result x is defined as follows. Consider the uniform Bernoulli distribution on the space Ω of all infinite 0-1-sequences. The measure of the set Ω_u of all infinite continuations of a finite string u is equal to $2^{-l(u)}$.

Let x be an input for a randomized algorithm A and let $\omega \in \Omega$ be an infinite sequence of zeros and ones. We denote by $A(x,\omega)$ the output of A on input x, if random bits used by the algorithm are taken from the sequence ω. More specifically, each call of a random generator returns the next bit of ω. If the algorithm A does not halt (for given x and ω), then the value $A(x,\omega)$ is undefined.

Let y be a possible output of A. Consider the set $\{\omega \mid A(x,\omega) = y\}$. This set is the union of intervals Ω_z over all outcomes z of coin tossing that guarantee that A prints y having x as input. The probability that A on input x outputs y is equal to the measure of this set.

In this section, we consider machines without input whose outputs are natural numbers. Here is an example of such machine. It tosses a coin until a 1 appears and outputs the number of 0s preceding the first 1. The probability p_i of the event "the output is i" is equal to $2^{-(i+1)}$. Indeed, the algorithm outputs i if and only if the first i random bits are zeros and the $(i+1)$st bit is 1. This happens with probability $2^{-(i+1)}$.

The sum of the series $\sum p_i$ is equal to 1 in this example. Indeed, the algorithm does not halt if and only if all random bits are zeros and this happens with zero probability.

We assign to every probabilistic machine (having no input and producing natural numbers) a sequence $p_0, p_1, \ldots$ of real numbers: p_i is the probability that the machine prints the number i. We say that the probabilistic machine *generates* the sequence $p_0, p_1, \ldots$. Which sequences $p_0, p_1, \ldots$ can be obtained in this way? There is an obvious necessary condition: $\sum p_i \leqslant 1$ (since the machine cannot produce two different outputs). However, this inequality is not sufficient, as there are countably many randomized algorithms and uncountably many sequences satisfying this condition.

Let us answer first a simplified question. Consider the halting probability of a randomized machine without input, i.e., the probability that the machine halts. Which real numbers can be halting probabilities of probabilistic machines without input? To answer this question we need to introduce the notion of a lower semicomputable real number.

A real number α is *lower semicomputable*, if it is equal to the limit of a computable non-decreasing sequence of rational numbers.

66 Prove that if α is a computable real number (i.e., there is an algorithm that for any given rational $\varepsilon > 0$ computes a rational approximation to α with precision ε) is lower semicomputable. [Hint: We can construct an increasing sequence using approximations from below.]

67 Show that a real number α is computable if and only if both numbers α and $-\alpha$ are lower semicomputable.

A real number α is lower-semicomputable if and only if the set of rational numbers that are less than α is enumerable. (It explains why lower semicomputable reals are also called *enumerable from below*.)

Indeed, let α be the limit of a non-decreasing computable sequence $a_0 \leqslant a_1 \leqslant a_2 \leqslant \ldots$ of rationals. For each i enumerate all rational numbers that are less than a_i. All rational number less than α (and no other) will appear in the enumeration, and only such numbers.

Conversely, assume that we can enumerate all rational numbers that are less than α. Omitting in this enumeration all numbers that are less than previously met ones, we obtain a non-decreasing sequence whose limit is α.

Using the notion of a lower semicomputable real, we obtain the following answer to the above question:

{enumerable

Theorem 38 (**a**) *Let M be a probabilistic machine without input. The halting probability of M is a lower semicomputable real number.*

(**b**) *Every lower semicomputable real is the halting probability of some probabilistic machine.*

$\lhd$ (a) Let p_n stand for the probability that M halts within n steps. The number p_n is rational: the algorithm can toss a coin at most n times within n steps, thus the halting probability is a multiple of $1/2^n$.

We can find p_n by simulating the run of the machine and probing all possible outcomes of the coin tossing. The sequence $p_0, p_1, \ldots$ is non-decreasing and its limit is equal to the halting probability of M.

(b) Assume that a real q is lower semicomputable. That is, there is a computable sequence $q_0, q_1, \ldots$ of rational numbers such that $q = \lim q_n$ and

$$q_0 \leqslant q_1 \leqslant q_2 \leqslant \ldots.$$

We have to construct a probabilistic machine whose halting probability is equal to q. Let the machine toss a coin and let $b_0, b_1, b_2, \ldots$ be the obtained random bits. Consider the real number $\beta = 0.b_0b_1b_2\ldots$; it is uniformly distributed in $[0,1]$. Let the machine (in parallel to coin tossing) compute the rational numbers $q_0, q_1, q_2, \ldots$. The machine halts when it finds out that $\beta < q$. That is, the machine halts if for some i the rational number $\beta_i = 0.b_0b_1\ldots b_i111\ldots$ (the currently known upper bound of β) is less than q_i (the currently known lower bound of q). See Fig. 9 for a symbolic representation of this argument.

The constructed machine halts if and only if $\beta < q$. Indeed, assume that β is less than q. The numbers q_i tend to q and the lower bounds β_i of β tend to β, as $i \to \infty$. Therefore for some i the number q_i is greater than β_i. On the other hand, if the machine halts then $\beta < q$ by construction.

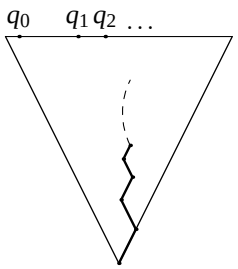

Figure 9: Comparing $\beta = 0.b_0b_1b_2\ldots$ and $q = \lim q_i$. {prefixpp.1

Thus the halting probability of the machine is equal to the probability of the event $\beta < q$. The latter probability equals the length of the segment $[0,q)$, that is, to q. (Recall that β is uniformly distributed in the segment $[0,1]$.) $\triangleright$

Let us return to probability distributions that can by generated by probabilistic machines. We need a new notion. A sequence $p_0, p_1, p_2, \ldots$ is *lower semicomputable* if there is a function $p(i,n)$, where i,n are integers and $p(i,n)$ is either a rational number or $-\infty$, with the following properties: the function $p(i,n)$ is non-decreasing in the second argument:

$$p(i,0) \leqslant p(i,1) \leqslant p(i,2) \leqslant \ldots,$$

and

$$p_i = \lim_{n\to\infty} p(i,n)$$

for all i.

One could say that the sequence p_i is lower semicomputable if the numbers $p_0, p_1, p_2\ldots$ are "uniformly lower semicomputable". The next theorem provides an alternative way to define lower semicomputable sequences.

Theorem 39 *A sequence $p_0, p_1, p_2\ldots$ is lower semicomputable if and only if the set of pairs $\langle r,i\rangle$, where i is a natural number and r is a rational number less than p_i, is enumerable.*

$\triangleleft$ Recall that a set is enumerable if there is an algorithm that prints all its elements in some order and with arbitrary delays between consecutive elements (the algorithm may not halt even if the set is finite).

Assume that a sequence $p_0, p_1, p_2, \ldots$ is lower semicomputable. Let $p(i,n)$ be the function from the definition of the lower semicomputability of $p_0, p_1, p_2, \ldots$. Arrange all the pairs $\langle r,i\rangle$ in a sequence so that every pair appears in the sequence infinitely many times. The algorithm enumerating all the pairs $\langle r,i\rangle$ with $r < p_i$ works in steps. On step n compare r and $p(i,n)$ where $\langle r,i\rangle$ is the nth pair in the chosen sequence. If $r < p(i,n)$ then print the pair $\langle r,i\rangle$, otherwise proceed to the next step. By definition, $r < \lim_n p(i,n)$ iff there exists n such that $r < p(i,n)$. Thus we will print all the pairs we have to print, and no other pairs.

Conversely, assume that the property $r < p_i$ is enumerable and let A be an algorithm enumerating all such pairs $\langle r,i\rangle$. To compute $p(i,n)$ we make n steps of the run of A. Consider all the

pairs that appeared within n steps and have i as the second component. Let $p(i,n)$ be equal to the largest first component of such pairs. If there are no such pairs, let $p(i,n) = -\infty$. As n increases, new pairs may appear and $p(i,n)$ may increase. The limit $\lim_n p(i,n)$ is equal to p_i, since all the rational numbers less than p_i will appear in the enumeration. $\triangleright$

This theorem explains why lower semicomputable sequences are also called *enumerable from below*.

We are now able to characterize probability distributions generated by probabilistic machines.

{semimeasur

Theorem 40 **(a)** *Let M be a probabilistic machine without input that outputs natural numbers. Let p_i denote the probability that the machine outputs i. The sequence of p_i is lower semicomputable and $\sum_i p_i \leqslant 1$.*

(b) *Let $p_0, p_1, \ldots$ be a lower semicomputable sequence of non-negative real numbers such that $\sum_i p_i \leqslant 1$. There is a probabilistic machine M that prints every i with probability exactly p_i.*

$\triangleleft$ The proof of item (a) is similar to the proof of corresponding statement in the previous theorem. We let $p(i,n)$ be the probability that M outputs i within n steps.

The proof of item (b) is also similar to the proof of corresponding assertion in the previous theorem. This time we assign to each natural i a subset of $[0,1]$ and the machine prints i if the real number $\beta = 0.b_0b_1b_2\ldots$ belongs to the set assigned to i. The sets assigned to different i's do not overlap. They may not cover the entire segment $[0,1]$. The set assigned to every i is a finite or countable union of half-open intervals $[a,b)$ of total length p_i.

In parallel, we toss a coin and obtain digits of the random number β. When we are sure that β gets into the set assigned to some natural number we print that number.

Here is a formal argument. Let $p(i,n)$ be the function of two variables from the definition of lower semicomputability of $p_0, p_1, \ldots$. Without loss of generality we may assume that $p(i,n) \geqslant 0$ for all i,n. Indeed, we can replace all negative values by zeros. We may assume also that for all n only finitely many values $p(i,n)$ are positive (let $p(i,n) = 0$ for all $i \geqslant n$). The probabilistic algorithm we construct runs in steps. On each step we allocate some space inside $[0,1]$. Our goal is that after the nth step the total length of intervals allocated to i is equal to $p(i,n)$ (for all i). This requirement is easy to keep: going from left to right, on step n we allocate for each i (such that $p(i,n) > p(i,n-1)$) a new interval of length $p(i,n) - p(i,n-1)$. We need to do this only for finitely many i, as for $i \geqslant n$ we have $p(i,n) = p(i,n-1) = 0$.

The total length of intervals used does not exceed 1, as $p(i,n) \leqslant p_i$ and $\sum p_i \leqslant 1$. Thus we will always be able to allocate the space we needed (at the left of the free space).

In parallel, the probabilistic machine tosses a coin, obtaining a random bit b_n on step n. It halts on step n and outputs i if it is known for sure that $\beta = 0.b_0b_1b_2\ldots$ belongs to the (interior) of the space allocated to i, i.e., if the closed interval consisting of all real numbers whose binary expansion starts with $b_0b_1\ldots b_n$ is included in the interior of the space allocated to i. (The interior of the segment $[u,v]$ is the interval (u,v).) By construction, for all i the measure of this set (interior of the space allocated to i) equals p_i. $\triangleright$

Any sequence p_i satisfying the conditions of the previous theorem is called a *lower semicomputable semimeasure* (or *enumerable from below semimeasure*) on $\mathbb{N}$. Sometimes we will use

also the notation $p(i)$ for p_i. We thus have two alternative definitions of a lower semicomputable semimeasure: (1) a probability distribution generated by a randomized algorithm; (2) a lower semicomputable sequence of non-negative reals whose sum does not exceed 1. The above theorem states that these definitions are equivalent.

The word "semimeasure" may look strange, but unfortunately there is no other appropriate term in the literature. Dropping semicomputability requirement, one can call any function $i \mapsto p_i$ with $\sum_i p_i \leqslant 1$ a *semimeasure* on $\mathbb{N}$. Every semimeasure on $\mathbb{N}$ defines a probability distribution on the set $\mathbb{N} \cup \{\bot\}$ where $\bot$ is a special symbol meaning "undefined". The probability of the number i is p_i and the probability of $\bot$ is $1 - \sum_i p_i$. In the sequel we consider lower semicomputable semimeasures only (unless stated otherwise explicitly).

We have considered so far (lower semicomputable) semimeasures on the natural numbers. The definition of a lower semicomputable semimeasure can be naturally generalized to the case of binary strings or any other constructive objects in place of natural numbers. For example, to define a notion of a lower semicomputable semimeasure on the set of binary strings we have to consider probabilistic machines whose output is a binary string.

Important remark: we will consider in Section 5 a notion of a semimeasure on the space consisting of all finite and infinite 0-1-sequences. Such a semimeasure is generated by a probabilistic machine that prints its output bit by bit and never indicates that the output string is finished. In particular the machine never halts. It leads to a different notion: all the machines considered in this section are required to halt after printing the output; for such machines, there is no essential difference between printing a binary string and a natural number.

4.2 Maximal semimeasures

{prefix-m}

Comparing two semimeasures on $\mathbb{N}$ we will ignore multiplicative constants. A lower semicomputable semimeasure m is called *maximal* if for any other lower semicomputable semimeasure m' the inequality $m'(i) \leqslant cm(i)$ holds for some c and for all i. (The name *greatest* (instead of "maximal") would be more accurate since we look for the greatest element of some partially ordered set, not the maximal one.)

{max-semi-M

Theorem 41 *There exists a maximal lower semicomputable semimeasure on* $\mathbb{N}$.

$\lhd$ We have to construct a probabilistic machine M with the following property. The machine M should print every number i with a probability that is at most constant times less than the probability that any other machine M' prints i (the constant may depend on M' but not on i).

Let the machine M pick at random a probabilistic machine M' and then simulates M'. The probability to pick each machine M' should be positive. If a machine M' is chosen with probability p then M will print a number i with probability at least $p \cdot$ (the probability that M' prints i). Thus one can let $c = 1/p$.

It remains to explain how to implement the random choice of a probabilistic machine. Enumerate all the probabilistic machines in a natural way; let $M_0, M_1, M_2, \ldots$ be the resulting sequence. We toss a coin until the first 1 appears. Then we simulate the machine M_i where i is the number of zeros preceding the first 1. $\rhd$

It is instructive to prove this theorem once more using the language of lower semicomputable sequences instead of probabilistic algorithms. Basically, we need to show that there exists a convergent lower semicomputable series having the lowest rate of convergence. That series should be greater than any other lower semicomputable convergent series (up to a multiplicative constant). More formally, we should consider only series with the sum at most 1, but this is not essential as anyway we allow to multiply the terms of a series by a constant.

To find such a series, we sum up with certain weights all the lower semicomputable series with sum at most 1. The weights should form a converging series too. This will imply that the resulting series converge. By construction it will be maximal (up to a multiplicative constant). There is only one problem left: how to guarantee that the resulting series is lower semicomputable.

The lower semicomputable of a semimeasure is witnessed by a computable function $p : \langle i,n\rangle \mapsto p(i,n)$. There are only countably many such functions, since there are only countably many algorithms. Enumerate all such functions, $p^{(0)}, p^{(1)}, p^{(2)}, \ldots$, and consider the function

$$p(i,n) = \sum_{k=0}^{n} \lambda_k p^{(k)}(i,n)$$

where λ_k is a computable sequence of rational numbers with $\sum_k \lambda_k \leqslant 1$, say, $\lambda_k = 2^{-k-1}$. The resulting function p is non-decreasing in n for every i. Indeed, as n increases, the number of terms in the sum defining p increases and the value of every term increases, too. And for all i we have

$$\lim_{n\to\infty} p(i,n) = \sum_k \lambda_k \lim_{n\to\infty} p^{(k)}(i,n).$$

That is, the constructed semimeasure is indeed equal to the sum of all lower semicomputable semimeasures with weights λ_k.

There is a bug in this argument. The function $p(i,n)$ should be computable, and thus we cannot use arbitrary enumeration of lower semicomputable functions in our construction. We need to arrange them so that the function $p : \langle k,i,n\rangle \mapsto p^{(k)}(i,n)$ is computable as a function of all its three arguments. Note that we cannot just let $p^{(k)}$ be the function computed by kth program: it may happen that the kth program does not define any lower semicomputable semimeasure. (It may compute a function which is not total, or a function that sometimes decreases in the second argument or a function whose sum is greater than 1.)

The bug can be fixed using the following

Lemma. Every program P computing a function of two natural arguments and taking rational values (and possibly the value $-\infty$) can be algorithmically transformed into a program P' having the following properties. The program P' defines a lower semicomputable semimeasure. If the program P itself defines a lower semicomputable semimeasure, then P' defines the same semimeasure.

Proof of the Lemma. Let P any program satisfying the condition of the Lemma. (We do not assume that P is total.) First we let $P'(i,n)$ be equal to the maximal number output within the first n steps in the computation of $P(i,0),\ldots,P(i,n)$. If none of this computations terminates within n steps or all the results are negative, we let $P'(i,n) = 0$. This definition guarantees that

$P'(i,n)$ is non-negative and is non-decreasing in n. For every i, if $P(i,n)$ is defined for all n and is non-negative and non-decreasing in n, then $\lim_n P'(i,n) = \lim_n P(i,n)$.

It remains to ensure that $\sum p'_i \leqslant 1$ where $p'_i = \lim_n P'(i,n)$. To this end first let $P'(i,n) = 0$ for all $n < i$. This transformation does not change the limit and preserves monotonicity in n. The advantage is that now the sum of $P'(i,n)$ over all i is finite and can be computed for every n. We need that this sum does not exceed 1. To enforce this we do not increase P' if we see that this would violate our restriction. We trim first the value $P'(i,n)$ for $n = 0$, then for $n = 1$ etc. The Lemma is proved.

Using the transformation described in the Lemma, we arrange all the lower semicomputable semimeasures into a computable sequence. The weighted sum of all its terms is a maximal lower semicomputable semimeasure. Thus we obtain another proof of Theorem 41.

Fix any maximal lower semicomputable semimeasure $p_0, p_1, p_2, \ldots$ on the natural numbers. We will use the notation $m(i)$ for p_i and the notation m for the semimeasure itself. The value $m(i)$ is called the *a priori probability* of i. (Another name for m is the *universal semimeasure* on $\mathbb{N}$.) Here is an explanation of this term. Assume that we are given a device (a black box) that after being turned on produces a natural number. For each i we want to get an upper bound for the probability that the black box outputs i. If the device is a probabilistic machine then *a priori* (without any other knowledge about the box) we can estimate the probability of i as $m(i)$. This estimate can be much more than the unknown true probability, but only $O(1)$ times less than it.

The a priori probability of a number i is closely related to its complexity. Roughly speaking, the less the complexity is, the larger the a priori probability is. More specifically, we will show that a slightly modified version of complexity (the so-called prefix complexity) of i is equal to the minus logarithm of $m(i)$.

4.3 Prefix machines

{prefix-ma}

The difference between prefix complexity and plain complexity can be explained as follows. Defining prefix complexity, we consider only "self-delimiting descriptions". This means that the decoding machine does not know where the description ends and has to find this information itself. One can clarify this idea in several non-equivalent ways. We will discuss all them further in detail.

Let us start with a following definition. Let f be a function whose arguments and values are binary strings. We say that f is *prefix-stable*, if the following holds for all strings x, y:

$$f(x) \text{ is defined and } x \text{ is a prefix of } y \quad \Rightarrow \quad f(y) \text{ is defined and is equal to } f(x).$$

{prefix-co

Theorem 42 *There exists an optimal prefix-stable decompressor (for the family of all prefix-stable decompressors).*

$\lhd$ Recall that a decompressor (description mode) is a computable function mapping strings to strings. (All strings are binary.) The plain complexity is defined using an optimal function in the class of all such functions. Now we restrict the class of decompressors to computable *prefix-stable* functions. We assign to each prefix-stable function D the complexity function KP_D, which is defined just as earlier: $KP_D(x)$ is the length of a shortest description of x with respect to D

(i.e., minimal $l(y)$ among all y such that $D(y) = x$). So the definition of $KP_D(x)$ coincides with that of $KS_D(x)$; we write KP instead of KS just to stress that we consider now only prefix-stable decompressors.

We have to show that there exists an optimal prefix-stable decompressor D (for the class of all prefix-stable decompressors). The latter means that for any other *prefix-stable* decompressor the inequality $KP_D(x) \leqslant KP_{D'}(x) + c$ holds for some c and all x.

Recall that for the plain complexity we have constructed an optimal decompressor D by letting

$$D(\hat{p}y) = p(y).$$

Here $\hat{p}$ is a self-delimiting description of p, say, $\hat{p} = \overline{p}01$ where $\overline{p}$ stands for the string p with all bits doubled. The notation $p(y)$ refers to the result printed by the program p given input y (more precisely, the string p is interpreted as a program in a universal programming language).

Is this decompressor a prefix-stable one? Certainly not. Indeed, there is a program p computing a function that is not prefix-stable, say, $p(0) = a$ and $p(00) = b$ where $a \neq b$. Then $D(\hat{p}0) = a$ and $D(\hat{p}00) = b$.

To construct an optimal prefix-stable decompressor, we modify the definition of D as follows. We enforce prefix-stability of programs by converting every program p to the program $[p]$ working as follows:

(1) Apply p to all inputs in parallel. If the computation of p on an input y halts with a result z we write down the pair $\langle y, z\rangle$. Let $\langle y_i, z_i\rangle$ denote the resulting sequence of pairs (enumerating the graph of p: $z_i = p(y_i)$).

(2) We delete some terms of the sequence $\langle y_i, z_i\rangle$ so that the resulting sequence is a graph of a prefix-stable function. More specifically, let us call strings y and y' *compatible* if one of them is a prefix of the other one (an equivalent definition: both strings are prefixes of a third string). We say that a pair $\langle y_i, z_i\rangle$ contradicts to a pair $\langle y_j, z_j\rangle$ if y_i is compatible with y_j, but $z_i \neq z_j$. We delete a pair $\langle y_i, z_i\rangle$ if it contradicts to a pair $\langle y_j, z_j\rangle$ with $j < i$. (The argument would work as well if we deleted a pair only when it contradicts to a *non-deleted* previous pair.)

(3) Computing the sequence $\langle y_i, z_i\rangle$ and filtering out some its terms is a process that does not depend on the input for the program $[p]$. The input string y is taken into account as follows. We wait until a (non-deleted) pair $\langle y_i, z_i\rangle$ appears such that y_i is a prefix of y. Once we encounter such a pair, we print the result z_i and halt.

For every program p the function $y \mapsto [p](y)$ is prefix-stable. Indeed, assume that $[p](y) = z$. By construction there is a non-deleted pair $\langle y_i, z\rangle$ such that y_i is a prefix of y. Assume furthermore that y is a prefix of y'. We need to show that $[p](y') = z$. The string y_i is a prefix of y' as well, therefore $[p](y') = z$ or $[p](y') = z_j$ where $\langle y_j, z_j\rangle$ is a non-deleted pair such that $j < i$ and y_j is a prefix of y'. In the latter case y_j is compatible with y_i and, since the pair $\langle y_i, z\rangle$ does not contradict to the pair $\langle y_j, z_j\rangle$, we have $z_j = z$.

If p is prefix-stable then no pair is deleted in the run of its transformed version $[p]$. Therefore $[p](y)$ is defined as p's output on y or a prefix of y. As we assume that p is prefix-stable, this is the same.

Now we are able to finish the proof. Let

$$D(\hat{p}y) = [p](y).$$

We have to verify that D is prefix-stable and optimal (in the class of all prefix-stable decompressors).

To prove the first statement, assume that $\widehat{p_1}y_1$ is a prefix of $\widehat{p_2}y_2$. We need to show that $D(\widehat{p_1}y_1)$ and $D(\widehat{p_2}y_2)$ coincide. As both the strings $\widehat{p_1}$, $\widehat{p_2}$ are prefixes of the string $\widehat{p_2}y_2$, they are compatible. Thus $p_1 = p_2$ (as the encoding $p \mapsto \widehat{p}$ is self-delimiting) and y_1 is a prefix of y_2. Since the program $[p_1]$ (=$[p_2]$) is prefix-stable, we conclude that $D(\widehat{p_1}y_1) = [p_1](y) = [p_2](y) = D(\widehat{p_2}y_2)$.

So we have shown that D is prefix-stable. To prove the optimality of D assume that a prefix-stable decompressor D' is given. Let p be its program. Then $D(\hat{p}y) = [p](y) = p(y)$. Therefore the complexity of all strings with respect to D' is at most $l(\hat{p})$ greater than the complexity with respect to D. $\triangleright$

Let us fix some optimal prefix-stable decompressor and omit the subscript D in $KP_D(x)$, speaking about the *prefix complexity* $KP(x)$ of x. As well as the plain complexity, the prefix complexity is defined up to an $O(1)$ additive term.

There is another way to define prefix complexity. Instead of prefix-stable functions we consider prefix-free functions. A function is called *prefix-free* if every two different strings in its domain are incompatible. If a prefix-free function is defined on a string, it is undefined on all its proper prefixes and continuations.

This time we restrict the class of decompressors to prefix-free ones, that is, computable prefix-free functions. We have the following theorem that is similar to Theorem 42:

{prefix-opt

Theorem 43 *The class of all prefix-free decompressors contains an optimal element.*

$\triangleleft$ The proof is very similar to the proof of Theorem 42. This time we construct, for every program p, a prefix-free program $\{p\}$ that works as follows:

(1) Just as before, run the program p on all inputs to obtain a sequence $\langle y_i, z_i\rangle$ of all pairs such that $z = p(y)$.

(2) Delete all pairs $\langle y_i, z_i\rangle$ such that y_i coincides with y_j for some $j < i$.

(3) Let y denote the input to the program $\{p\}$. We find the first non-deleted pair $\langle y_i, z_i\rangle$ with $y_i = y$ and output $z_i = \{p\}(y)$.

It is easy to verify that the mapping $y \mapsto \{p\}(y)$ is prefix-free for any p and coincides with the mapping $y \mapsto p(y)$ if the latter one is prefix-free. The rest of the proof repeats the corresponding part from the proof of Theorem 42. $\triangleright$

Let us fix some optimal prefix-free decompressor and let $KP'(x)$ denote the corresponding complexity.

Which of the complexity measures KP and KP' is "the right one"? This is a matter of taste. We will prove in Section 4.5 that these measures differ by an additive constant (and that both complexities coincide with the negative logarithm of the a priory probability). Thus the question is which of the two definitions is more natural. Again this is a matter of taste. Authors believe that the definition based on prefix-stable functions is more natural than the other one (which explains why we started with it). However, sometimes the second definition is more convenient. For instance, its use makes easier the proof of the theorem on the complexity of a pair (Section 4.6).

The properties of KP and KP' are similar to those of the plain complexity but differ in some important aspects:

- We start with a comparison of KS and KP:

$$KS(x) \leqslant KP(x) + O(1) \quad \text{and} \quad KS(x) \leqslant KP'(x) + O(1).$$

  These properties are straightforward, as both prefix-stable and prefix-free decompressors form a subclass in the class of all decompressors.

- Recall that $KS(x) \leqslant l(x) + O(1)$, as the optimal decompressor is better than the identity function. This argument is not valid for prefix complexity, as the identity function is neither prefix-stable nor prefix-free. We will show in Section 4.5 that this inequality is false for the prefix complexity. {prefix-com

- Nevertheless there is an upper bound for the prefix complexity in terms of the length. We will provide such bounds for KP', the same bounds hold for KP, the proofs being entirely similar. Let us show that $KP'(x) \leqslant 2l(x) + O(1)$. Indeed, consider a decompressor

$$D(\overline{x}01) = x$$

  where $\overline{x}$ stands for the string obtained by doubling all bits in x. This decompressor is prefix-free and $KP_D(x) = 2l(x) + 2$. By replacing $\overline{x}01$ by a more efficient self-delimiting encoding $\hat{x}$ we can obtain better upper bounds. For example, letting $\hat{x} = \overline{\mathrm{bin}(l(x))}01x$ we obtain the bound

$$KP'(x) \leqslant l(x) + 2\log l(x) + O(1).$$

  By iterating the construction, we obtain the bound

$$KP'(x) \leqslant l(x) + \log l(x) + 2\log\log l(x) + O(1)$$

  and so on.

- as well as the plain complexity, the prefix complexity does not increase when algorithmic transformation is applied:

$$KP'(A(x)) \leqslant KP'(x) + O(1).$$

  The constant $O(1)$ depends on A but does not depend on x. Indeed, if D is a prefix-stable decompressor then so is the composition $x \mapsto A(D(x))$. This is true for prefix-free decompressor as well, so we obtain a similar statement for KP' in place of KP. Using this property we can define prefix complexity of other constructive objects like pairs of strings, natural numbers, finite sets of strings etc., without specifying how to encode them by binary strings.

- For the prefix complexity, the inequality comparing the complexity of a pair of strings with their separate complexities is true up to a constant additive error term rather than logarithmic one:

$$KP(x,y) \leqslant KP(x) + KP(y) + O(1)$$

  (see below Theorem 54 in Section 4.6, p. 86).

- Let D be an optimal decompressor (from the definition of the plain complexity). Since the the transformation $p \mapsto D(p)$ does not increase complexity, we have

$$KP\,(D(p)) \leqslant KP\,(p) + O(1) \leqslant l(p) + 2\log l(p) + O(1).$$

Let p be a shortest description of x with respect to D, that is, $D(p) = x$ and $l(p) = KS\,(x)$. Then we have

$$KP\,(x) = KP\,(D(p)) \leqslant l(p) + 2\log l(p) + O(1) = KS\,(x) + 2\log KS\,(x) + O(1).$$

Using stronger bounds in place of the bound $KP\,(p) \leqslant l(p) + 2\log l(p) + O(1)$ we obtain the inequality

$$KP\,(x) \leqslant KS\,(x) + \log KS\,(x) + 2\log\log KS\,(x) + O(1)$$

and other similar inequalities.

4.4 A digression: machines with self-delimiting input

{prefix-sd}

This section is not used in the sequel. We analyze here different computational models with a "self-delimiting" input. Such models provide a motivation for the notions of prefix-stable and prefix-free functions.

Usually the input is given to a machine in such a way that the machine knows where the input string starts and ends. For example, defining a Turing machine computation we usually assume that initially the head is located at the first symbol of the input string and that its last symbol is followed be a special marker, say, a blank.

At the other hand, a machine with a self-delimiting input receives the input bits one by one and has no indication which of them is the last one. At certain time it should print a result and halt.

4.4.1 Prefix stable functions

Here is a refinement of this idea. Consider Turing machine that has an extra infinite one-way read-only *input tape*. The leftmost cell of the tape contains a special marker #. All the other cells contain either 0 or 1 (Fig. 10).

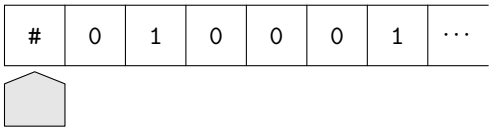

Figure 10: A head on a one-way input tape.

{read-only-

Initially the input tape head is located in the leftmost cell and thus scans the marker. The instruction performed by the machine is determined by the symbol it scans (and also by the symbol scanned on the work tape and machine's internal state, as usual). The possible actions are: changing the internal state, writing a symbol on the work tape, moving the heads (in any direction on the

work tape and to the right on the input tape). The result of the computation should be written on the work tape in the usual way. The work tape is initially empty.

Let M be a Turing machine as described above. For all possible contents of the input tape run the machine. If the computation halts, write down two stings: the string x consisting of all bits scanned by the input head, and the result y of the computation. Let Γ_M denote the resulting set of pairs $\langle x,y\rangle$. If the pairs $\langle x_1,y_1\rangle$ and $\langle x_2,y_2\rangle$ are in Γ_M, then the strings x_1 and x_2 are incompatible. Indeed, assume that x_1 is a prefix of x_2. Since the computation on x_1 does not go outside x_1, it will be valid for x_2 too, and the last bits of x_2 remain unused, thus the pair $\langle x_2,y_2\rangle$ does not belong to Γ_M.

In particular, the first components of different pairs in Γ_M are different. This means that Γ_M is a graph of a function. We denote this function by γ_M. Its arguments and values are binary strings. We say that M computes γ_M *in a prefix-free mode*. It is easy to see that the function γ_M is computable in the usual sense. Indeed, to compute $\gamma_M(x)$ we write x on the input tape and any symbols (say, zeros) to the right of x and then run M. If M halts and prints a result y, we verify whether it has scanned all symbols of x and no symbols beyond x. If the verification fails, we output no result, otherwise we print y and halt.

It is easy to see that the function γ_M is prefix-free (any two different strings in its domain are incompatible). The converse statement is true as well:

{prefix-fre

Theorem 44 *Every computable prefix-free function is computed by some machine in a prefix-free mode.*

$\lhd$ This statement is not that evident. Indeed, the (standard) machine computing a prefix-free function f knows where the input ends and can use this information. We need to construct another machine M such that $\gamma_M = f$.

The machine M works as follows. Fix a machine computing f in the usual sense. We simulate in parallel its computations on all possible inputs. Sometimes we will interrupt the simulation and scan a new symbol from the input tape. More specifically, when a new pair $\langle x,y\rangle$ with $f(x) = y$ appears, we compare x with the already scanned part r of the input tape. If r is not a prefix of x then we do nothing and wait until the next pair $\langle x,y\rangle$ appears. If r coincides with x, we output y and halt. Otherwise r is a proper prefix of x. In this case we read the input tape until we find the first bit where x differs from the contents of the input tape or we find out that the input tape begins with x. In the latter case we output y and halt. In the former case we return to the simulation process and continue it until the next pair $\langle x,y\rangle$ appears.

How does M start its work? Initially the scanned part of the input tape is empty. Once the first pair $\langle x,y\rangle$ appears, we look whether x is empty or not. If x is empty, we print y and halt. Otherwise we scan the input tape until we read x or find the first bit where x differs from the contents of the input tape (finding out that x is not a prefix of the input). In the first case we print y and halt. In the second case we wait for the next pair $\langle x,y\rangle$.

Formally speaking, we maintain the following invariant relation: after processing each pair, if r is the scanned part of the input tape, then either

(1) $f(r)$ is defined and the machine halts with the output $f(r)$, or

(2) r is not a prefix of x for all pairs $\langle x,y\rangle$ appeared so far, but every proper prefix r' of r is a proper prefix of one of such x's.

(A proper prefix of a string is any its prefix that is different from the string itself.)

It is easy to verify that this invariant relation implies that $f = \gamma_M$. We skip this verification and explain informally the main idea of the construction: if the scanned part r of the input is a proper prefix of a string in the domain of f then $f(r)$ is undefined and we can safely read the next bit of the input. $\triangleright$

An equivalent model can be defined in more "practical" terms. Consider computer programs that have instructions of the form

$$b := \mathit{NextBit}.$$

Executing this instruction, the program prints on the screen a prompt like "Enter the next bit" and waits until the user hits one of the keys "0" and "1". After he does this, the input bit is recorded in b and the computation resumes.

One can assign computable function f to every program of this type. Namely, $f(x)$ equals to y if the program prints y provided the user enters the bits of x successively in response to the program's prompts. If the program prints the result before the user enters all the bits of x or it asks for a new bit after all the bits of x are entered, then $f(x)$ is undefined.

It is easy to modify the arguments above to prove that programs of this type compute all the prefix-free functions and no other. (Moving the input head to the right is just reading the next input bit.)

4.4.2 Prefix stable functions

{prefix-nor

There is another way to enter a bit string into a program without specifying where the string ends: by pressing the "0" and "1" keys (and no other keys, so the end of the input is not specified). The input are stored in a queue that is accessible to the program.

To read a new input bit the program invokes the instruction

$$b := \mathit{NextBit}$$

This instruction removes the first (the oldest) bit from the queue and assigns it to the variable b. The program may use also the instruction

$$b := \mathit{NextExists}$$

to find out whether the queue is non-empty. We need to specify what happens if the program invokes the instruction *NextBit* when the queue is empty. We may agree that this causes a crash, or that the computation is delayed until the next bit arrives. It is not essential which of these two options is chosen, since we may guard the input statement by a waiting loop:

while not *NextExists* **do** {nothing};
$b := \mathit{NextBit}$

Programmers would this kind of access a "non-blocking read operation", while the input mode described in the previous section is "blocking read operation". The advantage of a non-blocking read operation is that we can do some useful work while waiting for the next input bit.

It is not clear how to define a function computed by a program that uses non-blocking read operation, since the output of of the program may depend not only on the input string, but also on timing.

We call a program *robust* if this is not the case (i.e., if the output is determined by the input string and does not depend on timing). If the program is robust, for any input string x there are two possibilities: (1) the program does not halt for any delays between the consecutive bits of x; or (2) for some y, the program outputs y for any delays between the consecutive bits of x.

In this way every robust program *computes* a function f such that $f(x)$ is undefined in the first case and equals y in the second case.

{prefix-co

Theorem 45 (**a**) *The function computed by a robust program is both computable and prefix-stable.* (**b**) *For every computable prefix-stable function there exists a robust program that computes it.*

$\lhd$ (a) The computability of f is straightforward: to compute $f(x)$ we start our robust program and enter all the bits of x (with arbitrary delays). Then we wait until the program prints a result, which by assumption is equal to $f(x)$ if f is defined on x and does not exist otherwise.

Let us prove that f is prefix-stable. We have to show (recall the definition from Section 4.3) that if a robust program prints y on some input x then it prints y on every input x' that is a continuation of x. Start the program and enter all the bits of x (with arbitrary delays). By assumption the program prints y and then halts. After that, input all the remaining bits of x' (the difference between x' and x) with arbitrary delays. Obviously, these extra bits do not affect the output of the program. Thus the program produces output y for input x' at least for some timing. (Being robust, it does the same for any timing.)

(b) Let f be a computable prefix-stable function f. The robust program r that computes f works as follows:

Using a (non-robust) algorithm that computes f, program r computes in parallel $f(x)$ for all inputs x. At the same time r reads all available input bits. Doing this, r looks for strings x and y such that $f(x) = y$ and x is a prefix of the input sequence. Once such pair $\langle x,y\rangle$ is found, program r prints y and halts.

Assume that $f(x) = y$ and all the bits of x are entered (with some delays). We have to prove that r prints y and halts whatever the delays are. Indeed, at certain time r knows that $f(x) = y$ and all the bits of x have been entered. At that time the program prints y and halts unless it has been halted earlier. The latter indeed can happen: the program can halt earlier with the result $f(x')$ where x' is some string compatible with x. However, since f is assumed to be prefix-stable, we have $f(x') = y$ and the output is the same.

If $f(x)$ is undefined and f is prefix-stable then $f(x')$ is undefined for all prefixes x' of x, hence the program does not terminate. $\rhd$

This theorem provides a motivation for the notion of a prefix-stable function.

68 Construct an algorithm that transforms any given program p using *NextBit* and *NextExists* calls into a robust program p' that computes the same function as p does, if p is robust (and some prefix-stable function if p is not).

[Hint: Use the construction from the proof of Theorem 45 back and forth.]

69 (Continued.) Prove that there exist no algorithm that for any given program p decides whether p is robust or not.

[Hint: This can be done in a standard way, by reducing the halting problem. See, e.g., [**?**]]

4.4.3 Continuous computable mappings

{prefix-sd-

There is another, more abstract, motivation for the notion of a prefix-stable function. It goes back to a general theory of computable functionals of higher type, but we restrict our attention to our particular case.

Let Σ denote the set of all finite and infinite binary sequences: $\Sigma = \Xi \cup \Omega$. For a finite string x let Σ_x denote the set of all finite and infinite continuations of x. We will consider Σ as a partially ordered set: $x \leqslant y$ if x is a prefix of y.

Consider a topology on Σ whose base consists of all sets of the form Σ_x. This means that a set is open if it is a union of some sets of this form. It is easy to verify that we indeed get a topology. (Note that the resulting topological space does not satisfy the separation axiom.)

The following statement is almost straightforward:

Theorem 46 *A set $A \subset \Sigma$ is open if and only if it satisfies the following conditions:*

(1) *if a finite string x is in A, then all finite and infinite continuations of x are in A;*

(2) *if an infinite sequence is in A, then some its finite prefix is in A.*

$\lhd$ Every union of base sets satisfies the conditions (1) and (2). Conversely, if a set A satisfies both conditions then it is equal to the union of Σ_x over all finite strings x in A. $\rhd$

Add to the natural numbers a new element $\perp$ ("undefined") and let $\mathbb{N}_\perp$ denote the resulting set. Consider the following partial order on this set: the element $\perp$ is less than all natural numbers, and all the natural numbers are pairwise incomparable (Fig. 11).

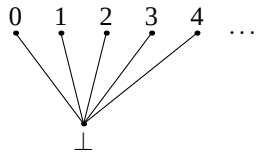

Figure 11: The topological space $\mathbb{N}_\perp$ {n-bottom}

Consider the following topology on the set $\mathbb{N} \cup \{\perp\}$. A set is open if it either does not include the element $\perp$ or coincides with $\mathbb{N} \cup \{\perp\}$. It is easy to verify that we get a topological space (that does not satisfy the separation axiom).

Let us identify partial mappings from Σ into $\mathbb{N}$ with total mappings from Σ into $\mathbb{N}_\perp$; the value $\perp$ replaces all undefined values. The next theorem characterizes continuous mappings (recall that a mapping is continuous if the preimage of every open set is open).

Theorem 47 *A* (*total*) *mapping $F : \Sigma \to \mathbb{N}_\perp$ is continuous if and only if the following are true:*

(1) *F is increasing, i.e., $x \leqslant y$ implies $F(x) \leqslant F(y)$ (the signs $\leqslant$ refer to the pre-ordering relations on $\mathbb{N}_\perp$ and Σ introduced above);*

(2) *if x is an infinite binary sequence and $F(x) \neq \perp$, then x has a finite prefix x' such that $F(x') \neq \perp$.*

$\lhd$ Let F be a continuous mapping. To verify the condition (1), assume that $x \leqslant y$ but $F(x) \not\leqslant F(y)$. Then $F(x)$ is a natural number (and not $\perp$) and $F(x) \neq F(y)$. The preimage of the open set $\{F(x)\}$ contains x and does not contain y hence it is not open.

Let us verify the condition (2). Assume that x is an infinite sequence and $F(x) \neq \perp$. The preimage of the set $\{F(x)\}$ is open and contains x. Thus it contains some finite prefix of x.

It remains to verify that any function F satisfying conditions (1) and (2) is continuous. We need to verify only that the preimage of every natural number is open (indeed, the preimage of the entire space is open and other open sets are unions of singletons formed by natural numbers). It is enough to verify that the preimage of every natural number satisfies the conditions (1) and (2) from the previous theorem. This is a straightforward corollary of our assumptions. (Note that if x' is a prefix of x and $F(x') \neq \perp$ then $F(x') = F(x)$, as F is increasing.) $\rhd$

For any given continuous mapping $F : \Sigma \to \mathbb{N}_\perp$ consider the set Γ_F of all pairs $\langle x, n\rangle \in \Xi \times \mathbb{N}$ such that $F(x) = n$. Note that the set Γ_F is only a part of the graph of the mapping F (we consider only finite strings x and require that $n \neq \perp$).

Theorem 48 *The mapping $F \mapsto \Gamma_F$ is a bijection between continuous mappings $\Sigma \to \mathbb{N}_\perp$ and sets $A \subset \Xi \times \mathbb{N}$ satisfying the following conditions:*

(1) $\langle x, n\rangle \in A,\ x \leqslant y \ \Rightarrow\ \langle y, n\rangle \in A$;

(2) $\langle x, n\rangle \in A,\ \langle x, m\rangle \in A \ \Rightarrow\ m = n$.

$\lhd$ Assume that the mapping F is continuous. If $F(x) = n \in \mathbb{N}$ then the condition (1) of the previous theorem guarantees that $F(y) = n$ for every $y \geqslant x$. This proves that the set Γ_F satisfies the condition (1). As $F(x)$ cannot be equal to two different numbers, the condition (2) is also satisfied. Thus, for every continuous mapping F the set Γ_F has properties (1) and (2).

It is easy to see that the set Γ_F uniquely determines F: if x is a finite string then $F(x)$ is the second component of the (unique) pair $\langle x, n\rangle \in \Gamma_F$. If there is no such pair then $F(x) = \perp$. If x is an infinite sequence then $F(x)$ is determined uniquely as $F(x')$ where x' is a sufficiently long prefix of x.

It remains to show that every set A having properties (1) and (2) is equal to Γ_F for certain F. For every finite x define $F(x)$ as the natural number n such that $\langle x, n\rangle \in A$, which is unique by property (2). If there is no such n then let $F(x) = \perp$. By condition (1) we get an increasing function. For every infinite $x \in \Sigma$ let $F(x)$ be equal to $F(x')$ where x' is any prefix of x such that $F(x') \neq \perp$. If there is no such x' then let $F(x) = \perp$. By property (1) the value of $F(x)$ is well defined. The constructed function F satisfies both conditions (1) and (2) from the previous theorem and is continuous. By construction we have $\Gamma_F = A$. $\rhd$

The conditions (1) and (2) mean that the set A is a graph of a prefix-stable function. We thus have a one-to-one correspondence between continuous mappings $\Sigma \to \mathbb{N}_\perp$ and prefix-stable functions.

Call a continuous mapping $F\colon \Sigma \to \mathbb{N}_\perp$ *computable* if the set Γ_F is enumerable. It is easy to verify that F is computable if and only if the restriction of F to those strings $x \in \Xi$ for which $f(x) \neq \perp$ is computable in the standard sense. (A partial function from Ξ to $\mathbb{N}$ is computable if and only if its graph is enumerable.) Thus computable continuous functions $\Sigma \to \mathbb{N}_\perp$ are basically the same as prefix-stable functions. This gives an extra motivation for the notion of a computable prefix-stable function.

4.5 The main theorem on prefix complexity

{prefix-eq}

In this section, we prove that all the three complexity measures, KP (prefix-stable decompressors), KP' (prefix-free decompressors) and the negative logarithm of the a priori probability coincide up to an additive constant. To this end we prove that three inequalities

$$-\log m(x) \leqslant KP(x) \leqslant KP'(x) \leqslant -\log m(x)$$

are true up to a constant error term. We start with two easy inequalities.

{kp-kpprime

Theorem 49

$$KP(x) \leqslant KP'(x) + O(1)$$

$\lhd$ This inequality would be evident if every prefix-free function were prefix-stable. This is not the case: a prefix-free function D is undefined on all the continuations of any string u in the domain of D. In contrast, a prefix-stable function D is defined on all the continuations v of any string u in the domain of D, and $D(v) = D(u)$.

Therefore we need a (simple) construction. Let D be a prefix-free decompressor. Define another decompressor D' as follows: $D'(y) = x$ if and only if $D(y') = x$ for some prefix y' of y. As D is prefix-free, such y' is unique, thus D' is well defined. To compute $D'(y)$ we just apply D in parallel to all the prefixes y' of y until we find a prefix y' such that $D(y')$ is defined.

By construction the function D' is prefix-stable and extends D. Therefore the complexity of any string with respect to D' does not exceed that with respect to D. (In fact, the complexities with respect to D and D' coincide, as the described transformation $D \mapsto D'$ does not affect shortest descriptions.) $\rhd$

We could try to prove the converse inequality in a similar way: consider the restriction of the given prefix-stable decompressor D to minimal descriptions. That is, let $D'(y) = z$ if $D(y) = z$ and $D(y')$ is undefined for all proper prefixes y' of y.

Note that this transformation is an inverse of the transformation used in the proof of the last theorem. The resulting function D' is indeed prefix-free. However it might be non-computable.

70 Find a computable prefix-stable function D for which the prefix-free function D' constructed in this way is not computable. [Hint: Let A be an enumerable undecidable set, whose complement is thus not enumerable. Let $f(0^n 11x) = 0$ for all natural numbers n and all binary strings x. Let also $f(0^n 1x) = 0$ for all $n \in A$ and all x.]

This problem shows that, in a sense, the non-blocking read operation is more powerful than the blocking one (see Section 4.4).

{m-kp}

Theorem 50

$$-\log m(x) \leqslant KP(x) + O(1).$$

$\lhd$ We have to prove that $2^{-KP(x)} \leqslant cm(x)$ for some constant c and for all x. Recall that m is the maximal lower semicomputable semimeasure. Thus it suffices to find an upper bound for the function $x \mapsto 2^{-KP(x)}$ that is a lower semicomputable semimeasure. (In this section we consider semimeasures on the set of all binary strings treated as isolated objects, as defined in Section 4.1.)

Let us construct a probabilistic machine generating this semimeasure. Toss a coin to obtain a sequence $b_0, b_1, b_2, \ldots$ of random bits. Simultaneously, apply the optimal prefix-stable decompressor D (from the definition of KP) to all prefixes of the sequence $b_0, b_1, b_2, \ldots$. If one of the computations

$$D(\Lambda), D(b_0), D(b_0 b_1), D(b_0 b_1 b_2), \ldots$$

terminates with a certain result, print that result and halt. Note that it does not matter which of the terminated computations we choose: prefix-stability of D guarantees that this choice does not affect the result.

Let x be a binary string and let p be a shortest description of x with respect to D. Then the machine prints x with probability at least $2^{-l(p)}$. Indeed, if the random sequence starts with p then the result of the machine is x. Thus the constructed machine generates a measure that is an upper bound for $2^{-KP(x)}$. $\rhd$

There is a slightly different proof of the same theorem, which does not involve probabilistic machines. The function $x \mapsto 2^{-KP(x)}$ is lower semicomputable. Thus it is enough to show that it is a semimeasure.

{prefix-coc

Theorem 51

$$\sum_x 2^{-KP(x)} \leqslant 1.$$

$\lhd$ For every string x let p_x be any shortest description of x (with respect to the optimal prefix-stable function from the definition of KP). For every different strings x and y the strings p_x and p_y are incompatible. Thus the statement is a direct corollary of the following

Lemma. Let $p_0, p_1, p_2, \ldots$ be pairwise incompatible strings (that is, neither of the strings is a prefix of another one). Then $\sum_i 2^{-l(p_i)} \leqslant 1$.

Indeed, for every i consider the set Ω_{p_i} of all infinite continuations of p_i. Its uniform Bernoulli measure is equal to $2^{-l(p_i)}$. As the strings p_i are pairwise incompatible, the sum of the measures of all sets Ω_{p_i} is at most 1. The Lemma and Theorem 51 are proved. $\rhd$

Theorem 51 implies that the inequality $KP(x) \leqslant l(x) + O(1)$ is false (and shows the difference between plain complexity KS and prefix complexity KP) Indeed, if it were true, the series

$$\sum_x 2^{-l(x)}$$

would converge. However for every n the terms of this series corresponding to strings x of length n sum up to 1 (there are 2^n such terms and each of them is equal to 2^{-n}).

71 Prove that even a weaker inequality $KP(x) \leqslant l(x) + \log l(x) + O(1)$ is false (in other words, the difference $KP(x) - l(x) - \log l(x)$ is not bounded by a constant). [Hint: Use the divergence of the harmonic series.]

It remains to prove the last (and most difficult) inequality:

{kpprime-m}

Theorem 52

$$KP'(x) \leqslant -\log m(x) + O(1).$$

$\lhd$ We present first a sketch of the proof. The semimeasure $m(x)$ is lower semicomputable, so we can generate lower bounds for $m(x)$ that converge to $m(x)$ but no estimates for the approximation error are given. The larger $m(x)$ is, the smaller $KP'(x)$ should be, that is, the shorter description p we have to provide for x. The descriptions reserved for different strings must be incompatible. (The descriptions p_1 and p_2 are incompatible if the intervals I_{p_1} and I_{p_2} do not overlap. Recall that the interval I_p consists of all real numbers whose binary expansion begins with p.) The inequality $l(p) \leqslant -\log_2 m(x)$ means that the length of the interval I_p is at least $m(x)$: $2^{-l(p)} \geqslant m(x)$.

Thus we have to assign to every string x an interval of length at least $m(x)$ so that the intervals assigned to different strings do not overlap.

Let us specify more carefully what we need. First, for each x it suffices to reserve an interval of the length $\varepsilon m(x)$ rather than $m(x)$, for some fixed positive ε. This relaxation causes the complexity increase at most by a constant.

Second, we are allowed to use only properly aligned intervals, i.e., intervals I_p for some binary string p. However, given the above relaxation, this restriction is not essential. Indeed, every interval $I \subset [0,1]$ contains a properly aligned interval that is at most four times shorter.

So we arrive to a problem that is quite similar to the problem considered in Section 4.1. There is a sequence of clients. Each client asks for some space inside $[0,1]$; client may increase its request from time to time. The important difference is that now the client are interested not in the total space allocated, but in the contiguous interval, which makes our "space management" job more difficult. To compensate this difficulty, we are allowed to reduce all the requests and multiply them by some constant ε.

Imagine that clients are processes running on a computer, and the memory manager has to allocate contiguous properly allocated memory according to their requests that increase in time. Once allocated memory cannot be freed (and reused for other process).

The simplest strategy is to allocate a new interval (in the free memory) each time the request increases. This does not work, however: if two clients' requests increase in alternating order and in small steps, the overhead cannot be compensated by any fixed ε, and we will run out of space.

The remedy is well known: one should look forward and increase the allocated interval significantly even if the current increase in the request is small. For example, one may allow only powers of 2 as the interval lengths (then the sum of the lengths is at most twice more than the maximal summand).

It is not hard to present a detailed proof based on this strategy, but we will not do that. Instead, we present a slightly different proof that uses the following statement(often called Kraft–Chaitin lemma, see **??**).

This lemma can be considered as a computable version of the Kraft theorem from the information theory (see p. 170).

Lemma. Let $l_0, l_1, l_2, \ldots$ be a computable sequence of non-negative integers such that

$$\sum_i 2^{-l_i} \leqslant 1.$$

There exists a computable sequence of pairwise incompatible binary strings $x_0, x_1, x_2, \ldots$ such that $l(x_i) = l_i$.

Note that the inequality of the lemma is a necessary condition for the existence of such a sequence: the intervals I_{x_i} do not overlap and their lengths are equal to 2^{-l_i}. The lemma states that this necessary condition is also sufficient.

So we again have an infinite sequence of clients, the ith client demands to allocate properly aligned interval of length 2^{-l_i} for her. The intervals reserved for different clients should not overlap. We need to design a computable strategy to fulfill all the client's requests.

There are several ways to describe such a strategy. Here is probably the simplest one: let us maintain the representation of the free space (part of $[0,1]$ that is not allocated) as the union of properly aligned intervals of different lengths.

Initially this list contains one interval $[0,1]$. We serve the requests $l_0, l_1, l_2, \ldots$ sequentially.

Assume that current request is l_i, so the required length is $w = 2^{-l_i}$. First note that one of the free intervals has length at least w. Indeed, if all the free intervals had smaller lengths, their sum (the total amount of free space) would be less than w since they have different lengths and the sum of powers of 2 less that $w = 2^{-l}$ is less than w.

If there is a free interval in the list that has size exactly w, our task is simple. We just allocate this interval and delete it from the free list (maintaining the invariant relation).

Assume that this is not the case. Then we have some intervals in the list that are bigger than requested. Using the best fit strategy, we take the smallest among these intervals. Let $w' > w$ be its length. Then we split free interval of size w' into properly aligned intervals of size $w, w, 2w, 4w, 8w, \ldots, w'/2$ (note that $w + w + 2w + 4w + 8w + \ldots + w'/2 = w'$. The first interval (of size w) is allocated, all the other intervals are added to the free list. We have to check out invariant relation: all new intervals in the list have different sizes starting with w and up to $w'/2$; old free intervals cannot have this size since w' was the best fit in the list.

Lemma is proved.

72 Prove that the described algorithm can be rephrased as follows: for each i use the the leftmost properly aligned interval of length 2^{-l_i} that does not overlap with previously allocated interval. [Hint: the construction used in the proof maintains also the following property: the lengths of the free intervals increase from left to right.]

Corollary. Let l_i be a computable sequence of natural numbers such that $\sum_i 2^{-l_i} \leqslant 1$. Then $KP'(i) \leqslant l_i + O(1)$.

Indeed, the Lemma provides a computable sequence of pairwise incompatible strings x_i of lengths l_i. Define a computable function D by letting $D(x_i) = i$. As x_i are pairwise incompatible, this function is prefix-free. And D is computable: given an input x we compare it with x_i for all $i = 0, 1, 2, \ldots$ successively. Once we find that $x = x_i$ we output i and halt.

(Note that, in this proof, we go back and forth between natural numbers and binary strings when we speak about the a priori probability and complexity.)

Let us return to the proof of the theorem. Consider the maximal lower semicomputable semimeasure m. By definition there exists a computable function $m(x,i)$ taking rational values that is non-decreasing in i such that

$$m(x) = \lim_{i\to\infty} m(x,i).$$

Let $m'(x,i)$ stand for the smallest power of two $(1, 1/2, 1/4, 1/8, \ldots)$ that is an upper bound for $m(x,i)$. The function $m'(x,i)$ is computable and non-decreasing in i. Its value is between $m(x,i)$ and $2m(x,i)$.

Say that a pair $\langle x,i\rangle$ is a *boundary* pair if $m'(x,i) > m'(x,i-1)$ (or if $i=0$ and $m'(x,0)>0$).

Let us show that the sum of $m'(x,i)$ over all boundary pairs $\langle x,i\rangle$ does not exceed 4. It is enough to show that for every fixed x the sum of $m'(x,i)$ over all boundary pairs $\langle x,i\rangle$ is at most $4m(x)$. This is true since for every fixed x each term in this sum is at least twice bigger than the preceding term. Thus the sum is at most twice bigger than its last term, $m'(x,i)$ for some i, which is less than $2m(x,i)$. Now recall that $m(x,i) \leqslant m(x)$. We see that the sum in question is at most $4m(x)$.

The set of all boundary pairs $\langle x,i\rangle$ is decidable: to find whether a pair $\langle x,i\rangle$ is a boundary pair we have to compare $m'(x,i)$ and $m'(x,i-1)$.

Enumerate all the pairs $\langle x,i\rangle$ and exclude all non-boundary ones. Let $\langle x_0,i_0\rangle, \langle x_1,i_1\rangle, \ldots$ be for the resulting sequence. Each boundary pair appears in this sequence exactly once. Define l_n by the equality

$$2^{-l_n} = m'(x_n,i_n)/4.$$

The sequence of l_n is computable and

$$\sum_n 2^{-l_n} = \frac{1}{4}\sum_n m'(x_n,i_n) \leqslant 1.$$

The corollary mentioned above implies that $KP'(n) \leqslant l_n + O(1)$. As x_n can be computed given n, we have

$$KP'(x_n) \leqslant KP'(n) + O(1) \leqslant l_n + O(1) = -\log m'(x_n,i_n) + O(1).$$

So for every x the complexity $KP'(x)$ does not exceed $-\log m'(x,i)$ if $\langle x,i\rangle$ is a boundary pair. Taking the maximal i with this property we get $-\log m(x) + O(1)$, therefore

$$KP'(x) \leqslant -\log m(x) + O(1).$$

Theorem is proved. $\triangleright$

So all the values KP, KP' and $-\log m$ differ by at most a constant. Given this, we do not distinguish in the sequel between KP and KP' (unless the difference in their definitions becomes essential for some special reason).

Let us note that actually we have proved the following statement, needed in Section 5.6:

{prefix-exp

Theorem 53 *Given a lower semicomputable sequence of reals $p_0, p_1, \ldots$ such that $\sum_i p_i \leqslant 1$, we can find a prefix-free decompressor D such that $KP'_D(i) \leqslant -\log_2 p_i + 2$.*

This means that given any algorithm enumerating the set of pairs $\langle r,i\rangle$ with $r < p_i$, we can find an algorithm for a decompressor D satisfying the latter inequality.

4.6 Properties of prefix complexity

{prefix-pr}

In this section we continue the study of the of prefix complexity. We first revisit some already established properties and present their alternative proofs based on the a priori probability.

It is well known that the series $\sum 1/n^2$ converges. Multiplying its terms by a constant, we obtain a lower semicomputable semimeasure. Thus the a priori probability of a natural number n is at least c/n^2 for some constant c. This implies that

$$KP(n) \leqslant 2\log n + O(1).$$

Let x_n be the nth string in the sequence $\Lambda, 0, 1, 00, 01, 10, 11, 000, \ldots$ of all binary strings. Then

$$KP(x_n) \leqslant KP(n) + O(1) \leqslant 2\log n + O(1) = 2l(x_n) + O(1)$$

(the last equality is true, since x_n is $n+1$ in binary notation without the leading 1, so the length of x_n is $\log n + O(1)$).

(There is a special case $n = 0$, as both $1/0^2$ and $\log 0$ are undefined; the changes needed to handle it are trivial.)

So we get the inequality $KP(x) \leqslant 2l(x) + O(1)$.

To prove a better upper bound for prefix complexity we may consider a convergent series

$$\sum \frac{1}{n\log^2 n}.$$

(To prove its convergence compare it with the corresponding integral.) Using this series, we obtain the inequality $KP(n) \leqslant \log n + 2\log\log n + O(1)$ or (for strings)

$$KP(x) \leqslant l(x) + 2\log l(x) + O(1)$$

(for the alternative proof of this inequality see p. 74).

By using the series $\sum 1/(n\log n(\log\log n)^2)$, $\sum 1/(n\log n\log\log n(\log\log\log n)^2)$ etc. we can improve the bound further.

Now we prove the inequality relating the prefix complexity of a pair to prefix complexities of its components.

{prefix-pai

Theorem 54

$$KP(x,y) \leqslant KP(x) + KP(y) + O(1).$$

Just as in the case of plain complexity, we define $KP(x,y)$ as the complexity of the string $[x,y]$ where $\langle x,y\rangle \mapsto [x,y]$ is a computable injective encoding of pairs of binary strings. (The complexity of a pair does depend on the choice of the encoding; switching to another computable injective encoding changes complexity at most by an additive constant. Indeed, the translation between any two computable injective encodings is an algorithmic transformation.)

$\lhd$ Consider the function m' defined as

$$m'([x,y]) = m(x)m(y)$$

(here x and y are binary strings, $[x,y]$ is the encoding of the pair and the values of m' is a real number). Here m stands for the a priori probability. If z is not an encoding of any pair, we let $m'(z) = 0$.

The function m' is lower semicomputable (the product of lower bounds for $m(x)$ and $m(y)$ is a lower bound for $m(x)m(y)$). Furthermore, we have

$$\sum_z m'(z) = \sum_{x,y} m'([x,y]) = \sum_{x,y} m(x)m(y) = \sum_x m(x) \sum_y m(y) \leqslant 1 \cdot 1 = 1.$$

Thus m' is a lower semicomputable semimeasure. Comparing m' with the a priori probability, we obtain the inequality $m'([x,y]) \leqslant cm([x,y])$ for some constant c. Hence

$$KP([x,y]) \leqslant KP(x) + KP(y) + O(1).$$

Theorem is proved. $\triangleright$

73 Prove that the sum $\sum_y m([x,y])$ differs from $m(x)$ by at most a constant factor (in both directions). {m-projecti

74 Let $f\colon \mathbb{N} \to \mathbb{N}$ be a strictly increasing computable function. Prove that the value $\sum\{m(k) \mid f(n) \leqslant k < f(n+1)\}$ differs from $m(n)$ at most by a constant factor. (So if we split the series $\sum_n m(n)$ into groups, the sums of the groups form essentially the same series!)

Let us prove now Theorem 54 using decompressors. It turns out that we need to use prefix-free (and not prefix-stable) decompressors.

Let us prove that $KP'([x,y]) \leqslant KP'(x) + KP'(y) + O(1)$. Let D be an optimal prefix-free decompressor used in the definition of KP'. Define a new prefix-free decompressor D'. Informally, the algorithm D' reads the input until it finds a description of x. Then it reads the rest of the input until it finds a description of y. Formally, we define D' as

$$D'(pq) = [D(p), D(q)].$$

Here pq stands for the concatenation of strings p and q. In other words, we try to split the input into two parts p and q in such a way that both $D(p)$ and $D(q)$ are defined.

We need to verify that D' is well defined. Indeed, assume that x is represented as pq in two different ways, $x = pq = p'q'$, and all the values $D(p)$, $D(q)$, $D(p')$, $D(q')$ are defined. Then p and p' are compatible (being prefixes of the same string x) and thus coincide (as D is prefix-free), hence $q = q'$.

In a similar way we can prove that the function D' is prefix-free. Let pq be a prefix of $p'q'$ and both belong to the domain of D. The strings p and p' are compatible and both $D(p)$ and $D(p')$ are defined, therefore $p = p'$. This implies that q is a prefix of q'. As both $D(q)$ and $D(q')$ are defined, we have $q = q'$.

The function D' is computable: to find $D'(x)$ we compute in parallel $D(p)$ and $D(q)$ for each possible way to split x into p and q. We have shown that there is at most one representation of x as pq such that $D(p)$ and $D(q)$ are defined. If we find such p and q, we output the string $[D(p), D(q)]$.

It remains to note that

$$KP_{D'}([x,y]) \leqslant KP_D(x) + KP_D(y).$$

Indeed, let p and q be shortest descriptions of x and y with respect to D. The string pq is a description of $[x,y]$ with respect to D' and has length $KP_D(x) + KP_D(y)$.

75 Prove Theorem 54 using the definition of prefix-free decompressors in terms of machines with blocking read operation (see Theorem 44 on p. 76).

76 A set of binary strings is called *prefix-free* if any two elements of it are of incompatible. Show that if both sets A and B are prefix-free then so is the set

$$AB = \{ab \mid a \in A, b \in B\}.$$

Which proof of Theorem 54 (using a priori probability or using prefix-free decompressors) is easier and more natural? It is a matter of taste — the authors believe that the first one is more natural. The next theorem provides an opposite example: encoding arguments here seem to be simpler than the arguments using the a priori probability.

{prefix-add

Theorem 55

$$KP(x, KP(x)) = KP(x) + O(1).$$

(Problem 19 asks to prove the same equality for the plain complexity.)

The value $KP(x,n)$ (where x is a string and n is a natural number) is defined in the usual way, as the complexity $KP([x,n])$ of some (injective computable) encoding of the pair $\langle x,n\rangle$.

$\lhd$ The inequality $KP(x) \leqslant KP(x, KP(x)) + O(1)$ is straightforward, as the string x can by computed given the string $[x, KP(x)]$.

To prove the converse inequality let D be an optimal prefix-free decompressor used in the definition of prefix complexity KP'. Define a new decompressor D' as

$$D'(p) = [D(p), l(p)].$$

The domain of D coincides with that of D, hence D' is prefix-free. Let p be a shortest description of x with respect to D. Then $l(p) = KP'(x)$ and therefore p is a description of the string $[x, KP'(x)]$ with respect to D'. Thus $KP_{D'}([x, KP'(x)]) \leqslant l(p) = KP'(x)$.

Is the theorem proved? There is one subtle point in the argument. We have proved the theorem for the complexity KP', defined via prefix-free decompressors. If we substitute KP for KP' in the equality $KP'(x, KP'(x)) = KP'(x) + O(1)$, its right hand side will change by an additive constant. The similar statement for the left hand side is not straightforward, as KP' has two occurrences there, and the second one is inside the argument. But at least we have $KP(x, KP'(x)) = KP(x) + O(1)$.

To finish the proof it remains to show that the function $KP(x,n)$ changes at most by a constant, as n changes by 1. This easily follows from the computability of mappings $[x,n] \mapsto [x,n+1]$ and $[x,n] \mapsto [x,n-1]$. $\rhd$

It is instructive to prove Theorem 55 using the a priori probability. Let $m(x)$ be the a priori probability of x. Define the function m' as

$$m'([x,k]) = \begin{cases} 2^{-k} & \text{if } 2^{-k} < m(x); \\ 0 & \text{otherwise.} \end{cases}$$

This function is lower semicomputable: given x and k, we generate lower bounds for $m(x)$ and output 0 until we find that $2^{-k} < m(x)$, and then we we output 2^{-k}.

For every fixed x the sum of $m'([x,k])$ over all k is a geometric series formed by powers of 2. Therefore this sum is less than $2m(x)$ (the largest term of the series is less than $m(x)$). Therefore, the sum of $m'([x,k])$ over all x and k is finite. Comparing $m'([x,k])$ and the a priori probability of $[x,k]$ we conclude that

$$m(x,k) \geqslant 2^{-k+O(1)}$$

if $2^{-k} < m(x)$. Taking the logarithms, we see that

$$KP\,(x,k) \leqslant k + O(1)$$

whenever $2^{-k} < m(x)$. The latter inequality holds for $k = -\lfloor \log m(x) \rfloor + 1$ and thus we have

$$KP\,(x, -\lfloor \log m(x) \rfloor + 1) \leqslant KP\,(x) + O(1).$$

It remains to recall that the function $KP\,(x,n)$ changes at most by a constant, as n changes by 1. The second proof of Theorem 55 (in the nontrivial direction) is finished.

77 This argument proves a bit more: $KP\,(x,m) \leqslant m + O(1)$ whenever $KP\,(x) \leqslant m$. How to derive this inequality from Theorem 55 (from its statement and not from its proof)?

We proceed now to the algorithmic properties of the function $KP\,(x)$. Like the plain complexity the prefix complexity is upper semicomputable but not computable. Moreover, there is no computable non-trivial (i.e. unbounded) lower bound for $KP\,(x)$. Indeed, since $KP\,(x) \leqslant 2KS\,(x) + O(1)$, every non-trivial lower bound of KP would yield a non-trivial lower bound of KS.

Recall that the plain Kolmogorov complexity $KS\,(x)$ can be defined as the smallest upper semicomputable function K such that the cardinality of the set $\{x \mid K(x) < n\}$ is $O(2^n)$ for all n (Theorem 8, p. 21). Here is a similar statement for the prefix complexity:

{kp-minimal

Theorem 56 *The function KP is the smallest (up to an additive constant term) upper semicomputable function K (mapping binary strings to natural numbers and $+\infty$) such that the series $\sum_x 2^{-K(x)}$ converges.*

$\lhd$ The function KP is upper semicomputable and the series $\sum_x 2^{-KP\,(x)}$ converges. Let K be any other function having these properties. Then the function $M(x) = c2^{-K(x)}$ where c is a small enough constant is a lower semicomputable semimeasure. As $m(x)$ is the maximal lower semicomputable semimeasure, we have $M(x) = O(m(x))$, that is, $\log M(x) \leqslant \log m(x) + O(1)$. It follows that $KP\,(x) \leqslant K(x) + O(1)$. $\rhd$

This theorem can be reformulated as follows. For every upper semicomputable function f mapping binary strings to natural numbers and $+\infty$ the assertions "$KP\,(x) \leqslant f(x) + O(1)$" and "$\sum_x 2^{-f(x)} < \infty$" are equivalent.

Note that the requirement "the series $\sum_x 2^{-K(x)}$ converges" is stronger than the requirement "the number of x such that $K(x) \leqslant n$ is $O(2^n)$" used in Theorem 8. Indeed, if $\sum_x 2^{-K(x)} \leqslant C$, then

the number of x such that $K(x) \leqslant n$ is at most $C2^n$. This observation gives another proof of the inequality $KS(x) \leqslant KP(x) + O(1)$.

It is instructive to compare plain and prefix complexity in two aspects: the average complexity of strings of given length and the number of strings that have complexity not exceeing given bound. Let us start with the first question.

We have seen that the plain Kolmogorov complexity of most strings of length n is close to n (p. 11 and Problem 2). One could expect the prefix complexity to be slightly bigger.

{average-kp

Theorem 57 **(a)** $KP(x) \leqslant l(x) + KP(l(x)) + O(1))$.

(b) *For some constant c and for all n, d the fraction of strings x such that $KP(x) < n + KP(n) - d$ among all strings of length n is at most $c2^{-d}$.*

$\lhd$ (a) Let $m(x)$ be the a priory probability of a binary string x and $m(n)$ be the a priory probability of a natural number n. Consider the function $m'(x) = 2^{-n}m(n)$ where n is the length of x. The sum of $m'(x)$ over strings of length n is equal to $m(n)$ hence $\sum_x m'(x) \leqslant 1$. Since the function m' is lower semicomputable, we conclude that $m'(x) \leqslant cm(x)$ for some constant c and all x. Taking the logartihms we obtain the inequality

$$KP(x) \leqslant n + KP(n) + O(1)$$

(the constant $O(1)$ does not depend on n).

(b) Consider the function

$$m'(n) = \sum_{l(x)=n} m(x),$$

the total a priori probability of all strings of length n. Since $m'(n)$ is lower semicomputable and $\sum_n m'(n) \leqslant 1$, we have $m'(n) = O(m(n))$. On the other hand, the a priori probability of the string consisting of n zeros is at least $cm(n)$ for some positive contant c. Thus we have

$$c_1 m(n) \leqslant \sum_{l(x)=n} m(x) \leqslant c_2 m(n).$$

So the sum of $m(x)$ over all binary strings of length n coincides with $m(n)$ (up to a constant factor). Thus the average of $m(x)$ over all strings x of length n is $m(n)/2^n$ (up to a constant factor). The fraction of strings x such that $m(x)$ is 2^d times bigger than the average, is at most 2^{-d} (Chebyshev's inequality). $\rhd$

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

(Similar question for plain complexity is studied in Problem 3.)

Now we estimate the number of strings with complexity at most n.

{bounded-kp

Theorem 58 *The number of strings x with $KP(x) < n$ is $2^{n-KP(n)+O(1)}$.*

$\lhd$ Let c_n be the number of strings x such that $KP(x) < n$. Let us rewrite the basic property of prefix complexity (the convergence of the series $\sum 2^{-KP(x)}$) in terms of c_n. There are exactly $c_{n+1} - c_n$ strings of complexity n. Therefore the series

$$\sum_n 2^{-n}(c_{n+1} - c_n)$$

converges. Regrouping the terms of this series we conclude that

$$\sum_n (2^{-(n-1)} - 2^{-n})c_n = \sum_n 2^{-n}c_n < \infty.$$

Since the function c_n is lower semicomputable, this implies that $2^{-n}c_n$ does not exceed the a priori probability $m(n)$ of n. Hence $c_n \leqslant m(n)2^n$ (up to a constant factor).

On the other hand, it is easy to construct an upper semicomputable function K whose values are natural numbers and $+\infty$ that takes the value n on (approximately) $m(n)2^n$ arguments. This can be done in many ways. For example, let us agree that for a string x of length n the value $K(x)$ can be either $+\infty$ or n; it is equal to n if the ordinal number of x in the list of all n-bit strings is less than $m(n)2^n$.

For this function K, the series $\sum 2^{-K(x)}$ converges. Therefore, $KP(x) \leqslant K(x) + O(1)$ hence $c_{n+O(1)} \geqslant m(n)2^n$. Both $m(n)$ and 2^n change at most by a constant factor as n increases by 1. Thus $m(n)2^n = O(c_n)$. $\rhd$

The last two theorems show that the difference between $KP(x)$ and $KS(x)$ can be bounded in terms of the complexity of the length of x (not just logarithm of the length as we have seen before).

There are several other inequalities of this type. Iterating the inequality $KP(x) \leqslant l(x) + KP(l(x))$, we obtain the following series of inequalities:

$$KP(x) \leqslant l(x) + l(l(x)) + KP(l(l(x)) + O(1),$$
$$KP(x) \leqslant l(x) + l(l(x)) + l(l(l(x))) + KP(l(l(l(x))) + O(1)$$

etc. A similar series of inequalities can be obtained as follows. Let D be the optimal decompressor from the definition of the plain (not prefix) Kolmogorov complexity. Combining the inequalities $KP(D(y)) \leqslant KP(y) + O(1)$ and $KP(x) \leqslant l(x) + KP(x) + O(1)$ we get the following series of inequalities:

{kp-ks-bour

Theorem 59

$$KP(x) \leqslant KS(x) + KP(KS(x))) + O(1),$$
$$KP(x) \leqslant KS(x) + KS(KS(x))) + KP(KS(KS(x)) + O(1),$$
$$KP(x) \leqslant KS(x) + KS(KS(x))) + KS(KS(KS(x))) + KP(KS(KS(KS(x))) + O(1)$$

etc.

All the inequalities in this sequence can by obtained from the first one.

There are many other interesting relations between plain and prefix complexity, see **??**.

4.7 Conditional prefix complexity and complexity of a pair of strings

{prefix-co}

4.7.1 Conditional prefix complexity

{prefix-co-

What it conditional prefix complexity? Each of the definitions of prefix complexity can be modified by adding a condition.

We start with the definition using prefix-stable functions. A function $D(y,z)$ is *prefix-stable with respect to y* if for every z the function $y \mapsto D(y,z)$ is prefix-stable:

$$D(y,z) \text{ is defined and } y \leqslant y' \ \Rightarrow\ D(y',z) = D(y,z).$$

We assume here that the first argument of D is a binary string; the notation $y \leqslant y'$ means that y is a prefix of y'.

Recall the definition of the (plain) conditional complexity from Section 2.2. A *conditional decompressor* (=description mode) is a computable function that maps pairs of binary strings to binary strings. If $D(y,z) = x$ then y is called a *description of x when z is known*. The complexity of x with condition z is the length of the shortest description. Then we fix an optimal conditional decompressor that gives minimal complexity (up to a constant).

Now we consider only decompressors that are prefix-stable with respect to the first argument. This smaller class of decompressors contains an optimal decompressor (for this class). The proof of this statement is similar to the proof of Theorem 42 (page 71) where an optimal unconditional prefix-stable decompressor is constructed. We modify this proof by adding the parameter z in all formulas. More specifically, let

$$D'(\hat{p}y,z) = [p](y,z).$$

Here $[p]$ stands for the program obtained from p via "prefix stabilization for a given y". This mean that for all p,z the function $y \mapsto [p](y,z)$ is prefix-stable, and if the function $y \mapsto p(y,z)$ itself is prefix-stable then it coincides with the function $y \mapsto [p](y,z)$. It is easy to verify that this is indeed possible and that D' is an optimal prefix-stable (with respect to the first argument) decompressor.

Fix an optimal conditional prefix-stable decompressor and denote the resulting complexity by $KP(x|z)$, the *prefix complexity of x with condition z*.

If we consider *prefix-free* decompressors (instead of prefix-stable ones) we obtain an alternative definition of conditional prefix complexity. The existence of an optimal function in this class of decompressors is proved in a similar way. The resulting complexity could be denoted by $KP'(x|z)$. Like their unconditional versions, functions $KP(x|z)$ and $KP'(x|z)$ differ by at most an additive constant, which does not depend on x and z:

$$KP'(x|z) = KP(x|z) + O(1).$$

As in the case of unconditional complexities, this is proved using the conditional a priori probability $m(x|z)$. It can be defined in two ways (using probabilistic machines and lower semicomputable semimeasures).

Let M be a probabilistic machine with an input. Let $p_M(x|z)$ denote the probability that M outputs the string x for input z. The function $\langle x,z\rangle \mapsto p_M(x|z)$ is lower semicomputable and for all z the sum $\sum_x p_M(x|z)$ does not exceed 1. Conversely, for every lower semicomputable function

$\langle x,z\rangle \mapsto p(x|z)$ that takes non-negative real values such that $\sum_x p(x|z) \leqslant 1$ for all z, there exists a probabilistic machine M with $p_M = p$.

The class of all functions p_M has an optimal function, that is, the greatest one up to a constant factor. Fixing an optimal function in this class, we obtain the *conditional a priori probability* $m(x|z)$ *of the string* x *with condition* z.

The inequality $KP(x|z) \leqslant KP'(x|z) + O(1)$ is easy (as in the unconditional case). To show that all three complexities $KP(x|z)$, $KP'(x|z)$ and $-\log m(x|z)$ coincide up to an additive constant, one has to prove the inequalities $-\log m(x|z) \leqslant KP(x|z) + O(1)$ and $KP'(x|z) \leqslant -\log m(x|z) + O(1)$. We omit those proofs since they repeat their unconditional versions.

One could say that these inequalities and their proofs are "relativizations" of the respective unconditional inequalities and proofs. The relativization is understood here in a non-standard way. In the Theory of Computation, relativization means that the class of computable functions is replaced by the class of *A-computable* functions, i.e., the class of functions computable with a given oracle A. (Here A is an arbitrary set of binary strings. A function is computable with oracle A if it is computed by an algorithm that is allowed to make queries of the form "$x \in A$?". That is, the algorithm calls an external procedure that on input x returns `true` or `false` depending on whether x is in A or not.) Almost all known theorems in the Theory of Computation generalize to A-computable functions.

By the way, the notion of Kolmogorov complexity can be relativized in a standard way, too. That is, for every set A we can define the plain Kolmogorov complexity $KS^A(x)$ and the prefix Kolmogorov complexity $KP^A(x)$ (see Section 6.4). However, we do not consider relativized Kolmogorov complexity now. Instead of algorithms having an oracle access to a set of strings we consider algorithms having an access to a finite string z. In this way we obtain conditional complexity $KS(x|z)$ or $KP(x|z)$ instead of $KS(x)$ (resp. $KP(x)$). Since z is finite, the access to it does not increase the power of algorithms (any z-computable function is computable without z). However, the access to z changes Kolmogorov complexity, if z contains non-negligible information. Here is another example of this kind of relativization: the quantity $I(x:y|z)$ can be considered as common information in x and y relative to z.

Important remark. Up to now the structure (prefix relation) used in the definition of prefix-stable and prefix-free functions is applied to descriptions only. The described objects, as well as conditions, can have no structure at all.

The other approach is also possible: we could take into consideration the binary relation "to be a prefix of" on described objects as well. This will lead us to monotone complexity (see Chapter 5) and decision complexity (Chapter 6). On the other hand, we could consider he relation "to be a prefix of" on conditions as well (see Section 6.3). The resulting complexities make sense, however, they are not well studied yet.

Note that all the requirements in the definitions of prefix-free and prefix-stable decompressors treat different conditions separately. For example, requiring that a machine can tell when the input ends, we allow this decision depend on the condition. This explains why the statement of Problem 23 (p. 34) is not true for prefix complexity:

79 Show that $KP(y|x)$ does not exceed the minimal prefix complexity of a program mapping x to y (up to an additive constant error term). The converse statement is false. (Both statements hold {prefix-cor

for every programming language, the additive constant depends on the chosen [Hint. It is easy to see that $KP\,(y|l(y)) \leqslant l(y) + O(1)$. Indeed, every string y is its own self-delimiting description when $l(y)$ is known. If the converse inequality were true, there would be 2^n different programs of prefix complexity at most n.]

4.7.2 Properties of conditional prefix complexity

Let us mention several simple results about conditional prefix complexity.

- $KP\,(x|z) \leqslant KP\,(x) + O(1)$.

  Indeed, any prefix-stable (or prefix-free) unconditional decompressor $y \mapsto D(y)$ can be considered as prefix-stable (resp. prefix-free) conditional decompressor $\langle y,z\rangle \mapsto D(y)$ that just ignores the second argument z.

  Using semimeasures: any probabistic machine without input can be considered as a machine that has input but ignores it. And any lower semicomputable semimeasure $q(x)$ can be treated as a family $q'(x|z) = q(x)$ indexed by z.

- $KP\,(x|x) = O(1)$.

  Indeed, the decompressor $D(y,z) = z$ is prefix-stable (recall that prefix-stability requirements deals with y, not z) and $KP_D(x|x) = 0$. We can also change it to get a prefix-free decompressor: let $D(\Lambda,z) = z$ where Λ is an empty string and let $D(y.z)$ be undefined if $y \neq \Lambda$. Finally, the family of semimeasures can be constucted as follows: $q(x|x) = 1$ and $q(x|z) = 0$ for $z \neq x$.

- $KP\,(f(x,z)|z) \leqslant KP\,(x|z) + O(1)$ for any computable function f and for any strings x,z such that $f(x,z)$ is defined. (The constant in $O(1)$ may depend on f but not on x and z.)

  Indeed. let D be the optimal prefix-stable [prefix-free] conditional decompressor. The mapping D': $\langle y,z\rangle \mapsto f(D(y,z),z)$ is also a prefix-stable [resp. prefix-free] decompressor and $KP_{D'}(f(x,z)|z) \leqslant KP_D(x|z)$.

  In terms of semimeasures the same argument goes as follows: let $m(x|z)$ be the a priori probability of x with condition z; consider the semimeasure

  $$q(x|z) = \sum\{m(x'|z) \mid f(x',z) = x\}$$

  (for each z this is an image of the semimeasure $x \mapsto m(x,z)$ under the mapping $x \mapsto f(x,z)$); it is easy to check that q is lower semicomputable, that $\sum_x q(x|z) \leqslant 1$ and $q(f(x,z)|z) \geqslant m(x|z)$. Since m is optimal, we get the desired inequality for a priori probabilities and their logarithms).

- $KP\,(x|z) \leqslant KP\,(x|f(z)) + O(1)$ for any computable function f and for any x,z if $f(z)$ is defined (the constant in $O(1)$ may depend on f but not on x and z).

  (Indeed, consider the decompressor $\langle y,z\rangle \mapsto D(y,f(z))$ or the conditional semimeasure $q(x|z) = m(x|f(z))$.)

- $KP\,(f(x)|x) = O(1)$ for any computable f and for all x such that $f(x)$ is defined.

  (A simple corollary.)

- $KS\,(x|z) \leqslant KP\,(x|z) + O(1)$

  Indeed, prefix-stable and prefix-free decompressors for a subclass in the class of all decompressors used in the definition of $KS\,(x|z)$.

- $KP\,(x|z) \leqslant KS\,(x|z) + 2\log KS\,(x|z) + O(1)$

  This is a corollary of previous statements. Indeed, let D be the optimal conditional decompressor (not necessarily prefix-stable or prefix-free). Then

  $$KP\,(D(y,z)|z) \leqslant KP\,(y|z) + O(1) \leqslant KP\,(y) + O(1) \leqslant l(y) + 2\log l(y) + O(1).$$

  If y is the shortest description of x with condition z, then $l(y) = KS\,(x|z)$.

  In the same way one can prove a stronger inequality

  $$KP\,(x|z) \leqslant KS\,(x|z) + \log KS\,(x|z) + 2\log\log KS\,(x|z) + O(1)$$

  etc.

4.7.3 Prefix complexity of a pair

As we have seen (Theorem 54, p. 86), $KP\,(x,y)$ does not exceed $KP\,(x) + KP\,(y) + O(1)$. Let us prove a stronger inequality:

{prefix-pai

Theorem 60

$$KP\,(x,y) \leqslant KP\,(x) + KP\,(y|x) + O(1).$$

◁ We can use either prefix-free decompressors or semimeasures. Both versions are instructive.

Using prefix-free decompressors: Let D be the optimal unconditional prefix-free decompressor. Let D_c be the optimal conditional prefix-free decompressor. Consider the function D' defined as follows:

$$D'(uv) = [D(u), D_c(v, D(u))]$$

(for u and v such that the right hand side is defined). Following the proof of Theorem 54, we note that D' is well defined and is an prefix-free (unconditional) decompressor. The concatenation of the shortest D-description for x and the shortest D_c-description for y (with condition x) is a description for $[x,y]$.

(Note that the order of u and v is crucial for this argument: replacing uv by vu we get into a trouble: to find where v ends, we have to use the prefix-free property of D_c, but it is valid only for a fixed condition and $D(u)$ is not determined yet.)

Using semimeasures: Let $m(x)$ be the unconditional a priori probability of x and let $m(y|x)$ be the conditional a priori probability of y when x is known. Consider the function m' defined as follows:

$$m'([x,y]) = m(x)m(y|x)$$

(we assume that $m'(z) = 0$ for strings z that are not encodings of any pairs). Then m' is lower semicomputable (being a product of two non-negative lower semicomputable functions), and

$$\sum_z m'(z) = \sum_{x,y} m(x)m(y|x) = \sum_x \bigl[m(x)\sum_y m(y|x)\bigr] \leqslant \sum_x m(x) \leqslant 1.$$

Therefore, $m([x,y]) \geqslant \varepsilon m'([x,y]) = \varepsilon m(x)m(y|x)$. $\triangleright$

80 Prove that $KS\,(x,y) \leqslant KP\,(x) + KS\,(y|x) + O(1)$.

[Hint: One may use prefix-free decompressor and append the (plain) description of y to the prefix-free description of x. The other argument: count the number of pairs such that $KP\,(x) + KS\,(y|x) \leqslant n$. We have at most $2^k \cdot m(k) \cdot 2^{n-k} = 2^n m(k)$ pairs such that $KP\,(x) = k$, and the sum over k gives $2^n \cdot O(1)$.]

Further improvements are possible. First note that we can use pairs of strings as conditions by using some computable injective encoding (changing the encoding we change the complexity at most by a constant). For similar reasons we can speak about complexity of a triple of strings. Now we can write the following chain of inequalities (the $O(1)$ terms are omitted):

$$KP\,(x,y) \leqslant KP\,(x,KP(x),y) \leqslant KP\,(x,KP\,(x)) + KP\,(y|x,KP\,(x)) = KP\,(x) + KP\,(y|x,KP\,(x)).$$

Here the equality $KP\,(x,KP\,(x)) = KP\,(x)$ (Theorem 55) is used as well as the inequality for the entropy of pairs (Theorem 60). We get an inequality that can be considered as a strong form of Theorem 60, since $KP\,(y|x,KP\,(x)) \leqslant KP\,(y|x)$ (because x can be produced from $[x, f(x)]$ by an algorithm). As L.A. Levin (see [**?**]) and also G. Chaitin (see [**?**]) have noticed, this refined inequality is (remarkably) an equality:

{prefix-pai

Theorem 61

$$KP\,(x,y) = KP\,(x) + KP\,(y|x,KS\,)) + O(1).$$

$\triangleleft$ In one direction the inequality is already known (see the discussion above). One can give also a direct argument: to get a prefix-free description of a pair $\langle x,y\rangle$, it is enough to start with prefix-free description of x and then append the prefix-free description of y with conditions x and $KP\,(x)$ (note that $KP\,(x)$ is just the length of the prefix-free description of x). After the machine reads the first part and stops, we know both x (its output) and $KP\,(x)$ (the length of the input), so we have all needed information to restore y (in a self-delimiting way).

Using semimeasures, we can prove the same inequality as follows. Consider a function m' such that

$$m'([x,y]) = \sum_{\{k|2^{-k}<2m(x)\}} 2^{-k} m(y|x,k)$$

This function is below semicomputable and its sum over all x, y is finite (for each x and k the sum over all y does not exceed 1, then the sum over all k such that $2^{-k} < 2m(x)$ does not exceed $4m(x)$, and the sum over x does not exceed 4). So we compare m' with the a priori probability and conclude that for $k = -\lfloor \log_2 m(x) \rfloor$ we get the term that we want to estimate.

Now let us consider the reversed inequality:

$$KP\,(x) + KP\,(y|x,KP\,(x)) \leqslant KP\,(x,y) + O(1).$$

Let us start with a simple (but incorrect!) proof of a stronger (but incorrect!) statemenet

$$KP(x) + KP(y|x) \leqslant KP(x,y) + O(1).$$

In terms of semimeasures this equality can be rewritten as follows:

$$m(x)m(y|x) \geqslant \varepsilon m([x,y])$$

(for some ε and for all x,y). Here m stands for a priori probabilities (both conditional and unconditional ones). Let us rewrite this inequality as

$$m(y|x) \geqslant \varepsilon \frac{m([x,y])}{m(x)}.$$

It is enough to show that the function

$$m'(y|x) = \varepsilon \frac{m([x,y])}{m(x)}$$

for any fixed x is a semimeasure (for some ε); after that we can compare it with the maximal semimeasure $m(y|x)$ and get the desired result. We need to show that the sum of $m'(y|x)$ over y does not exceed 1:

$$\sum_y m'(y|x) = \varepsilon \frac{\sum_y m([x,y])}{m(x)} < 1.$$

Indeed, the function $x \mapsto \sum_y m([x,y])$ is a semimeasure (its sum over all x equals $\sum_{x,y} m([x,y]) \leqslant 1$) and therefore this function is bounded by $m(x)/\varepsilon$ for some ε.

What is wrong with this argument? We have not checked that the semimeasure we constructed is lower semicomputable. There are two cases where we need to check this. In one of them it it is easy: the function $\sum_y m([x,y])$ is lower semicomputable since m is lower semicomputable. But in the other case, for function $m([x,y])/m(x)$, the lower semicomputable function $m(x)$ is in the denominator, and when $m(x)$ increases, the fraction decreases.

The correct proof of the weaker ineqality follows the same scheme but uses some additional tricks. We have to prove that for $z = KP(x)$ the inequality

$$m(y|x,z) \geqslant \varepsilon \frac{m([x,y])}{m(x)}$$

holds. The problem is that the right hand side is not lower semicomputable. But for $z = KP(x)$ we can replace $m(x) \approx 2^{-KP(x)}$ by 2^{-z} and consider the function

$$m'(y|x,z) = m([x,y])2^z.$$

This function in lower semicomputable. But now it is not a semimeasure: the sum $\sum_y m'(y|x,z)$ is bounded by 1 only if

$$\sum_y m([x,y]) \leqslant 2^{-z}$$

which is not true if z is large. However, we know that $\sum_y m([x,y]) = O(m(x)) = O(2^{-KP(x)})$, so there exists a constant c such that

$$z \leqslant KP(x) - c \Rightarrow \sum_y m'(y|x,z) \leqslant 1.$$

But this is not enough: we need a family of semimeasures that satisfy this inequality for all x and z. So we "trim" the function m' and get another function m'' such that:

- function $\langle y,x,z\rangle \mapsto m''(y|x,z)$ is lower semicomputable;
- the inequality
$$\sum_y m''(y|x,z) \leqslant 1$$
is true for all x and z;
- the exists a constant c such that
$$z \leqslant KP(x) - c \Rightarrow m''(y|x,z) = m'(y|x,z).$$

How to perform "trimming"? This trick was explained in Section 4.2: we look at the increasing approximations from below and let them through only if the do not violate the required bound for the sum. , .

Now, comparing m'' with the a priori probability and taking the logarithms, we conclude that

$$z \leqslant KP(x) - c \Rightarrow KP(y|x,z) \leqslant KP(x,y) - z + c'$$

for some c, c' and for all x, y, z.

Now we let z be equal to $z = KP(x) - c$. Note also that changing z by 1 changes the value $KP(y|x,z)$ by at most $O(1)$ (increasing/decreasing the second component of a pair is a computable function). Therefore, $KP(y|x, KP(x) - c) = KP(y|x, KP(x)) + O(1)$. ▷

Note that we get Theorem 22 (p. 37), which says that $KS(x,y) = KS(x) + KS(y|x) + O(\log n)$ for strings of complexity at most n, as a corollary.

Indeed, the replacement of KP by KS changes all three terms by at most $O(\log n)$. It remains to note that the difference between $KS(y|x, KP(x))$ and $KS(y|x)$ is bounded by $O(\log n)$. In this way we get a new proof of Theorem 21 that replaces counting by manipulations with semimeasures.

Recalling that $m(x) \approx \sum_y m([x,y])$ (up to $O(1)$ factor, Problem 73, p. 87), we may rewrite the statement of Theorem 61 as follows:

$$m(y|x, KP(x)) \approx \frac{m([x,y])}{\sum_y m([x,y])}$$

The right hand side of the equation can be interpreted as the conditional probability of the event "the second component of the pair equals y" where condition is "the first component of the pair equals x".

81 Prove that

$$KP(x|z) \leqslant KP(x|y) + KP(y|z) + O(1)$$

for any strings x, y, z. (This result can be improved if we replace $KP(x|y)$ by a smaller term $KP(x|y,z)$.)

82 Prove the "relativized" version of Theorem 61:

$$KP(x,y|z) = KP(x|z) + KP(y|x, KP(x|z), z) + O(1).$$

Using Theorem 61 twice, we a get a formula for the prefix complrxity of a triple. Indeed, the triple $\langle x,y,z\rangle$ can be considered as a pair whose first component is $\langle x,y\rangle$ and the second component is z. Therefore,

$$KP(x,y,z) = KP(z|x,y,KP(x,y)) + KP(x,y) + O(1).$$

Using Theorem 61 once again, we get the following result:

{prefix-tri

Theorem 62

$$KP(x,y,z) = KP(z|x,y,KP(x,y)) + KP(y|x,KP(x)) + KP(x) + O(1).$$

We can change the order of transformations (using the z-relativized version of Theorem 61) at the second step:

$$KP(x,y,z) = KP(y,z|x,KP(x)) + KP(x) = \\ = KP(z|y,KP(y|x,KP(x)),x,KP(x)) + KP(y|x,KP(x)) + KP(x)$$

(we omit the $O(1)$-terms for brevity).

It is interesting that this leads to a slightly different version of Theorem 62: the two last terms are the same but the first term is different. We still have the conditional complexity of z but now we have two conditions $KP(x)$ and $KP(y|x,KP(x))$ instead of $KP(x,y)$. Note that the sum of the complexities in the condition is exactly $KP(x,y)$ according to Theorem 61. Therefore, the pair of complexities has no less information than $KP(x,y)$. In fact the reverse is also true (when x and y are conditions). Indeed, let z be the pair $\langle KP(x), KP(y|x,KP(x))\rangle$; in the second formula the first term is zero (i.e., $O(1)$). So we get the following corollary:

{prefix-pai

Theorem 63

$$KP(KP(x)|x,y,KP(x,y)) = O(1),$$
$$KP(KP(y|x,KP(x))|x,y,KP(x,y)) = O(1).$$

(Of course the same is true for $KP(y)$ and $KP(x|y,KP(y))$.)

83 Give a direct proof of Theorem 63. [Hint: Knowing x, y $KP(x,y)$, we may look for an upper bound d for $KP(x)$ such that $KP(y|x,d)+d$ becomes equal to $KP(x,y)$. The coincidence

(up $O(1)$) implies that $d = KP(x) + O(1)$: indeed, if $d = KP(x) + m$ for some m, the complexity $KP(y|x,d)$ can decrease (because of this m) at most by $O(\log m)$, and the sum becomes bigger.]

Using Theorem 61 we can easily show that the basic inequality of Theore 24 (p. 42) is true with $O(1)$-precision for prefix complexity (recall that we have logarithmic error term for plain complexity):

{prefix-bas

Theorem 64

$$KP(x,y,z) + KP(x) \leqslant KP(x,y) + KP(x,z) + O(1)$$

for every three strings x,y,z.

$\lhd$ Indeed, the right hand side can be rewritten as

$$KP(x) + KP(y|x,KP(x)) + KP(x) + KP(z|x,KP(x)),$$

and the levt hand side equals

$$KP(x) + KP(y,z|x,KP(x)) + KP(x).$$

It remains to prove that

$$KP(y,z|x,KP(x)) \leqslant KP(y|x,KP(x)) + KP(z|x,KP(x)),$$

and this inequality is a relativized version of Theorem 54 (p. 86). $\rhd$

Let us provide also a direct proof of Theorem 64 using semimeasures. We have to show that (up to $O(1)$-factors)

$$m(x,y,z)m(x) \geqslant m(x,y)m(x,z),$$

where m is the maximal lower semicomputable semimeasure. Dividing by $m(x)$, we get an inequality

$$\frac{m(x,y)m(x,z)}{m(x)} \leqslant m(x,y,z).$$

Let us check that the left hand side of this inequality has a finite sum (over all triples x,y,z. Indeed,

$$\sum_{y,z} \frac{m(x,y)m(x,z)}{m(x)} \leqslant m(x)$$

(since $\sum_y m(x,y) \leqslant m(x)$ and $\sum_z m(x,z) \leqslant m(x)$). (We omit $O(1)$ factors for brevity.)

This is not enough: since we have $m(x)$ in the denominator, the fraction

$$\frac{m(x,y)m(x,z)}{m(x)}$$

is not (necessarily) lower semicomputable and we cannot use the maximality property. So we need to use the following trick (similar to the trick used in the proof of Theorem 61) to construct a lower semicomputable upper bound for this fraction.

For each n consider the function $m_n(x,y)$ which is obtained from $m(x,y)$ by 2^{-n}-trimming: the sum $\sum_y m(x,y)$ is forced to be at most 2^{-n}. Note that $\sum_y m(x,y) = m(x)$ (up to $O(1)$-factors) and therefore $m_n(x,y) = m(x,y)$ for $n = KP(x)$. Then we consider the function

$$\langle x,y,z\rangle \mapsto \sum_{n\geqslant KP(x)} \frac{m_n(x,y)m_n(x,z)}{2^{-n}}$$

It is an upper bound since it contains the term with $n = KP(x)$. On the other hand,

$$\sum_{x,y,z}\sum_{n\geqslant KP(x)} \frac{m_n(x,y)m_n(x,z)}{2^{-n}} \leqslant \sum_x \sum_{n\geqslant KP(x)} \frac{\sum_y m_n(x,y)\sum_z m_n(x,z)}{2^{-n}}$$
$$\leqslant \sum_x \sum_{n\geqslant KP(x)} 2^{-n} \leqslant \sum_x 2m(x) \leqslant 2.$$

(As before, we omit $O(1)$-factors which lead only to $O(1)$-factor in the final inequality.)

84 Show that the inequality of Theorem 26 (p. 44) is true for prefix complexity with $O(1)$-precision: {condit-tri

$$2KP(x,y,z) \leqslant KP(x,y) + KP(x,z) + KP(y,z) + O(1)$$

for all strings x,y,z. [Hint: add the basic inequality $KP(x,y,z) + KP(z) \leqslant KP(x,z) + KP(y,z)$ to the inequality $KP(x,y,z) \leqslant KP(x,y) + KP(z)$.]

85 Prove that there exists c such that for every string x and for every positive integer n there exists a string y of length n such that {increasing

$$KP(x,y) \geqslant KP(x) + n - c$$

[Hint: for every z and n there exists a string y of length n such that $KP(y|z) \geqslant n$.]

A similar statement can be formulated for n-bit extensions of a given string x (its version for plain complexity makes Problem 34 on p. 38)

{increasing

Theorem 65

$$\max\{KP(xy)|l(y) = n\} \geqslant KP(x|n) + n - O(1).$$

In other terms, for some c and all x and n we can append n bits to x in such a way that its complexity is at least n bits more that $KP(x|n)$ (this is not exactly the increase in the complexity since we compare $KP(xy)$ with $KP(x|n)$ and not $KP(x)$).

$\lhd$ In terms of a priori probabilities this inequality says that

$$2^n \min\{m(xy)|l(y) = n\} \leqslant m(x|n)\cdot O(1)$$

The left hand side does not exceed $\sum\{m(xy)|l(y) = n\}$ (the sum may only decrease is we replace all summands by the least one). But the latter sum is (as a function of x and n) a lower semicomputable semimeasure, so it remains to compare it with the maximal semimeasure $m(x|n)$. $\rhd$

86 Show that a bit weaker statement with$KP(x) - KP(n)$ instead of $KP(x|n)$ (in the right hand side) can be derived from the statement of Problem 85.

5 Monotone complexity

{monot}

5.1 Probabilistic machines and semimeasures on the tree

{monotsm}

Chapter 4 defines a priori probability by using probabilistic algorithms (machines) that may print some number as their output and then terminate. In this chapter we consider another type of probabilistic (=randomized) algorithms. These algorithms output a binary sequence bit by bit and do not necessarily terminate. The output, therefore, is a random variable whose values are finite and infinite sequences of bits (i.e., elements of the set Σ of all finite and infinite sequences of bits).

Consider the following simple algorithm of this type. It just sends random bits directly to the output:

while true do
  b:=*random*;
  OutputBit(*b*);
od

Its output therefore is a random variable that is uniformly distributed over Ω, the set of all infinite binary sequences.

But it is quite possible (for another algorithm) that some finite sequence is printed with positive probability. This happens when algorithm with positive probability stops after sending some bits to the output (or runs forever without sending any bits to the output).

For each algorithm A of the described type we consider a function a that is defined on binary strings and whose values are non-negative reals:

$$a(x) = \Pr[\text{the output of } A \text{ starts with } x]$$

More formally this function is defined in the following way. Each probabilistic algorithm defines a mapping $\bar{A}$ of the set Ω (infinite sequences of zeros and ones) into the set Σ. Namely, $\bar{A}(\omega)$ is a sequence of output bits that appears if we use the terms of the sequence ω as random bits (this means that each statement *b := random* assigns to b the first unused bit of ω). For example, if A is the program mentioned above, then $\bar{A}(\omega) = \omega$ for all ω.

Then $a(x)$ is defined as the measure of the preimage of the set Σ_x under the mapping $\bar{A}$ (where Σ_x is the set of all finite and infinite sequences having prefix x). We say that A *generates* the distribution a.

87 What are $\bar{A}$ and a, if the algorithm A prints an infinite sequence of zeros (not using random bits at all)?

A natural question arises: what is the class of all functions a that correspond to randomized algorithms A of the described type? Here is the answer.

{monotsm-c1

Theorem 66 *Let A be a randomized algorithm of the described type and let a be the corresponding function. Then:*

(**a**) $a(x) \geqslant 0$ *for all x;*

(**b**) $a(\Lambda) = 1$ (*here* Λ *is the empty string*)*;*

(**c**) $a(x) \geqslant a(x0) + a(x1)$ *for every string x;*

(**d**) *the function a is lower semicomputable.*

The notion of the lower semicomputable (enumerable from below) sequence of reals was defined in Section 4.1 (p. 65). For the functions on strings the definition is quite similar: we require that $a(x) = \lim_i a(x,i)$ where a is a computable function, $a(x,i)$ is defined for all strings x and for all non-negative integers i, has rational values (special symbol $-\infty$ is allowed) and non-decreases as i increases.

$\lhd$ The first three claims are obvious:

(a) Probability is always non-negative.

(b) $a(\Lambda) = 1$ since the empty string is a prefix of any output.

(c) $a(x) \geqslant a(x0) + a(x1)$, since the events "the output starts with $x0$" and "the output starts with $x1$" are inconsistent and both are subsets of the event "the output starts with x".

Note that the inequality (c) can be strict; the difference

$$a(x) - a(x0) - a(x1)$$

is the probability of the event "the output is exactly the string x" (no bits appear after it).

(d) To prove that a is lower semicomputable, we need to construct approximations from below for $a(x)$ for any given string x. Let us simulate the behavior of A for all possible values of random bits. During this simulation we discover values of random bits that guarantee that output starts with x, i.e., we find some intervals I in Ω such that $\overline{A}(\omega)$ starts with x for all $\omega \in I$. The probability $a(x)$ is the measure of the union of all these intervals, and the approximation $a(x,i)$ is the measure of the union of all the intervals discovered up to the step i of the simulation. $\rhd$

The function a that is defined on all binary strings, takes real values and satisfies the conditions (a)–(d) of Theorem 66 is called an *lower semicomputable semimeasure on the binary tree*. It is important not to mix semimeasures on the binary tree and semimeasures defined in Chapter 4 that were functions on natural numbers (or on binary strings that correspond to natural numbers) and correspond to probabilistic algorithms that print some number (or string) and terminate.

All functions that satisfy conditions (a)–(c) are called *semimeasures on the binary tree*, or *tree semimeasures*; the condition (d) additionally requires that a tree semimeasure is lower semicomputable.

88 Show that tree semimeasures are in a one to one correspondence with measures on the set Σ of all finite and infinite binary sequences. Given a semimeasure a, find the measure of the set that consists of all infinite sequences that have prefix x. [Answer: the measure of this set equals the limit of the (decreasing) sequence

$$\alpha_n = \sum\{a(y) | y \text{ is a string of length } n \text{ that has prefix } x\}$$

Here α_n is defined for $n \geqslant l(x)$ and equals $a(x)$ if $n = l(x)$.]

89 Show that for a semicomputable tree semimeasure the sum $\sum_x a(x)$ can be infinite. [Hint: Consider the algorithm that copies random bits to output.]

The converse of Theorem 66 is also true:

{monotsm-cr

Theorem 67 *Every lower semicomputable tree semimeasure corresponds to some probabilistic algorithm.*

$\lhd$ The idea of the proof can be easily explained in terms of space allocation, as it was done for Theorem 40 (p. 68). The difference is that now the requests are hierarchical. Two big organizations (called 0 and 1) need space in Ω (which we identify with $[0,1]$); the subsets allocated for 0 and 1 should be disjoint, and their space requests increase over time (but never become greater than 1 in total).

Each of the organizations has two divisions (called $00,01$ inside 0 and $10,11$ inside 1 that request some space inside the regions allocated to their organization as a whole. Their requests also increase over time, but never become greater (in total) than the organization's request (at the same time). Then we consider subdivisions (say, 01 has subdivision 010 and 011) that have increasing requests that do not go out of the request of their parent division, and so on.

For each subdivision x (at any level) we have increasing requests. All the allocations are final, i.e., the space allocated to some x remains allocated to x.

This scheme is used in the proof as follows: having a lower semicomputable semimeasure a, we construct a family of requests such that the limit of the requests for subdivision x is equal to $a(x)$. Then we choose a way to satisfy all the requests and then say that if a sequence of random bits gets into the region allocated to x, then the output of randomized algorithms starts with x.

It is more or less obvious that the requests can indeed be fulfilled. However, we provide a more formal argument (and explain the intuitive meaning of its steps).

Lemma 1. Let a be a lower semicomputable semimeasure on the binary tree. Then there exists a total computable monotone (in the second argument) function $\langle x,i\rangle \mapsto a(x,i)$ whose values are non-negative rational numbers with denominators being powers of two and:

(1) $\lim_i a(x,i) = a(x)$ for every string x;

(2) for each i the function $x \mapsto a(x,i)$ is a semimeasure that has only finitely many non-zero values.

In other terms, the memory manager can impose the following additional restrictions:

- all the requests should be rational numbers whose denominators are powers of two;
- at each step only finitely many subdivisions can have nonzero requests;
- at each step requests are coherent (the request of any subdivision should be greater than or equal to the sum of requests of its children).

Proof of the Lemma. Or goal is to change the function a from the definition of lower semicomputable semimeasure (not changing the semimeasure itself) so that it satisfies the requirements of the Lemma. First, we make all values rational numbers whose denominators are powers of two. To achieve this, we replace $a(x,i)$ by the closest rational number with denominator 2^i not exceeding $a(x,i)$ (negative numbers are replaced by zeros).

Then we fulfill the second requirement and let $a(x,i)$ be zeros for all strings x whose length exceeds i.

To fulfill the third requirement, we perform the replacement

$$a(x,i) := \max\left(a(x,i), a(x0,i) + a(x1,i)\right)$$

iteratively starting from long strings and then decreasing the length of x. Since $a(x)$ is by definition a semimeasure, these replacements do not violate the inequality $a(x,i) \leqslant a(x)$.

It is easy to check that our corrections do not change the limit values $\lim_i a(x,i)$ (for all x), so this limit is still equal to $a(x)$.

Lemma 1 is proved.

To formulate the next lemma we need several auxiliary definitions. A "simple semimeasure" (on the binary tree) is a semimeasure that has only finitely many nonzero values and all these values are rational numbers whose denominators are powers of two.

A "simple set" is the union of a finite number of intervals in Ω. (Recall that an interval in Ω is a set of the form Ω_z that consists of all infinite sequences having prefix z. Therefore, a set is simple if we need to know only a finite prefix of ω to decide whether ω belongs to this set.)

A "simple family" is a family of simple sets A_x (for some binary strings x) such that only finitely many sets among A_x are non-empty and for each string x the sets A_{x0} and A_{x1} are disjoint subsets of A_x.

For such a family the function $x \mapsto \mu(A_x)$, where μ stands for the uniform measure on Ω is a simple semimeasure. We say that the family A_x "implements" this semimeasure.

Lemma 2. Each simple semimeasure can be implemented by a simple family.

Proof. We construct this family starting from the empty string x and then gradually increasing the length of the index string x. At each step our goal is to find two disjoint simple sets A_{x0} and A_{x1} inside the set A_x that is already constructed. This is possible since the required measures do not exceed (in total) the measure of A_x. Lemma 2 is proved.

Lemma 3. Let $b(x)$ be a simple semimeasure and let B_x be a simple family of intervals that implements b. Let c be another simple semimeasure such that $c(x) \geqslant b(x)$ for all x. Then we can construct a simple family C_x implementing c such that $C_x \supset B_x$ for all x.

Proof. Let us repeat the argument used to prove Lemma 2. Now we have two disjoint simple subsets of a simple set and need to increase their measures (keeping them disjoint). It is easy to see that this is indeed possible if the space restrictions are not violated. Lemma 3 is proved.

The proofs of Lemma 2 and Lemma 3 are effective in the natural sense: given the tables of values for simple semimeasures, we can algorithmically construct the simple family required.

Now we apply Lemma 3 iteratively to the simple semimeasures that are obtained by Lemma 1. In this way we get a two-parametric family of simple sets $U(x,i)$ such that

- the description of $U(x,i)$ (i.e., the list of intervals) is a computable function of x and i;
- the uniform measure of the set $U(x,i)$ is equal to $a(x,i)$ (and therefore tends to $a(x)$ as $i \to \infty$);
- for each x and i the sets $U(x0,i)$ and $U(x1,i)$ are disjoint subsets of the set $U(x,i)$;
- $U(x,i) \subset U(x,i+1)$ for each x and i.

Now the probabilistic algorithm that generates the semimeasure a can be constructed as follows: we construct the sets $U(x,i)$ for all x and i and in parallel generate random bits obtaining a sequence ω. If at some step we discover that $\omega \in U(x,i)$ for some x and i, we output those bits of the string x that have not yet been printed.

Note that if $\omega \in U(x,i)$ then $\omega \in U(y,i)$ for every prefix y of x. Note also that ω cannot be an element of both $U(x,i)$ and $U(x',i)$ if strings x and x' are inconsistent (neither of them is the prefix of the other one). Therefore the bits sent to the output never need to be "recalled".

An output of this algorithm starts with some string x if and only if the sequence ω of random bits belongs to the union of the increasing sequences of sets $U(x,i)$ (for $i = 0,1,2,\ldots$). The probability of this event is the limit of measures of the sets $U(x,i)$, and this limit is by construction equal to $a(x)$, so we have achieved our goal. $\triangleright$

Theorems 66 and 67 show that lower semicomputable semimeasures can be equivalently defined as probability distributions generated by randomized algorithms (of the described class).

There is an important special case when a randomized algorithm almost surely generates an infinite sequence (i.e., the probability to get a finite sequence is zero). Such algorithms generate computable measures, as the following theorem shows.

{monotsm-se

Theorem 68 (**a**) *Let μ be a computable measure on Ω. Then function p defined as $p(x) = \mu(\Omega_x)$ is a lower semicomputable semimeasure and $p(x) = p(x0) + p(x1)$ for all x.*

(**b**) *If a lower semicomputable semimeasure p satisfies the equality $p(x) = p(x0) + p(x1)$ for all x, then it determines some computable measure on Ω.*

$\triangleleft$ (a) If a real number α is computable and a_n is a rational approximation to α with accuracy $1/n$, then $b_n = a_n - 1/n$ is a lower bound for α that is at most $2/n$ apart from α. The sequence b_n constructed in this way can violate the monotonicity requirement but we may replace it by the sequence

$$c_n = \max(b_0, b_1, \ldots, b_n)$$

that is a non-decreasing sequence of rational numbers tending to α. Therefore, every computable real number is lower semicomputable. Doing this in parallel for all x, we obtain computable rational lower bounds for $p(x)$ tending to $p(x)$ and prove that every computable measure is an lower semicomputable semimeasure. Since Ω_x is the union of two disjoint subsets Ω_{x0} and Ω_{x1}, we also have $p(x) = p(x0) + p(x1)$.

(b) Let p be a lower semicomputable semimeasure such that $p(x) = p(x0) + p(x1)$ for all x. We show inductively how $p(x)$ can be found up to any precision for every x. For empty x we have $p(\Lambda) = 1$ by definition. Imagine that we already know how to find $p(x)$ with arbitrary precision for some string x. How can we do the same for $p(x0)$ and $p(x1)$? We have to wait until the sum of (increasing) lower bounds for $p(x0)$ and $p(x1)$ become close enough to the (decreasing) upper bound for $p(x)$. In other terms, an upper bound for $p(x1)$ can be obtained if we take an upper bound for $p(x)$ (constructed recursively) and subtract a lower bound for $p(x0)$. $\triangleright$

This theorem can be interpreted in the following way. Assume that we need a generator of random reals (=sequences of zeros and ones) whose output has a prescribed distribution p (this means that the probability to get an output that starts with x is equal to $p(x)$). Then Theorems 67 and 68 guarantee that if p is a computable distribution, then such a generator can be implemented as a randomized algorithm that uses the internal source of random bits that has uniform distribution.

Note that the construction used in the proof of Theorem 67 can be simplified in the special case when we deal with computable measures (and not arbitrary semicomputable semimeasures). This simplified construction goes as follows. Let us divide the interval $[0,1]$ into two parts of lengths

$p(0)$ and $p(1)$. The first part is then divided again into parts of length $p(00)$ and $p(01)$, the second one is divided into two parts of length $p(10)$ and $p(11)$, and so on. In this way for each string z we get an interval π_z inside $[0,1]$, and the segments π_z for all strings z of any given length cover $[0,1]$ without overlaps.

Now construct the probabilistic algorithm as follows. This algorithm uses independent tosses of a fair coin to get a sequence α of random bits that has uniform distribution. This sequence is considered as a binary representation of some real in $[0,1]$; this real is also denoted by α. In parallel the probabilistic algorithms looks for binary strings z such that the real number α lies strictly inside the interval π_z (and this is guaranteed by the available information about α and the current approximations to the endpoints of π_z; these approximations are computed with increasing precision).

The strings z discovered in this way are are compatible (one being a prefix of another). The more bits of α we know, the longer z can be. These strings are prefixes of some bit sequence that is the output of our randomized algorithm.

The algorithm described can output a finite sequence. This happens if α coincides with an endpoint of some π_z. However, there are countably many endpoints, so this event has probability 0. Note also that the output of the algorithm starts with z if and only if α belongs to the (open) interval π_z, so the probabilities are correct.

More formally, we have described a transformation T of the input bit sequence α into the output bit sequence $\beta = T(\alpha)$ such that the image of uniform measure under T is the measure p.

(This trick is well known. For example, imagine that you have a fair coin and you need to simulate the coin that has probabilities $2/3$ and $1/3$. Then you generate a random real uniformly distributed in $[0,1]$ (by fair coin tossing) and compare this real number with threshold $2/3$. To simulate the second coin tossing, you divide both intervals $[0,2/3]$ and $[2/3,1]$ in the same proportion $2:1$. The algorithm described earlier does exactly this.)

To understand the relations between the classes of random sequences with respect to different distributions, we need to look more closely on T. Consider the family I_z of intervals that corresponds to the uniform measure, i.e., I_z is the interval inside $[0,1]$ that is formed by reals whose binary representation starts with z (including the endpoints, so I_z is a closed interval).

Using this notation, we describe the transformation $T : \Omega \to \Sigma$ as follows: a string y is a prefix of $T(\alpha)$ if there exists a prefix x of α such that I_x is strictly inside π_y (i.e., is a subset of the interior of π_y).

In the similar way we can define another transformation $U : \Omega \to \Sigma$ that goes in another direction: string x is a prefix of $U(\beta)$ if β has some prefix y such that π_y is strictly inside I_x.

We would like to say that transformations T and U are inverse to each other, since T converts a sequence α into a real number that has binary representation α, and then converts its back into a bit sequence using "p-representation" instead of binary representation, while U does exactly the same in the other direction. But this is not literally true for several reasons.

As we have mentioned, the rational numbers whose denominators are powers of two, have two binary representations. The similar problems appears with the endpoints of intervals π_z and p-representation. Also it may happen that some π_z has zero length, and then all the sequences that start with z correspond to the same point. On the other hand, is some infinite sequence has positive

p-measure, the entire interval on the real line corresponds to this sequence.

But if we forget these problems for a while, we can indeed think of T and U as transformations that are mutually inverse of each other and relate uniform distribution and p-distribution. This informal idea can be verbalized in many ways. As we have already seen, if α is a random element of Ω with the uniform distribution, then $T(\alpha)$ has distribution p. On the other hand, we get move in another direction:

90 Prove that if no singleton has positive p-distribution, and β is a random variable that has distribution p, then $U(\beta)$ is infinite with probability 1 and is uniformly distributed in Ω. [Hint: Each sequence β determines the sequence of decreasing closed intervals that have only one commont point; the probability that this common point is rational equals zero. The probability for $U(\beta)$ to get inside π_z is correct by definition, and any other interval is a countable union of π_z if we ignore its endpoints.]

This problems shows how to generate an uniform distribution from a non-uniform one. A toy example of this type: assume that we have a biased coin with probability $2/3$ of getting a head, and we need to simulate a fair coin tossing. In this special case the task is easy without any special theory: let both players toss a coin once; if both have the same, we have a draw and everything is repeated; if the results are different, the player who has a head wins. (This construction, unlike the one used in the proof, is valid for any probability, not only $2/3$.)

The assumption (no sequences of positive measure) is important: not any distribution can be used to simulate the uniform one. For example, if $p(000\ldots000) = 1$ for any number of zeros and p equals zero for any other z, this "random bits generator" does not provide any randomness, it just generates zeros and is completely useless. A similar situation arises if some infinite sequence has positive measure, i.e., if there exists an infinite sequence ω and a number $\delta > 0$ such that $p(x) \geqslant \delta$ for any x that is a prefix of ω. In this case the random number generator generates ω with positive probability, so we cannot simulate the uniform distribution.

However, we are more interested about relation between ML-random sequences with respect to different measures.

Theorem 69 (**a**) *If a sequence α is ML-random with respect to the uniform measure, then the sequence $\beta = T(\alpha)$ is infinite and random with respect to measure p.*

(**b**) *If the sequence β is ML-random with respect to p and is not computable, then the sequence $\alpha = U(\beta)$ is infinite, ML-random with respect to uniform measure and $T(\alpha) = \beta$.*

(Note than some of the statements of this Theorem are corollaries of Theorem 99, p. 142.)

$\lhd$ (a) A random sequence with respect to the uniform measure is not computable, therefore it cannot represent a rational number or a computable number. Since the endpoints of all π_z are computable, the sequence $\beta = T(\alpha)$ is infinite.

Assume that an algorithm is given that for any $\varepsilon > 0$ covers the sequence β by a family of intervals Ω_u. Consider the closed intervals π_u (for corresponding u's) and replace them by slightly larger open intervals. Then we get an algorithm that covers the real number with binary representation α by a family of intervals with small sum of lengths. It can be easily converted to an algorithm that covers $\alpha \in \Omega$ by intervals Ω_v that have small sum of uniform measures (an interval on the real

line is replaced by the union of disjoint intervals Ω_v). And this is not possible since α is random with respect to the uniform distribution.

(b) Here we need some additional precautions. Let us note first that if $\{\beta\}$ has positive p-measure, then β is computable. (We assume that p is a computable measure.) Indeed, in this case $p(z) > \varepsilon$ for some ε. There are finitely many (say, k) sequences whose measure is greater than ε. Let us increase ε a bit so that all these k sequences still have measure greater than ε. Then for sufficiently large n (say, starting from N) there are k strings z of length n such that $p(z)$ is greater than (increased) ε. Knowing N and (increased) ε, we can find these strings, so all k sequences are computable.

Therefore, if β is not computable, the lengths of intervals π_z for strings z being prefixes of β, tend to 0. Therefore, these (closed) intervals have unique intersection point x. It is an interior point for all those intervals π_z since β contains infinitely many zeros and one (being non-computable). Then we note that x is not a rational number (it would make β computable) and therefore the sequence $\alpha = U(\beta)$ is infinite and $T(\alpha) = \beta$.

It remains to show that α is random (with respect to the uniform distribution). Assume that there exists a family of intervals Ω_u that covers β and has small total uniform measure (in Ω). We can transform them to open intervals on the real line that cover x and have small sum of lengths. Then we consider closed intervals π_z that get inside these open intervals. For one of them the string z is a prefix of β, since x is an interior point and the lengths of π_z tend to zero. $\triangleright$

This theorem implies the following statement: if a sequence ω is random with respect to some computable measure, then ω is either computable or is Turing-equivalent to some sequence that is random with respect to the uniform measure.

A *Turing equivalence* of two sequences α and β means that α is Turing-reducible to β and vice versa. And α is Turing-reducible to β if there exists an algorithm that computes α using β as an oracle, i.e., as an external procedure that can be called and returns nth bit of β given n. In our case these reductions are provided by transformations T and U.

Sequence that are random with respect to some computable measure were called "proper" in **??** (English translation).

A natural question related to this problem: is there a sequence that is not ML-random with respect to any computable measure? or even a sequence that is not Turing-equivalent to any ML-random (with respect to the uniform measure) sequence? (Note that we can replace 'the uniform measure' by 'any computable measure'.) Here are some observations:

1. The first question has a positive answer: there exists a sequence that is not random with respect to any computable measure. It can be constructed using the notion of randomness deficiency (see Section 5.9, p. 141):

91 Prove that for every binary string x and for every computable measure P on can effectively find a string y with prefix x that has arbitrary large randomness deficiency with respect to P (randomness deficiency is defined below in Section 5.9 as the difference between $-\log P(x)$ and a priori complexity $KA(x)$). [Hint: Let us extend x in such a way that each next bit decreases the P-measure of the corresponding interval at least by factor $3/2$. This can be done effectively, so the complexity increases slowly while the measure decreases fast.] Considering all the computable measures (each should be treated infinitely many times), show that there exists a sequence that is

not random with respect to any computable measure.

Essentially the same argument can be explained using "generic" sequences. Recall that a subset A of Ω is *everywhere dense* if it has non-empty intersection with every interval. A famous *Baire theorem* says that the intersections of a countable family of open sets A_i (an open set is a union of intervals) that are everywhere dense is nonempty and, moreover, everywhere dense.

92 Prove Baire theorem starting with any string and adding suffixes to get inside dense open sets (one by one).

Now we consider effectively open sets (unions of enumarable families of intervals) that are everywhere dense. We get a countable family of open sets that are dense everywhere. Their intersection is an everywhere dense sets whose elements are called *generic* sequences.

Informally speaking, generic sequence violates any law that prohibits a enumerable dense set of prefixes.

93 Prove that every generic sequence viiolates the Strong Law of Large Numbers. [Hint: The set of binary strings of length greater that N that have more than 99% of ones forms a dense effectively open set; the same is true for the set of strings with more than 99% zeros.]

94 Prove that no generic sequence is computable. [Hint: the set of all sequences that differ from a given computable sequence is open and everywhere dense.]

Note that the definition of a generic sequence (unlike randomness) does not refer to any measure.

95 Prove that a generic sequence is not ML-random with respect to any computable measure. [Hint: It is enough to construct an effectively open dense set that has small measure. This can be done by iteratively chosing a smaller half of an interval, or almost smaller if the halves have almost equal size.]

Zvonkin and Levin ([**?**], remark after Definition 4.4) claim that it is easy to show that the characteristic sequence of the universal enumerable set is not ML-random with respect to any computable measure. They don't say what kind of universality is needed etc., but most probably the statement they had in mind follows from the following result:

96 Show that there exists an enumerable set whose characteristic sequence is not random with respect to any computable measure. [Hint: The complexity of the prefixes of every characteristic sequence of an enumerable set is logarithmic; it remains to guarantee that any computable measure of the prefixes decreases fast. It can be done as follows: we split $\mathbb{N}$ into arithmetic sequences and devote ith of them to ith computable measure; since we don't know whether it is indeed a computable measure, we get an enumerable set, not a decidable one.]

2. It is more difficult (though still possible) to construct a sequence that is not Turing-equivalent to any sequence that is ML-random with respect to the uniform measure. Moreover, we know which direction is difficult: we can construct a sequence α such that no ML-random sequence is reducible to α. Moreover, one can construct a probabilistic machine that generates such sequences with positive probability. This is done in [**?**].

3. As we shall see, the converse statement is false: every sequence is Turing-reducible to some ML-random sequence (with respect to the uniform measure, see Theorem 101, p. 147). {gacs-reduc

4. One can also construct a sequence that is Turing-equivalent to a ML-random one (with

respect to the uniform measure) but is not random with respect to any computable measure. For example, we may interleave (using even and odd places) a generic sequence τ with a random sequence ω such that τ is Turing-reducible to ω.

(Here we use the results mentions in the preceding paragraph. Note also that if a sequence is ML-random with respect to some computable measure P that its subsequence formed by the terms with even indices in random with respect to the projection of the measure P.)

The sequences that are not ML-random with respect to any computable measure are somehow similar to nonstochastic (in Kolmogorov sense) objects (see Section **??**). Moreover, it is easy to see that if a sequence is ML-random with respect to some computable measure, then its prefixes are stochastic (Problem **??**, page **??**).

In the end of this section we present the following easy corollary of the results proved above:

{monotsm-ar

Theorem 70 *If, in place of the uniform distribution, the random bits used by a probabilistic machine have another computable probability distribution, then the distribution generated by the machine is still lower-semicomputable.*

$\lhd$ Indeed, every computable distribution is an output distribution for some probabilistic machine, and therefore combining two machines we get a machine having the same output distribution and the uniform source distribution.

One may also repeat the proof of Theorem 66 and notice that the intervals discovered have computable measures and thus we get lower bounds for probabilities. $\rhd$

5.2 Maximal semimeasure on the binary tree

{monotmax}

Theorem 71 *The class of all lower semicomputable semimeasures on the binary tree has the greatest element* (*up to a constant factor*)*: there exists a semimeasure a in this class such that for every other a' in the same class the inequality $a'(x) \leqslant ca(x)$ holds for some constant c and for all x.*

This element is traditionally called the *maximal lower semicomputable semimeasure on the binary tree* (though it is not only the maximal, but also the greatest element in the partial order), or the *universal semimeasure* on the binary tree.

$\lhd$ We can use the same idea as for semimeasures on $\mathbb{N}$ (Theorem 41, p. 69). Consider a probabilistic machine A that first chooses at random some probabilistic machine and then simulates it. If a semimeasure a' corresponds to a probabilistic machine A', then $a'(x) \leqslant (1/\varepsilon)a(x)$ where ε is the probability that machine A' is chosen. $\rhd$

Another proof deals with functions, not machines: first we construct a sequence $a_0, a_1, \ldots$ of semimeasures and then consider the function $a = \sum_i \lambda_i a_i$ where λ_i are computable coefficients that have sum 1 (e.g., $\lambda_i = 2^{-i-1}$).

A delicate point: we need a sequence that includes all (tree) semimeasures that are computable from below, and the sequence itself should be computable from below. This means that we need a lower semicomputable function $\langle i,x\rangle \mapsto u(i,x)$ such that (1) for any fixed i the function $u_i : x \mapsto u(i,x)$ is a tree semimeasure; (2) the sequence u_i contains all lower semicomputable tree semimeasures.

This can be done either by enumerating all probabilistic machines (and that corresponds to the first proof) or by enumerating all lower semicomputable functions and then "trimming" them to make them semimeasures and leaving them unchanged if they already are semimeasures. See the similar argument for semimeasures on $\mathbb{N}$ (Section 4.2, p. 69). Note that if the condition $p(x) \geqslant p(x0) + p(x1)$ is violated, we should increase $p(x)$ unless this makes $p(\Lambda)$ greater than 1.

97 Provide the missing details in this argument.

(Remark: The proof gives a bit more than we have claimed. Indeed, we get a lower bound not only for the probability of the event "output *starts with* x", which is $p(x)$, but also a lower bound for the probability of the event "the output is *exactly* x", which is $p(x) - p(x0) - p(x1)$. So not only $a(x)$, but also $a(x) - a(x0) - a(x1)$ is maximal for the universal machine we constructed.)

98 Prove that all these arguments can be applied to the case of algorithms that send natural numbers (not bits) to the output one at a time. These algorithms correspond to lower semicomputable semimeasures on the set of all (finite and infinite) sequences of natural numbers.

99 (Continued.) Let m be the maximal lower semicomputable semimeasure on the set of all finite and infinite sequences of natural numbers. Show that its restriction on the sequences of length 1 coincides (up to $O(1)$ factor) with the a priori probability on natural numbers (Chapter 4), and its restriction to binary sequences coincides (up to $O(1)$ factor) with the universal tree semimeasure we have defined in this section.

Let us fix some maximal lower semicomputable semimeasure on the binary tree and denote it by $a(x)$. One can call $a(x)$ an *a priori probability of a tree vertex* x, however, one should distinguish it from the a priori probability defined in Chapter 4. However, we can consider the expression

$$KA\,(x) = -\log a(x)$$

and call it *a priori complexity* of a string x. (This does not create any confusion, since in Chapter 4 the logarithm of the maximal semimeasure coincides with the prefix complexity and does not require a special name.) Since different maximal semimeasures differ at most by $O(1)$ factor, the a priori complexity is defined up to an additive $O(1)$ term.

In the next section we study the properties of a priori complexity. Let us note that by definition the a priori complexity need not be an integer (or even rational) number. But this does not matter much, since most of the statements about complexity are true "up to $O(1)$ term", and we may replace $-\log a(x)$ by a minimal integer n such that $a(x) > 2^{-n}$. An important detail: we use the strict inequality since we want the resulting function to be lower semicomputable. In the sequel we indicate the rare cases where this rounding (or its absence) can be important.

5.3 A priory complexity and its properties

{kanptape}

Theorem 72 **(a)** $KA\,(x) \leqslant l(x) + O(1)$ *for each* x.

(b) $KA\,(x) \leqslant KP\,(x) + O(1)$ *for each* x.

(c) *Let* $x_0, x_1, \ldots$ *be a computable sequences of incomparable strings (i.e., none of them is a prefix of another one). Then* $KA\,(x_i) = KP\,(x_i) + O(1) = KP\,(i) + O(1)$.

(d) $KP\,(x) \leqslant KA\,(x) + 2\log l(x) + O(1)$.

(**e**) *Moreover,* $KP(x) \leqslant KA(x) + KP(l(x)) + O(1)$,

(**f**) *and even more,* $KP(x|l(x)) \leqslant KA(x) + O(1)$*;*

(**g**) *A sequence of zeros and ones is computable if and only if a priori complexity of its prefixes is bounded.*

(**h**) *If* $f : \Sigma \to \mathbb{N}_\perp$ *is a computable continuous mapping, then* $KP(f(x)) \leqslant KA(x) + O(1)$ *for each string* x *such that* $f(x)$ *is defined (is not equal to* $\perp$*).*

$\lhd$ (a) The function $p(x) = 2^{-l(x)}$ is a lower semicomputable semimeasure. Therefore $p(x) \leqslant ca(x)$ for some c and all x.

(b) The machines that print a binary string (as a whole) and then halt, form a subclass of the machines that generate output bits one by one. Therefore, $m(x) \leqslant ca(x)$ where m is the a priori probability as defined in Chapter 4.

It is instructive to rephrase this argument using semimeasures. Let $m'(x)$ be the sum of $m(y)$ taken over all strings y that are prefixes of x (including x itself). Here m is the maximal semimeasure on $\mathbb{N}$ as defined in Chapter 4. Modify m' and let $m'(\Lambda)$ be equal to 1. Then m' is a semimeasure on the binary tree and therefore $m(x) \leqslant m'(x) = O(a(x))$.

(c) Let x_i be a computable sequences of incomparable binary strings. The function $i \mapsto a(x_i)$ (where a is the a priori probability on the tree) is a lower semicomputable semimeasure on $\mathbb{N}$. Indeed, it is lower semicomputable and the events "output starts with x_i" are disjoint and therefore the sum of their probabilities does not exceed 1. Therefore $KP(i) \leqslant KA(x_i) + O(1)$.

On the other hand, $KP(x_i) = KP(i) + O(1)$, since i can be algorithmically transformed into x_i and vice versa; finally, $KA(x_i) \leqslant KP(x_i) + O(1)$ according to (b).

(d) Let a be the universal semimeasure (a priori probability) on the binary tree. Consider the function u defined as $u(x) = a(x)/l(x)^2$. It is lower semicomputable. Moreover, since the sum of $a(x)$ over all strings x of length n does not exceed 1 (these strings are not prefixes of each other), we get

$$\sum_x u(x) = \sum_n \sum_{l(x)=n} \frac{a(x)}{n^2} \leqslant \sum_n \frac{1}{n^2} = O(1),$$

so we get the desired inequality.

(e) can be proved in a similar way, this time we let $u(x) = a(x)m(l(x))$ where m is a priori probability on $\mathbb{N}$ (as defined in Chapter 4).

(f) Consider the function

$$u(x,n) = \begin{cases} a(x), \text{ if } l(x) = n, \\ 0, \text{ if } l(x) \neq n. \end{cases}$$

Then for each n the function $x \mapsto u(x,n)$ is a semimeasure in the sense of Chapter 4 (the sum of values does not exceed 1), and we get the desired inequality.

(g) For a give computable (infinite) sequence ω of zeros and ones consider a "probabilistic" algorithm that ignores random bits and just computes and sends to the output the sequence ω (bit by bit). The corresponding semimeasure equals 1 on any prefix of ω, therefore the universal semimeasure (whose logarithm is a priori complexity) of all prefixes of ω is greater that some positive constant.

The converse implication is a bit more complicated. Assume that a priori probabilities (the values of the universal semimeasure a on the binary tree) of all prefixes of ω are greater than some rational $\varepsilon > 0$. Consider the set B of all binary strings x such that $a(x) > \varepsilon$. The set B contains all prefixes of ω and is a subtree (if some string is in B, then all its prefixes are in B). Moreover, any prefix-free subset of B (that does not contain a sequence and its prefix at the same time) has at most $1/\varepsilon$ elements (since the corresponding events are disjoint, their total probability does not exceed 1). Finally, the set B is enumerable (having more and more precise approximations to $a(x)$ from below, we eventually discover all elements in B).

These properties of B are sufficient to conclude that the sequence ω is computable. Indeed, consider the maximal (having the maximal cardinality) prefix-free subset $x_1, \ldots, x_N$ of B. For each of x_i consider all its continuations that belong to B. All of them (for a given i) are prefixes of one sequence; otherwise we can find two inconsistent strings and replace x_i by them (which is not possible, since the subset is maximal).

So for each i we have a (finite or infinite) branch in B going through it, and it is computable since B is enumerable. The sequence ω is one of these branches (otherwise we could add a sufficiently long prefix of ω to the set which is maximal).

(h) Consider the probabilistic machine that corresponds to the maximal semicomputable semimeasure on the binary tree, and apply function f to its output. This composition is a probabilistic machine as defined in Chapter 4, and it remains to compare it to the universal machine that generates the maximal lower semicomputable semimeasure on $\mathbb{N}$ (logarithm of this semimeasure is $KP + O(1)$). $\triangleright$

Note that the a priori complexity is quite different from the complexities already known (plain and prefix complexities). Its definition uses a tree structure that exists on the set of finite binary strings, and algorithmic transformation that ignore this structure can increase a priori complexity more than by $O(1)$.

100 Show that one can find a string x that has $O(1)$ a priori complexity but x^R (reversed x) has arbitatrily large complexity. (Formally: there exists c such that for every n there is a string x satistying the inequalities $KA(x) < c$ and $KA(x^R) > n$.) [Hint: the string x can be of the form $1000\ldots$]

So (unlike before) we cannot speak about a priori complexity of some constructive object (a pair, a graph, a finite set etc.) since it depends on the encoding.

The difference between a priori complexity of a string x of length n and other complexities of x (plain, prefix) is still $O(\log n)$. However, it is important that n stands for the length of x, not for the complexity of x. (For example, if x is a string of n zeros, its a priori complexity is bounded while plain and prefix complexities are not.)

101 Prove that the differences $KS(x) - KA(x)$ and $KA(x) - KS(x)$ could be of order $\log n$ for some strings of length n (and for arbitrarily large n). [Hint: $KS(x)$ can be much greater than $KA(x)$ if x consists of zeros only. On the other hand, $KS(x)$ is greater than $KA(x)$ if x is a prefix of a ML-random sequence; in this case $KA(x) = l(x) + O(1)$, but $KS(x)$ can be smaller than $l(x)$ by $\log l(x)$, see Problem 38.]

102 Prove that

$$KA(xy) \leqslant KP(x) + KA(y) + O(1),$$

where xy is the concatenation of strings x and y. It is important that x is on the left of y: for $KA(yx)$ the the statement is false. [Hint: Let U be a probabilistic algorithm in the sense of Chapter 4 that generates the maximal lower semicomputable semimeasure (a priori probability) on strings as isolated objects, considered in Chapter 4. Let V be the probabilistic algorithm in the framework of this chapter that generates maximal semimeasure on the binary tree. Then combine U and V as follows: first, run U until it prints something and terminates. Then run V using the rest of the input and add its outbut bits to the string generated by U. To show that $KA(xy)$ cannot be replaced by $KA(yx)$, let $y = 0^n$ and $x = 1$.]

103 Prove that for each string x at least one of the numbers $KA(x0)$ and $KA(x1)$ is at least $KA(x)+1$. (Here it is important that $KA(x)$ is defined as $-\log a(x)$ without rounding). Using this observation, prove that for any string x and for any integer $n \in N$ there exists a string y of length n such that $KA(xy) \geqslant KA(x) + n$. {increasing

(Cf. Theorem 65 on p. 102 and Problem 34 on p. 38; note that now we do not have n as condition and even do not have term $O(1)$ in the inequality.)

Another property of the a priory complexity is an immediate consequence of its definition. Let μ be a computable measure on Ω. Then for some c and every x we have

$$KA(x) \leqslant -\log \mu(\Omega_x) + c$$

Indeed, the a priori probability on the binary tree is greater than μ (or any other computable measure, or even lower semicomputable semimeasure) up to a $O(1)$ factor, and it remains to take logarithms.

This (very simple) property is important since it is the basis for a criterion of Martin-Löf randomness in terms of a priori complexity: a sequence ω is ML-random with respect to a computable measure μ if and only if this inequality turns into an equality for prefixes of ω, i.e., if the difference $-\log \mu(\Omega_x) - KA(x)$ has a constant upper bound for all x that are prefixes of ω (it always has a constant lower bound as we just mentioned).

This criterion follows from Levin–Schnorr theorem that provides randomness criterion in terms of monotone complexity and we postpone its proof to Section 5.6 where Levin–Schnorr criterion is considered. But first we have to define monotone complexity (Section 5.5) and this definition uses the notion of a computable mapping of the space Σ into itself (Section 5.4).

One can characterize a priori complexity as the smallest upper semicomputable (=enumerable from above) function that satisfies some condition (similar characterization for plain complexity was provided by Theorem 8 (p. 21) and by Theorem 56 (p. 89) for prefix complexity). Here is the corresponding statement:

{ka-criteri

Theorem 73 *The function KA is a minimal (up to an additive constant) upper semicomputable function K such that*

$$\sum_{x \in M} 2^{-K(x)} \leqslant 1$$

for any prefix-free set M of binary strings.

$\lhd$ Since the strings $x \in M$ are inconsistent (none of them is a prefix of another one), the corresponding sets Σ_x (of all finite and infinite sequences with prefix x) are disjoint and the sum of probabilities does not exceed 1.

On the other hand, let K be an upper semicomputable function that satisfies this condition. We have to construct a lower semicomputable semimeasure that is greater that 2^{-K}. The latter function is lower semicomputable but is not necessarily a semimeasure; its values on x, $x0$ and $x1$ can be unrelated. So we need first to increase K when it is unavoidable. Let $a(x)$ be the least upper bound of all the sums of the form

$$\sum_{x\in M} 2^{-K(x)}$$

over all prefix-free sets of strings that start with x. It is easy to check that $a(x)$ is indeed a lower semicomputable semimeasure and $2^{-K(x)}$ does not exceed $a(x)$. Theorem is proved. $\rhd$

5.4 Computable mappings of type $\Sigma \to \Sigma$

{tree-mappi

The algorithms (machines) used in the definition of the universal semimeasure on the binary tree consist of two parts: the random bit generator and the algorithm that transforms the sequence of random bits into the output. In this section we look more closely at this second part and introduce the notion of a computable mapping of the set Σ (of all finite and infinite sequences of zeros and ones) into itself. Let us stress that we consider mappings that are defined on the entire Σ; however, some of their values can be equal to the empty string Λ (that represents an "undefined value" in a sense).

5.4.1 Continuous mappings of type $\Sigma \to \Sigma$

{tree-conti

Let $f\colon \Sigma \to \Sigma$ be a mapping defined on the entire Σ. We say that f is *continuous* if it has the following two properties:

(1) f is monotone: if $x \in \Sigma$ is a prefix of some $y \in \Sigma$, then $f(x)$ is a prefix of $f(y)$. (Each sequence is a prefix of itself.)

(2) The value $f(\omega)$ for an infinite sequence ω is the least upper bound of the values $f(x)$ on all finite prefixes x of the sequence ω.

We use the notation $x \preccurlyeq y$ for the relation "x is a prefix of y"; here $x, y \in \Sigma$ may be finite or infinite. We have $x \preccurlyeq x$ for any x; if $x \preccurlyeq y$ for an infinite sequence x, then $x = y$. The requirement (1) says that f is monotone with respect to the partial order $\preccurlyeq$ on Σ. This requirement guarantees that the values $f(x)$ for all finite prefixes x of some sequence ω are consistent (continue each other); their "union" (=least upper bound under $\preccurlyeq$-ordering) coincides with $f(\omega)$ due to (2).

104 Show that the notion of continuity defined above is the standard continuity notion with respect to the topology on Σ defined in Section 4.4.3 (p. 79). [Hint: a very similar notion of continuous mappings $\Sigma \to \mathbb{N}_\perp$ was studied in the same section.]

Let $f\colon \Sigma \to \Sigma$ be a continuous mapping. Consider the set Γ_f that consists of all pairs $\langle x, y\rangle$ of binary strings x and y such that $y \preccurlyeq f(x)$. (The set Γ_f may be called the *lower graph* of the mapping f.)

For any continuous $f\colon \Sigma \to \Sigma$ the set Γ_f has the following three properties:

(1) $\langle x, \Lambda\rangle \in \Gamma_f$ for every string x;

(2) If $\langle x,y\rangle \in \Gamma_f$, then $\langle x',y'\rangle \in \Gamma_f$ for every $x' \succcurlyeq x$ and $y' \preccurlyeq y$.

(3) If $\langle x,y_1\rangle$ and $\langle x,y_2\rangle$ belong to Γ_f, then the strings y_1 and y_2 are consistent (one of them is a prefix of another one).

The first two properties are obvious. The third one is true since any two prefixes of a (finite or infinite) sequence are consistent.

The following theorem shows that a continuous mapping is defined uniquely by its lower graph.

{continuous

Theorem 74 *The mapping $f \mapsto \Gamma_f$ is one to one correspondence between continuous functions of type $\Sigma \to \Sigma$ and sets of pairs of strings that satisfy conditions* (1)–(3)*.*

$\lhd$ Let f be a set of pairs satisfying the conditions (1)–(3). These conditions guarantee that for any string x the set F_x of all y such that $\langle x,y\rangle \in F$ is non-empty and every $y_1, y_2 \in F_x$ are consistent. Let $f(x)$ be the least upper bound of F_x. The property (2) guarantees that $x \preccurlyeq x'$ implies $f(x) \preccurlyeq f(x')$ (since F_x increases as x increases). Therefore we may define $f(\omega)$ as the union (least upper bound) of $f(x)$ for all strings $x \preccurlyeq \omega$. Then the mapping f is continuous. It is easy to check that we get a mapping which is an inverse mapping to the correspondence $f \mapsto \Gamma_f$. $\rhd$

5.4.2 Monotone machines with non-blocking read operation

A continuous mapping $f\colon \Sigma \to \Sigma$ is called *computable* if the corresponding set Γ_f is enumerable. (By definition all computable mappings are continuous.)

This definition is complete and does not require any interpretation in terms of machines. All we say below about the interpretation of this notion is terms of machines of special type is not necessary (and is not used in the sequel). However, to get a motivation for this definition it is instructive to understand which type of machines (programs) corresponds to computable continuous mappings of type $\Sigma \to \Sigma$.

Let us consider programs that use a non-blocking read operation (we can get the next bit from the input queue and also check whether this queue is nonempty). We have discussed this type of input paradigm in Section 4.4.2, p. 77. However, now we assume that the output is created bit by bit, using the procedure call *OutputBit*(b) with a Boolean argument.

The output sequence generated by a program of this type can be finite or infinite. In general, it depends not only on the input sequence but also its timing (on the moments when keys "0" and "1" were pressed). We say that a machine (program) is *robust* if timing does not matter, i.e., if the output sequence depends only the input sequence but not on the timing. (Of course, the output timing may still depend on the input timing.) A robust program determines (computes) some mapping of the set Σ into itself.

Theorem 75 *Robust program compute computable mappings (in the abstract sense, as described above); every computable mapping is computed by some robust program.*

$\lhd$ Assume that M is a robust program. Let x and x' be two (finite or infinite) sequences such that $x \preccurlyeq x'$. Let us show that $M(x) \preccurlyeq M(x')$ where $M(z)$ stands for the output of program M on the

input z (since M is robust, the output depends only on z, not on the timing). If x is infinite, this is trivial ($x = x'$). Assume that x is finite. There are two possibilities: $M(x)$ is either finite or infinite.

If $M(x)$ is finite, let us submit input x and wait until $M(x)$ appears at the output. This should happen at some point; after that we submit the remaining bits of x' (that are not in x) to the input. Then we get output $M(x')$ which by construction is the extension of $M(x)$.

If $M(x)$ is infinite, then every bit of $M(x)$ should appear at some time after we submit x to the input. Since the remaining bits of x' can be sent after this moment, this bit should appear also in $M(x')$. Therefore, $M(x) = M(x')$ in this case.

It is also clear that for an infinite sequence ω the value $M(\omega)$ is the union of $M(x)$ for finite $x \preccurlyeq \omega$; indeed, at each moment only finite number of input bits are read.

The set of all pairs of strings x, y such that $y \preccurlyeq M(x)$ is enumerable since we can enumerate it by simulating the behavior of M on all inputs. So each robust machine computes a computable mapping.

On the other hand, let f be an arbitrary computable mapping. We show how to construct a robust machine M that computes it. The machine M enumerates the lower graph Γ_f of the mapping f. At the same time M reads input bits and stores them. If it turns out that Γ_f includes a pair $\langle x, y\rangle$ such that x is a prefix of the input sequence, we output the remaining bits of y (the requirements (2) and (3) guarantee that all the strings y found in this way are consistent so there is no need to recall the bits already sent to the output). $\triangleright$

5.4.3 The set of continuous mappings is enumerable

The definition of computability based on robust machines seems to be more natural than the abstract one. However, it has the same drawback as in the case of prefix-stable programs: there is no (algorithmic) way to find out whether a given program is robust. So the class of robust program is not a syntactically defined class.

Nevertheless, there exists an algorithmic transformation of programs that transforms every program into a robust one (and does not change the mapping computed by it if it was robust). This transformation goes back and forth between mappings and corresponding enumerable sets: we transform a program into an enumerable set of pairs (i.e., into an algorithm enumerating this set), then we "trim" this set of pairs and transform it back into a program.

We do not describe this process in detail, since robust programs are more a motivation for the definition of a computable mapping than a technical tool. Instead, we prove that the set of computable mapping is enumerable in the following sense:

{monot-enum

Theorem 76 *There exists an enumerable set U of triples $\langle n, x, y\rangle$ (here n is a natural number while x and y are binary strings) such that:*

(**1**) *for every n the set $U_n = \{\langle x, y\rangle \mid \langle n, x, y\rangle \in U\}$ is a lower graph of some computable mapping $u_n \colon \Sigma \to \Sigma$ (i.e., satisfies the requirements* (1)–(3) *of Theorem 74).*

(**2**) *every computable mapping of the set Σ into itself is equal to u_n for some n.*

$\triangleleft$ Consider the universal enumerable set W of triples: every enumerable set of pairs appears among W_n. Then we "trim" W to enforce the requirements (1)–(3) for all W_n and leave unchanged

the sets W_n that already satisfy these requirements. After that all W_n are lower graphs for some computable mappings w_n and any computable mapping appears among w_n.

The trimming is made in two steps: first we "delete contradictions" and then we "fill the gaps". The contradiction is formed by two pairs $\langle x_1, y_1\rangle$ and $\langle x_2, y_2\rangle$ where x_1 is consistent with x_2 but y_1 is not consistent with y_2. (It is easy to see that two pairs with this property cannot appear simultaneously in the lower graph of a continuous mapping.) The contradictions are eliminated in the most simple way: if a pair contradicts with another pair that is generated earlier and was not deleted, we delete this pair from the enumeration. In this way we get an enumerable set without contradictions. The we fill the gaps by adding all pairs $\langle x, \Lambda\rangle$ and adding for each pair $\langle x, y\rangle$ all the pairs $\langle x', y'\rangle$ with $x' \succcurlyeq x$ and $y' \preccurlyeq y$. It is easy to see that the set remains enumerable and is the one we need. $\triangleright$

This theorem is used in the next section to prove that (optimal) monotone complexity function exists.

5.5 Monotone complexity

{monotone-c

To define monotone complexity we use computable mappings of type $\Sigma \to \Sigma$ as decompressors (description modes). For a fixed decompressor $D\colon \Sigma \to \Sigma$ the *monotone complexity* of a string x (with respect to D) is defined as the minimal length of a string y such that $x \preccurlyeq D(y)$. Monotone complexity is denoted by $KM_D(x)$.

(This definition can be applied to infinite sequences x without any changes but we follow the tradition and consider $KM_D(x)$ only for finite x unless the opposite is said explicitly.)

105 Prove that the monotone complexity of an infinite sequence (defined in a natural way) is the limit of the increasing sequence of monotone complexities of its prefixes.

Theorem 77 *There exists an optimal decompressor, i.e., a computable mapping $D\colon \Sigma \to \Sigma$ such that KM_D is minimal up to additive constant: for any computable $D'\colon \Sigma \to \Sigma$ there exists a constant c such that*

$$KM_D(x) \leqslant KM_{D'}(x) + c$$

for every string x.

$\triangleleft$ Let U be the set of triples whose sections are all lower graphs of all computable mappings (constructed in Theorem 76, p. 119). Let D_n be a computable mapping that has lower graph U_n. Then let us define a mapping D as follows:

$$D(\hat{n}z) = D_n(z),$$

where $\hat{n}$ is the prefix-free encoding of the number n (say, its binary representation with doubled digits followed by 01) and z is an arbitrary element of Σ. In terms of the lower graph: consider the set of all pairs $\langle \hat{n}u, v\rangle$ such that $\langle n, u, v\rangle \in U$. It is easy to check that we indeed get a computable mapping. If some (monotone) decompressor D' has number n (i.e., its lower graph coincides with U_n), then $KM_D(x) \leqslant KM_{D'}(x) + l(\hat{n})$ for every x. $\triangleright$

As usual, we fix some optimal monotone decompressor (description mode), i.e., some computable mapping D that satisfies the statement of this theorem and define *monotone complexity* of a string x as $KM_D(x)$. Notation: $KM(x)$ (the subscript D is omitted).

Theorem 78 (**a**) *Monotone complexity is a monotonic function:* $KM(x) \leqslant KM(y)$ *if* $x \preccurlyeq y$;

(**b**) *the function KM is upper semicomputable;*

(**c**) $KM(x) \leqslant l(x) + O(1)$;

(**d**) $KM(x) \leqslant KP(x) + O(1)$;

(**e**) $KA(x) \leqslant KM(x) + O(1)$;

(**f**) *an infinite sequence of zeros and ones is computable if and only if the monotone complexity of its prefixes is bounded;*

(**g**) *if* $f\colon \Sigma \to \Sigma$ *is a computable mapping, then* $KM(f(x)) \leqslant KM(x) + O(1)$ (*the constant hidden in* $O(1)$ *may depend on* f *but not on* x)*;*

(**h**) *if* $f\colon \Sigma \to \mathbb{N}_\perp$ *is a computable mapping, then* $KP(f(x)) \leqslant KM(x) + O(1)$ (*the constant hidden in* $O(1)$ *may depend on* f *but not on* x).

It is instructive to compare these statements with the properties of a priori complexity given in Theorem 72 (p. 113). Since monotone complexity is not smaller than a priori complexity (statement (e)), some properties of the a priori complexity are valid also for monotone complexity. In particular, we conclude immediately that $KP(x|l(x)) \leqslant KM(x) + O(1)$ and $KP(x) \leqslant KM(x) + KP(l(x)) + O(1)$. Note also that for computable sequences of inconsistent strings (none is a prefix of another one) the prefix and a priori complexities coincide up to an additive constant and monotone complexity is between them. Therefore it coincides with them: if $x_0, x_1, \ldots$ is a computable sequence and $x_i \not\preccurlyeq x_j$ for $i \neq j$, then $KM(x_i) = KA(x_i) + O(1) = KP(x_i) + O(1)$.

$\lhd$ The statement (a) is a direct consequence of the definition: if $D(u) \succcurlyeq y$ then $D(u) \succcurlyeq x$ for every x that is a prefix of y. One could say that in the definition of monotone complexity one need to describe not the string exactly, but any of its continuations, and the longer the string is, the more difficult this task becomes (the set of continuations becomes smaller).

The statement (b) is true since the lower graph of a computable mapping is enumerable.

To prove (c) it is enough to remark that the identity mapping $\Sigma \to \Sigma$ such that $D(x) = x$ for all $x \in \Sigma$ is computable.

To compare KM and KP (statement (d)) it is enough to note that any computable mapping $\Sigma \to \mathbb{N}_\perp$ becomes a computable mapping of type $\Sigma \to \Sigma$ if $\mathbb{N}_\perp$ is embedded into Σ (and $\perp$ becomes an empty string). More formally, let D be a prefix-stable decompressor used in the definition of KP. In can be extended to a computable mapping of type $\Sigma \to \Sigma$ (the strings where D was undefined are mapped into Λ and the values on infinite strings are determined by the continuity requirement).

To compare KM and KA (statement (e)) we have to recall the remark we started with: a probabilistic algorithm is a random bits generator whose output is fed into a computable mapping of Σ into itself. Let D be the optimal decompressor used in the definition of the monotone complexity. Consider a probabilistic algorithm that feeds random sequence into D. What is a probability of getting some string x (or some string that starts with x) as the output? Obviously, this probability is

at least $2^{-l(y)}$ for any string y such that $D(y) \succcurlyeq x$, since the random string starts with y with probability $2^{-l(y)}$ and this guarantees that the output of D will start with x. (We return to the comparison of KM and KA in Theorem 80.)

The statement (f): one implication is a straightforward corollary of the corresponding statement of Theorem 72; the other implication is obvious, all the prefixes of a computable sequence ω have bounded complexity since there exists a computable mapping $\Sigma \to \Sigma$ that is equal to ω everywhere.

For (g), let us consider the monotone decompressor that is the composition of an optimal monotone decompressor and the mapping f. In this statement the sequence $f(x)$ can be infinite; if we don't want to deal with the complexities of infinite sequence, the statement should be reformulated as follows: for each f there exists a constant c such that for all x, y such that $y \preccurlyeq f(x)$ the inequality $KM(y) \leqslant KM(x) + c$ holds.

The similar argument works for (h), but this time the composition of the optimal monotone decompressor and f is a prefix-stable decompressor. (One can also derive this statement from a similar statement about a priori complexity.) $\triangleright$

106 Prove that $KM(xy) \leqslant KP(x) + KM(y) + O(1)$ (here xy stands for the concatenation of strings x and y). In particular, $KM(xy) \leqslant KP(x) + l(y) + O(1)$. [Hint: Consider the optimal prefix-free decompressor D_p and optimal monotone decompressor D_m. Now let $D'(uv) = D_p(u)D_m(v)$ (when D_p stops reading the input, the remaining part of the input is read by D_m.] {prefix-mor

107 Show that in the preceding problem one can replace $KM(y)$ by the "conditional" monotone complexity $KM(y|x)$ defined in a natural way (we do not require "monotonicity" with respect to the condition x, see Chapter 6 for details). {kp-km}

108 Prove that the statement (g) remains true if we replace KM by KA (in the both sides of the inequality). [Hint: the mapping f can be applied to the output of a probabilistic machine; the new machine is not better than the optimal one.]

We can give an equivalent definition of the monotone complexity that does not use computable mappings of type $\Sigma \to \Sigma$; in this way we get a simpler (but less natural, in our opinion) definition.

Let Ξ be the set of all binary strings. Consider the binary relation "to be compatible" (or "consistent") on this set: x is compatible with y if $x \preccurlyeq y$ or $y \preccurlyeq x$ (equivalent property: x and y are prefixes of the same string). An enumerable set (binary relation) $D \subset \Xi \times \Xi$ is called *regular*, if it has the following property:

$$\langle x_1, y_1\rangle \in D,\ \langle x_2, y_2\rangle \in D \quad \text{and} \quad (x_1 \text{ is compatible with } x_2) \ \Rightarrow\ (y_1 \text{ is compatible with } y_2)$$

for all x_1, x_2, y_1, y_2. Then the monotone complexity of a string y with respect to D is defined as the minimal length of a string x such that $\langle x, y\rangle \in D$. There is an optimal regular enumerable binary relation on Ξ.

109 Prove that this definition leads to a notion of monotone complexity that is differs from the previous one by at most $O(1)$. [Hint: The lower graph of any computable mapping $\Sigma \to \Sigma$ is a regular binary relation. On the other hand, if D is a regular binary relation, the "gap filling" described in the proof of Theorem 76 makes it a lower graph of some computable mapping.]

It is instructive to compare this definition with the definition of plain complexity (where we use graphs of computable functions, i.e., uniform enumerable sets, instead of regular relations D). In

the definition of monotone complexity we do not require D to be uniform: several pairs $\langle x,y\rangle$ with the same x and different y are allowed; we require only that all y's in these pairs are compatible. This makes KM smaller; for example, all prefixes of some computable sequence (say, $0000\ldots$) have bounded complexity (note that $KS(0^n) = KS(n)$ is about $\log n$ for most n).

On the other hand we put additional restrictions: if a string x is a description of some string y, then the strings that are compatible with x can be descriptions only of strings that are compatible with y. This makes complexity larger. This is especially clear when we consider complexities of the elements of a computable sequence of pairwise incompatible strings: monotone complexity in this case coincides with prefix complexity and the difference can be about $\log n$ for strings of length n.

Summing up (and recalling that both a priori complexity and plain complexity differ from the prefix one at most by $O(\log n)$ for strings of length n), we come to the following conclusion:

{km-ks-diff

Theorem 79 *The difference between $KS(x)$ and $KM(x)$ is bounded by $O(\log n)$ for strings of length n and may be both positive and negative with absolute value $\log n - O(1)$ for n bit strings for infinitely many n.*

We return to the comparison of different versions of complexity in Chapter 6. Now we mention (without proof) only one statement of this type:

{gacs-diffe

Theorem 80 *The difference $KM(x) - KA(x)$ is not bounded* (*from above*).

Let us explain the informal meaning of this theorem. Recall that in both definitions (of $KM(x)$ and $KA(x)$) we use computable continuous mapping $f\colon \Sigma \to \Sigma$ and consider the preimage of the set Σ_x of all sequences starting with x. Defining KA, we are interested in the measure of this preimage, while for KM we are looking for the largest interval of type Σ_y which is a subset of this preimage. This shows that $KA_f \leqslant KM_f$, and the difference can be large, if the preimage is "sparse" (consists of large number of small intervals). The question is how large this difference could be for an optimal computable mapping.

Recall out metaphor of space allocation (we allocate subsets of $[0,1]$ for a countable number of clients) used in the proofs of Theorem 40 (p. 68) and Theorem 52 (p. 83). The difference between prefix complexity and the logarithm of the a priori probability on $\mathbb{N}$ has the same nature (difference between the total measure and the maximal contiguous interval). However, in that case we were able to perform some kind of "consolidation" by modifying the description mode and the price was just a constant factor.

Now we have a more delicate task since our clients form a hierarchy. This makes reorganization more difficult and consolidation leads to more the constant factor overhead.

However, the technical details of this argument (given in P. Gacs's article [**?**]) are rather tedious, and authors are unable to understand and clearly explain this argument, so readers are referred to the original paper.

110 Prove that $KM(x) \leqslant KA(x) + O(\log KA(x))$. [Hint: in fact $KM(x|KA(x)) \leqslant KA(x) + O(1)$. Indeed, if $KA(x) = k$, then x at some point appears in the growing subtree of strings whose a priory complexity is less than $k+1$; this tree at all times has width (the cardinality of maximal antichain) at most 2^{k+1}, so looking at the maximal elements of this tree, we cover it by 2^{k+1} growing branches. For the complete argument see Theorem 102, p. 152.]

{km-ka-rela

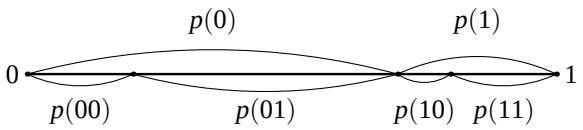

Figure 12: The construction of π_x.

{monotlev-p

5.6 Levin–Schnorr theorem

{monotlev}

The definition of a priori complexity guarantees that for any lower semicomputable semimeasure p the inequality $KA(x) \leqslant -\log p(x) + c$ holds for some c and for every x. It turns out that if p is a (computable) measure, then this inequality is true not only for a priori complexity KA but also for a (larger) monotone complexity KM.

{monotlev-u

Theorem 81 *Let μ be a computable probability distribution on Ω. Let $p(x) = \mu(\Omega_x)$. Then there exists a constant c such that*

$$KM(x) \leqslant -\log p(x) + c$$

for every string x.

$\lhd$ The idea of the proof can be explained as follows. The difference between KM and KA appears since we are unable to allocate contiguous space to hierarchical users' requests, since we do not know which of the current requests will increase in the future. However, if we have a measure (and not a semimeasure), we can solve this problem and allocate contiguous intervals. (Feel free to ignore this metaphor if it is confusing: we provide a formal proof in the next paragraphs.)

For each string x we define an interval π_x inside $[0,1]$. The interval π_x is defined in such a way that:

- the length of π_x equals $p(x)$;
- $\pi_\Lambda = [0,1]$ (here Λ is the empty string);
- for each string x the interval π_x is split by some its point into intervals π_{x0} (left part) and π_{x1} (right part).

(See Figure 12.)

We consider also another family of intervals that correspond to the uniform measure. Let I_x be the interval of reals whose binary representation starts with x. We call the intervals I_x *binary intervals*.

Now consider the set G of all pairs $\langle x,y\rangle$ of strings such that (binary) interval I_x is located inside the interior of π_y. The set G is enumerable. Indeed, since the function p is computable, we can find the endpoints of intervals π_y with arbitrary precision, and if they are greater (less) then some rational number, this fact will be eventually discovered.

Note also that the property $\langle x,y\rangle \in G$ remains true if we replace x by any string that starts with x (since the segment I_x becomes smaller) or replace y by any prefix of y (since π_y becomes larger). If

$\langle x,y_1\rangle \in G$ $\langle x,y_2\rangle \in G$, the segments π_{y_1} and π_{y_2} have a common interior point (they both contain I_x), therefore the strings y_1 and y_2 are compatible. So Theorem 74 (p. 118) guarantees that there exists a computable mapping of Σ into itself whose lower graph is G. We use this mapping as the decompressor in the definition of monotone complexity. Then $KM_D(y)$ equals to the minus binary logarithm of the biggest binary interval that is located strictly inside π_y. It remains to note that any open interval of length h contains a closed binary interval of length $h/4$, and compare D with the optimal decompressor. ▷

111 Prove the claim about binary intervals (see above). [Hint: let u be a power of two such that $h/4 \leqslant u < h/2$. Then any interval of length h intersects at least three consecutive binary intervals of length u and contains the middle one.]

Theorem 81 provides a theoretical justification for the following approach used by A.N. Kolmogorov and his students to get upper bounds for the complexity of Russian texts. While reading the text (one letter at a time), the reader tries to guess the next letter. The guess is formulated as a probability distribution over the alphabet. Then the next letter is read and we add $-\log p$ to the complexity, where p is the probability of that letter with respect to the guessed distribution.

If we believe that the behavior of the reader is computable, the result is an upper bound for the complexity. Indeed, the reader provides (some part) of a computable probability distribution on the set of strings telling the conditional probabilities along some path, and the complexity of text does not exceed the sum of negative logarithms of these probabilities (Theorem 81).

Of course, it is not practical to require that the reader provides at each step the list of probabilities for all the letters; one can suggest some standard types of answers like "the next letter is A with probability 0.5, all other vowels are equiprobable and have total probability 0.3, all other letters are equiprobable". Note also that we get an upper bound for the conditional complexity of the text where the condition is the background of the reader. (For example, if reader knows the text by heart, or just is familiar with the author's writings, the bound can be very small.)

Now we are ready to formulate the criterion of Martin-Löf randomness that uses monotone complexity: a sequence is ML-random if the inequality of Theorem 81 becomes an equality for its prefixes.

Let us formulate this statement precisely. Let μ be a computable probability distribution on the set Ω of all infinite bit sequences and let $p(x)$ be the measure of the interval Ω_x: $p(x) = \mu(\Omega_x)$.

{levin-schn

Theorem 82 (Levin–Schnorr) *A sequence $\omega \in \Omega$ is Martin-Löf random with respect to a computable probability distribution μ if and only if*

$$-\log p(x) - KM(x) \leqslant c$$

for some c and for every prefix x of ω.

◁ We have to prove theorem in both directions. Let us show first that if (for a given sequence ω) the difference $-\log p(x) - KM(x)$ is unbounded, then this sequence is not ML-random (i.e., the set $\{\omega\}$ is an effectively null set).

Fix some constant c and consider all strings x such that the difference $-\log p(x) - KM(x)$ is greater than c. (This difference is sometimes called *randomness deficiency*, but this term has

different meanings, e.g., in Chapter **??** it is used in a different way, so we avoid this name.) This set is denoted by D_c.

The set D_c is enumerable (since p is computable and KM is upper semicomputable, the difference is lower semicomputable).

Lemma 1. The set of all infinite sequences that have a prefix in D_c has μ-measure at most 2^{-c}.

Informally speaking, this is true because on this set the measure μ is 2^c times smaller than the a priori probability (and the latter does not exceed 1). More formally this argument can be explained as follows.

We are interested in the measure of the union of intervals Ω_x for all $x \in D_c$. Without changing this union, we may keep only minimal $x \in D_c$ (i.e., strings $x \in D_c$ such that no prefix of x belongs to D_c). Let $x_0, x_1, \ldots$ be these minimal elements of D_c. (We do not claim the the set of minimal elements is enumerable, so this sequence may be non-computable.)

For each x_i consider the minimal description p_i (according to the definition of the monotone complexity: $x_i \preccurlyeq D(p_i)$ where $D\colon \Sigma \to \Sigma$ is the optimal monotone decompressor). Then $l(p_i) = KM(x_i) < -\log p(x_i) - c$. Moreover, none of p_i is a prefix of another one (otherwise the corresponding x_i would be compatible). Therefore $\sum_i 2^{-l(p_i)} \leqslant 1$ (being the sum of uniform measures of disjoint sets Ω_{p_i}). The corresponding $p(x_i)$ are 2^c times smaller, so we get the statement of Lemma 1.

Our assumption guarantees that the sequence ω has prefixes from D_c for every c. To prove that $\{\omega\}$ is an effectively null set, we need to cover ω by an enumerable family of intervals with total measure not exceeding 2^{-c}, and we can use intervals from D_c.

However, we need to be careful here. We know that for intervals from D_c the total measure (i.e., the measure of their union) does not exceed 2^{-c} (as the Lemma says), but the definition needs that the sum of measures of all intervals does not exceed 2^{-c}. We cannot solve this problem by considering only minimal points (maximal intervals), since the set of minimal points is not always enumerable. Instead we can use the following statement:

Lemma 2. Any enumerable set of strings $x_0, x_1, \ldots$ can be transformed into an enumerable set of incompatible strings with the same union $\cup_i \Omega_{x_i}$. This transformation is effective (an algorithm that enumerates the first set can be transformed into an algorithm that enumerates the second one).

Indeed, if during the enumeration we get a string that is an extension of the previously enumerated one, this string can be omitted (since the corresponding interval is already covered). If we get a string y that is a (proper) prefix of a string x enumerated earlier, we have to split the difference $\Omega_y \setminus \Omega_x$ into finite number of disjoint intervals and replace y by strings that define those intervals. Lemma 2 is proved.

Applying Lemma 2, we get an enumerable set of incompatible strings; these strings may be not in D_c but that is not important. It is enough to know that they correspond to disjoint intervals that cover ω and the union of these interval has μ-measure at most 2^{-c} according to Lemma 1.

Proving the converse implication, we need to show that if a sequence ω belongs to an effectively null set then the differences between the negative logarithms of the measure and the monotone complexity of ω-prefixes are unbounded. The idea of this construction may be explained as follows: given a set of small measure, we construct a monotone decompressor that treats favorably the elements of this set (i.e., provides short descriptions for their prefixes).

Let us provide details now. Assume that ω belongs to a set U which is an effectively null set (with respect to measure μ). For each c we can effectively find a family of intervals $\Omega_{x_0}, \Omega_{x_1}, \ldots$ that cover U (and therefore ω) and have total measure less than 2^{-c}. If we multiply the measures of all these intervals by 2^c, the sum is still less than 1. Consider the computable sequence $p_i = 2^c \mu(\Omega_{x_i})$. Applying Theorem 53 (p. 86), we get a prefix-free decompressor for which the prefix complexity of i does not exceed $-\log \mu(\Omega_{x_i}) - c + 2$. A composition of this decompressor and the computable mapping $i \mapsto x_i$ is a prefix-free decompressor D_c such that

$$KP'_{D_c}(x_i) \leqslant -\log \mu(\Omega_{x_i}) - c + 2.$$

(The subscript c in D_c is used to stress that the construction depends on c.) Monotone complexity does not exceed the prefix one, so if the difference between the negative logarithm of the measure and the prefix complexity is large, the same is true for monotone complexity. It remains to combine the decompressors D_c into one decompressor (not depending on c).

We use the same trick that was was used in the construction of an optimal decompressor. We want the string $\hat{c}u$ to be the description of the string v if u is a description of v with respect to D_c. Here $\hat{c}$ is a self-delimited encoding of length $O(\log c)$ for a natural number c. If the decompressor D is constructed in this way, the following inequality holds (for all c):

$$KP'_D(x_i) \leqslant -\log \mu(\Omega_{x_i}) - c + O(\log c)$$

Since the monotone complexity does not exceed the prefix one, we replace $KP'_D(x_i)$ by $KM(x_i)$ and conclude that all the strings x_i (for a given c) have the difference at least $c - O(\log c)$. If an infinite sequence belongs to U, it has a prefix of this type for any c, therefore the difference is unbounded for its prefixes.

Levin–Schnorr theorem is proved. $\triangleright$

In fact the proof give us a bit more that we claimed. Here are several modifications of Levin–Schnorr theorem that can be extracted from it:

{levin-schr

Theorem 83 *We may replace the monotone complexity $KM(x)$ by the a priori complexity $KA(x)$ in the statement of the previous theorem.*

$\triangleleft$ The a priori complexity does not exceed the monotone one, so the difference may only increase. So we need to change only the first part of the proof. It is easy: in the proof of Lemma 1 we should note that $\sum_i 2^{-KA(x_i)} \leqslant 1$, since this sum is the some of a priori measures of disjoint intervals Ω_{x_i}. $\triangleright$

{levin-schr

Theorem 84 *We can also replace the monotone complexity $KM(x)$ by the prefix complexity $KP(x)$.*

$\triangleleft$ Here we go in the other direction and increase complexity, so only the second part of the proof needs to be redone. And this is trivial — recall that in fact we got just an upper bound for prefix complexity. $\triangleright$

Theorem 84 is nowadays the most popular version of Levin–Schnorr randomness criterion (see, e.g., [**?**]). However, authors still believe the it is more natural to use monotone (or a priori) complexity.

Indeed, for the monotone complexity the difference between the negative logarithm of the measure and the complexity is always positive and is bounded if and only if the sequence is random (To be more precise, the difference is always bounded from below and is bounded from above if and only if the sequence is random.) If we use prefix complexity instead, the difference can become negative. For example, in the case of the uniform measure $-\log\mu(\Omega_x)$ is just the length of string x, and prefix complexity may be greater than the length (the difference can be of order $\log n$, see Theorem 57, p. 90).

Moreover, the use of the monotone complexity allows us to strengthen the Levin–Schnorr theorem as follows:

{levin-schr

Theorem 85 *If a sequence ω is not random with respect to measure μ, then the difference $-\log p(x) - KM(x)$ for prefixes x (of ω) is not only unbounded, but also tends to infinity.*

$\lhd$ In the proof of theorem 82 we have constructed a prefix-free decompressor that provides short descriptions p_i for strings x_i and guarantees that the prefix complexity of x_i (with respect to this decompressor) does not exceed $-\log\mu(\Omega_{x_i}) - c$. To get the required bound for the monotone complexity, we may use (for each i) the extensions of p_i as descriptions of the extensions of x_i in such a way that the length of the descriptions corresponds to the measure of described strings, as it was done in the proof of Theorem 81 (p. 124).

More formally, we can use the inequality $KM(xy) \leqslant KP(x) + KM(y|x)$ (Problem 107) and the relativized version of Theorem 81 saying that $KM(y|x) \leqslant -\log\mu_x(\Omega_y)$ for any computable family of measures that (computably) depends on parameter x. Here μ_x is the measure that is concentrated on the set Ω_x and is defined as follows: $\mu_x(\Omega_y) = \mu(\Omega_{xy})/\mu(\Omega_x)$.

For the case of the uniform measure (where $-\log\mu(\Omega_x) = l(x)$) we can use a simpler argument and say that $p_i z$ is a description of $x_i z$ for any string z. $\rhd$

We provided some argument in favor of using monotone complexity in the randomness criterion. However, a version that uses prefix complexity has its own advantages. Note that the notion of a ML-random sequence is invariant under permutation of indices (if the measure is invariant or is changed according to the permutation), but the notion of a prefix (and therefore the criterion of randomness in terms of prefixes) is not. Using KP, one can get an invariant criterion of ML-randomness.

Let F be a finite set of indices (natural numbers) and let ω be a binary sequence. By $\omega(F)$ we denote the restriction of ω onto F, i.e., the binary string formed by bits ω_i such that $i \in F$ (in the increasing order of indices).

Let μ be a computable measure on Ω. For every finite set $F \subset \mathbb{N}$ and string Z whose length equals the cardinality of F, we consider the event $\omega(F) = Z$. Its μ-probability is denoted by $\mu_{F,Z}$.

112 Let ω be a ML-random sequence with respect to μ. Prove that

$$KP(F, \omega(F)) \geqslant -log\mu_{F,\omega(F)} - c$$

for some c and for all finite F.

[Hint: the measure of the set of all sequences for which this inequality does not hold for some fixed c, does not exceed 2^{-c} multiplied by the sum of a priori probabilities of all pairs F, Z and therefore does not exceed 2^{-c}.]

(Note that if F is an initial segment of $\mathbb{N}$, then F is determined by $\omega(F)$ and can be eliminated, so we return to the previous statement.)

In fact, the condition given by the last problem is also sufficient. Moreover, it is enough to require this inequality for any increasing computable sequence of finite sets that cover the entire $\mathbb{N}$.

113 Assume that μ is a computable probability distribution on Ω. Let $F_0 \subset F_1 \subset F_2 \subset \ldots$ be a computable sequence of finite sets and $\bigcup_i F_i = \mathbb{N}$. Assume that {levin-schn

$$KP(F_i, \omega(F_i)) \geqslant -\log \mu F_i, \omega(F_i) - c$$

for some c and for all i. Then ω is ML-random with respect to μ.

[Hint: Using permutation of indices, we may assume that F_i are initial segments of $\mathbb{N}$. Then we repeat the proof of Levin – Schnorr theorem using only strings of appropriate length and splitting other intervals into unions of appropriate intervals.]

This statement implies, for example, that a two-dimensional bit sequence (i.e., a mapping $\mathbb{Z}^2 \to \{0,1\}$) is ML-random with respect to the uniform measure (all bits are independent; 0 and 1 are equiprobable) if and only if $N \times N$ square centered at the origin has complexity $N^2 - O(1)$ (for all odd N).

The case of the uniform measure is rather important; let us write down all what we have proved for this case:

{levin-schn

Theorem 86 (**a**) *Upper bound:*

$$KA(x) \leqslant KM(x) + O(1) \leqslant l(x) + O(1);$$

for any string x.

(**b**) *Randomness criterion: the sequence ω is ML-random with respect to the uniform measure if and only if these inequalities become equalities for prefixes of ω:*

$$KA((\omega)_n) = KM((\omega)_n) + O(1) = n + O(1).$$

(**c**) *If ω is not ML-random with respect to the uniform measure, then the difference $n - KM((\omega)_n)$ (and therefore $n - KA((\omega)_n)$ tends to infinity as $n \to \infty$.*

(**d**) *The sequence ω is ML-random with respect to the uniform measure if and only if $KP((\omega)_n) \geqslant n - c$ for some c and for all n.*

(**e**) *The sequence ω is ML-random with respect to the uniform measure if and only if $KP(F, \omega(F)) \geqslant |F| - c$ for some c and for all finite sets F.*

For the case of the uniform measure there exists one more criterion of Martin-Löf randomness. It is interesting since it uses only plain complexity (and not the prefix or monotone versions). It is a bit strange that this criterion was discovered only recently (see [**?**]) since similar suggestions were considered in the end of 1960ies (see [**?**, **?**]), and the proof of this criterion uses only ideas and methods well known at that time.

{miller-yu-

Theorem 87 *Assume that $f\colon \mathbb{N}\to\mathbb{N}$ is a computable total function and the series $\sum 2^{-f(n)}$ converges. Let ω be a ML-random sequence with respect to the uniform measure. Then*

$$KS((\omega)_n|n) \geqslant n - f(n) - O(1)$$

(*i.e., there exists c such that for every n the inequality $KS((\omega)_n|n) \geqslant n - f(n) - c$ holds*).

$\lhd$ Assume that the claim is false. This means that for every c there exists n such that

$$KS((\omega)_n|n) < n - f(n) - c.$$

In other words, for every c the sequence ω is covered by some interval Ω_x such that

$$KS(x|n) < n - f(n) - c,$$

where n is the length of x. For each n there are at most $2^{n-f(n)-c}$ intervals with this property and their total measure is at most $2^{-f(n)}2^{-c}$ (for a given n). The total measure of all such intervals (for all n) is

$$2^{-c}\left(\sum_n 2^{-f(n)}\right)$$

and the sequence ω forms an effectively null set: choosing an appropriate c we get ω's cover that has small measure. Therefore, ω is not ML-random. (Note that the sum of the series $\sum 2^{-f(n)}$ may be a non-computable real number; this does not matter since we may use any upper bound for it.) $\rhd$

This theorem implies, for example, that for any ML-random sequence (with respect to the uniform measure) the plain complexity of its prefix of length n is at least $n - 2\log n - O(1)$ and even $n - \log n - 2\log\log n - O(1)$, since the corresponding series converge.

Making function f smaller, we get the claim of the theorem stronger. It turns out that for some f we get a randomness criterion in this way:

{miller-yu-

Theorem 88 *There exists a total computable function $f\colon \mathbb{N}\to\mathbb{N}$ with the following property: if for some sequence ω and for some c the inequality*

$$KS((\omega)_n|n) \geqslant n - f(n) - c,$$

holds for all n, then ω is ML-random with respect to the uniform measure.

$\lhd$ We need to prove that every non-random sequence (i.e., every sequence that belongs to the largest effectively null set) has "simple" prefixes. Note that we also need to choose the function f.

To explain how to do this, let us assume that we are given a family of intervals with total measure at most ε. Let F be the set of strings that define these intervals (i.e., the family consists of intervals Ω_x for all $x \in F$). Let us sort strings in F according to their length and for each length n consider the total measure of intervals that correspond to n-bit strings in F. Let it be $2^{-f(n)}$ (by the definition of f). Then we have $\sum_n 2^{-f(n)} \leqslant \varepsilon$. On the other hand, the set F contains $2^{n-f(n)}$

strings of length n, and each of these strings can be described (when n and other parameters of the construction are given) by $n - f(n)$ bits. This gives an upper bound for the complexity of all the strings in F. Note also that any infinite sequence that is covered by our intervals has a prefix in F.

Now we return to the proof. Consider the largest effectively null set. For each $\varepsilon > 0$ there exists its covering by interval of total length at most ε, and we can use the construction above to get the corresponding function f with $\sum_n 2^{-f(n)} \leqslant \varepsilon$. We need to combine those functions for different ε into one function f as the theorem requires. This is done as follows.

For each $c = 0, 1, 2, \ldots$ consider the covering by a family of intervals with total measure not exceeding 2^{-3c}, the corresponding set F_c of strings and the corresponding function f. Then we decrease f by $2c$ and obtain a function f_c such that

$$\sum_n 2^{-f_c(n)} < 2^{-c}$$

(we get 2^{-c} instead of 2^{-3c} since we have decreased f by $2c$). The set F_c contains $2^{n-f_c(n)-2c}$ strings of length n, and any non-random sequence has a prefix in F_c.

Then $f(n)$ is defined by the equation

$$2^{-f(n)} = \sum_c 2^{-f_c(n)}.$$

This guarantees that

$$\sum_n 2^{-f(n)} = \sum_n \sum_c 2^{-f_c(n)} = \sum_c \sum_n 2^{-f_c(n)} \leqslant \sum_c 2^{-c} \leqslant 1.$$

On the other hand, the set F_c is enumerable given c (according to the definition of an effectively null set), so any its element x of length n is determined (when n and c are known) by its ordinal number (in the enumeration of strings of length n in F_c), i.e., by $n - f_c(n) - 2c$ bits:

$$KS(x|n,c) \leqslant n - f_c(n) - 2c + O(1),$$

which implies

$$KS(x|n) \leqslant n - f_c(n) - 2c + O(\log c) < n - f(n) - c$$

for any $x \in F_c$ of length n (for large enough c).

Now let ω be any non-random sequence. As we have seen, for each c ω has a prefix in F_c. Let n be the length of this prefix. Then

$$KS((\omega)_n|n) < n - f(n) - c$$

(assuming that c is large enough), which contradicts our assumption.

However, this does not complete the proof, since we need a *computable* function f, and the set F_c is only enumerable, so we don't know when all strings of length n have been appeared, and therefore cannot compute f. To fill this gap, recall that we started with a family of intervals (that covers the largest effectively null set). In this covering we may split a large interval Ω_z into many small intervals Ω_{zt} (for all strings t of some length). This allows us to make f_c computable if we

require (without loss of generality) that the length of the intervals in the enumeration can only increase. The same argument can be applied to all f_c in parallel and makes f computable.

Finally, there is a (trivial) technical problem: the statement requires f to be integer-valued, so some rounding is needed. $\triangleright$

The two last theorems together provide a randomness criterion that uses plain complexity (and not monotone or prefix complexity). This criterion is "robust": one can replace the conditional complexity $KS((\omega)_n|n)$ by the unconditional one, $KS((\omega)_n)$, or by conditional prefix complexity, $KP((\omega)_n|n)$.

Indeed, this replacement increases complexity, therefore only Theorem 88 needs to be verified. For the prefix complexity version: we use that for any finite set A and for any its element x the inequality $KP(x|A) \leqslant \log_2 |A| + O(1)$ holds (we consider a prefix-free encoding by the strings of length $\log_2 |A|$).

The case of the unconditional plain complexity is a bit more difficult. As we do not know n, we need to describe a string $x \in F_{c,n}$ (here $F_{c,n}$ is the set of all strings $x \in F_c$ that have length n) by its ordinal number in the entire set F_c (and not by its ordinal number in $F_{c,n}$ as before). Enumerating F_c in increasing length order, we need

$$\log(|F_{c,0}| + |F_{c,1}| + \ldots + |F_{c,n}|)$$

bits for that, and everything is OK if the last term $|F_{c,n}|$ is greater than the sum of all preceding terms (in this case the increase is at most twofold). We can achieve this using the same trick as before: replacing a string by all its continuations of a greater length. Note that this is done separately for each c, so the condition c remains, but this does not matter since it gives only $O(\log c)$ additional bits.

So we get the following result:

{miller-yu-

Theorem 89 *A sequence ω is ML-random if and only if for any computable total function $f\colon \mathbb{N} \to \mathbb{N}$ such that $\sum 2^{-f(n)} < \infty$ the inequality*

$$KS((\omega)_n) \geqslant n - f(n) - O(1)$$

holds.

This criterion uses only plain unconditional complexity and is the most popular version of Miller–Yu theorem.

This criterion has a drawback: there is a quantifier over f. It can be placed differently (there exists some f that rejects all the non-random sequences, as Theorem 88 says), but still it would be nice to get rid of f completely. It is indeed possible, the price is that we have to reinsert prefix complexity into the statement:

{miller-yu-

Theorem 90 *A sequence ω is ML-random with respect to the uniform measure if and only if*

$$KS((\omega)_n) \geqslant n - KP(n) - O(1).$$

$\lhd$ If the series $\sum_n 2^{-f(n)}$ converges for a computable f, then $KP(n) \leqslant f(n) + O(1)$. Therefore the condition with prefix complexity is stronger than that in Theorem 89.

Therefore, we need to prove only the converse implication: if for every c there exists n such that

$$KS((\omega)_n) < n - KP(n) - c,$$

then ω is not ML-random. This can be done in the same way as in Theorem 87. We need only to note that the set of all strings x such that

$$KS(x) < l(x) - KP(l(x)) - c$$

(here $l(x)$ stands for the length of x) is enumerable. $\rhd$

In this theorem we can also replace $KS((\omega)_n)$ by $KS((\omega)_n|n)$.

114 Verify that this is indeed possible.

115 Show that we cannot let $f(n) = 2\log n$ in Theorem 88. [Hint: Theorem 87 says that for a random ω we have a stronger inequality $KS((\omega)_n) \geqslant n - \log n - 2\log\log n - O(1)$. Therefore, if we computably interleave random sequence with the zero sequence (and zeros are sparse enough), we get a non-random sequence such that $KS((\omega)_n) \geqslant n - 2\log n - O(1)$. Similar argument shows that we cannot get a computably convergent series $2^{-f(n)}$ for a function f that makes Theorem 88 true.]

All the results above still do not answer a very natural question: may be one can eliminate f completely and require that $KS((\omega)_n) \geqslant n - O(1)$ (similar to monotone complexity criterion)?

Of course, this would be the most natural version of the randomness criterion, so it was tried in the very beginning. Martin-Löf noticed that this approach does not work: any binary string is a substring of a random sequence, so any random sequence contains arbitrarily large groups of zeros. And if a string of length n ends with k zeros, then its complexity is at most $n - k + 2\log k + O(1)$ ($2\log k$ bits are needed for a prefix-free encoding of k and $n-k$ bits for the rest), and the difference between length and (plain) complexity is at least $k - 2\log k - O(1)$.

The following theorem (see [**?**, **?**]) gives a more precise bound for the unavoidable difference between length and complexity:

{martin-lof

Theorem 91 *There exists some c such that for any $\omega \in \Omega$ the inequality*

$$KS((\omega)_n) \leqslant n - \log n + c$$

holds for infinitely many n.

$\lhd$ For each n let us *select* $1/n$-th fraction of all strings of length n, i.e., $\lfloor 2^n/n \rfloor$ strings of length n. We want to do this in such a way that each infinite sequence has infinitely many selected prefixes (and the set of selected strings is decidable).

Why is this possible? The series $\sum 1/n$ diverges so we can split its terms into infinitely many groups, and each group has sum greater than 1. Using one group, we get one layer of Ω-covering (this means that each sequence $\omega \in \Omega$ has a prefix among the strings that correspond to that layer).

To do this, we consider the strings in the order of increasing length and select string whose prefixes are not yet selected. (There is the rounding problem since $2^n/n$ is not an integer, but it can be easily fixed.)

Every selected string of length n can be described (if n is known) by its ordinal number, and this requires $n - \log n$ bits. Therefore, the conditional complexity of this string (with condition n) is at most $n - \log n + O(1)$. Moreover, if we make a combined list of all selected strings (in the order of increasing length), the ordinal number increases by $O(1)$ factor. Indeed, the number of selected strings grows almost as a geometric sequence, and adding all selected strings of smaller lengths increases cardinality only by $O(1)$ factor. This implies the statement of Theorem 91. $\triangleright$

116 Prove that the statement of Theorem 91 is true not only for some c but for every c (including the negative ones).

[Hint: If the series $\sum 2^{-f(n)}$ diverges, we can increase a bit the function f keeping this property: there exists a function g such that $g(n) - f(n) \to \infty$ and $\sum 2^{-g(n)} = \infty$.]

117 Show that the statement of Theorem 91 (the conditional complexity version) remains true if we replace logarithm by any computable function f such that the series $\sum 2^{-f(n)}$ diverges.

Martin-Löf claims in [**?**] that the same generalization is possible for unconditional complexity (and refers to an unpublished paper for the proof). The same statement (attributed to Martin-Löf) can be found also in [**?**]. [It is not clear how to prove it.]

Note also that the statement of Theorem 87 has a slightly different form in [**?**]:

118 Prove that if a sequence ω is ML-random with respect to the uniform measure, and $f\colon \mathbb{N} \to \mathbb{N}$ is a computable total function such that the series $\sum 2^{-f(n)}$ *computably* converges, then $KS((\omega)_n|n) \geqslant n - f(n)$ for all sufficiently large n. [Hint: If a series computably converges, and the inequality is false infinitely many times, the tails of the series can be used to get coverings that have small measure.]

Another natural question: what happens if we require high complexity not for all (sufficiently long) prefixes but for infinitely many of them? In the same Martin-Löf paper **??** paper the following results are stated:

119 Prove that for almost all (with respect to the uniform measure) sequences $\omega \in \Omega$ there exists c such that $KS((\omega)_n|n) \geqslant n - c$ for infinitely many n.

[Hint: If it is not the case, then for every c there exists N such that ω-prefix of every length $n > N$ has complexity less than $n - c$. For given c and N the set of all ω with this property has measure at most 2^{-c}. As N increases, this set increases and the union over all N has measure at most 2^{-c} by continuity.]

120 If for a given sequence ω there exists a constant c such that $KS((\omega)_n|n) \geqslant n - c$ for infinitely many n, then ω is ML-random with respect to the uniform measure. [Hint: If ω is covered by some interval in a family of total measure less than 2^{-c}, then every sufficiently long prefix of ω can be described (when length is given) by its ordinal number in the set of all strings of this length covered by some interval, and this requires $2\log c + n - c$ bits.]

121 Prove that the statement of the previous problem remains true if we replace conditional complexity $KS((\omega)_n|n)$ by unconditional complexity $KS((\omega)_n)$.

[Hint: Use Problem 6 or, better, Problem 39.]

The last two problems refer to a set of measure 1 that is a subset of the set of all ML-random sequences. Its complement is a null set; if it were an effectively null set, we would get another criterion for ML-randomness. However, it is not the case.

Recently in [**?**, **?**] it was shown that this set has a natural description: it is the set of ML-random sequences relativized to oracle $\mathbf{0}'$; these sequences are sometimes called "2-random" (while ML-random sequences are called "1-random").

5.7 The random number Ω

{chaitin-om

The following theorem provides an interesting application of the randomness criterion given in the previous section. Let m be a maximal lower semicomputable semimeasure on the set of natural numbers (e.g, let $m(x)$ be equal to $2^{-KP(x)}$; we can use also the distribution on the outputs of the universal probabilistic machine, see Chapter 4). G. Chaitin suggested to consider the number

$$\Omega = \sum_n m(n)$$

(the halting probability for the universal probabilistic machine; the sum of the "least convergent" lower semicomputable series) and made the following interesting observation:

{omega-is-r

Theorem 92 *The binary representation of Ω is Martin-Löf random with respect to the uniform distribution.*

Note that the value of Ω depends of the choice of a maximal lower semicomputable semimeasure, but the statement remains true for every choice.

$\lhd$ Assume that the first n binary digits of Ω are given. They form the binary representation of a number Ω_n which is a lower bound for Ω with approximation error at most 2^{-n}. Generate lowers bounds for $m(0), m(1), m(2), \ldots$ in parallel until the sum of these lower bounds becomes greater than $\Omega_n - 2^{-n}$. This does happen at some point since the sum of the series is Ω and hence is greater than our threshold. Then make a list of all i that appear in this sum (with a non-zero lower bound for $m(i)$).

Note that this list includes all i such that $m(i) \geqslant 2 \cdot 2^{-n}$ (if some i with this property were omitted, the approximation error would exceed 2^{-n}). Therefore, all i such that $KP(i) < n - c$ (for some c that depends on the choice of function m but not on n) appear in this list. Thus, the minimal integer that is not in the list has complexity at least $n - c$. This implies that both the list itself (which determines this minimal integer) and the number Ω_n (which allows us to construct the list) have complexity at least $n - c'$ for some other c' and for all n. It remains to use the randomness criterion in its prefix complexity version (Theorems 84 and 86). $\rhd$

One can define the notion of a (Martin-Löf) random real number directly. A set X of reals is an effectively null set if there is an algorithm that for any rational $\varepsilon > 0$ enumerates a cover of X by intervals with rational endpoints and total measure (length) at most ε. A real number is Martin-Löf random (with respect to the standard measure on $\mathbb{R}$) if it does not belong to any effectively null set (=does not belong to the maximal effectively null set).

122 Prove that a real number is random (according to this definition) if and only if its binary representation is a random sequence (with respect to the uniform measure on Ω).

123 Prove that a square (sine, exponent) of a random real is a random real. [Hint: A preimage of a null set is a null set, and this argument can be effectivized.]

124 Can the sum of two random real numbers be a non-random real? [Hint: the numbers may be "dependent".]

The random number Ω (or, better to say, any Ω-number, since different maximal lower semicomputable semimeasures lead to different numbers) has interesting properties that makes it a rather special random number.

First of all, Ω is lower semicomputable. (Note that the set of lower semicomputable real numbers is countable and therefore is a null set, but — as we see — not an effectively null set.) This property of Ω has interesting corollaries:

125 Show that if α is a lower semicomputable real number, then the number $\alpha+\Omega$ is random.

126 Prove that any real number is a sum of two random real numbers. [Hint: It is enough to know that random numbers form a set of full measure.]

[Here one should also write about Solovay reducibility for reals, the maximal element etc. — but first we should understand this!] .]

[Using Ω as an oracle allows us to decide any enumerable set. What else can be said about Ω? how the Ω's for two different optimal semimeasures are related?]

The number Ω can be considered as an "infinite version" of special objects of complexity n that are considered in Theorem 15 (p. 27). Moreover, there is a direct connection between these notions.

{chaitin-fi

Theorem 93 *Let Ω_n be the binary string formed by first n bits of the binary representation of Ω. Then Ω_n has the properties described in Theorem 15; each of the objects listed there, say, $B(n)$, can be algorithmically obtained from $\Omega_{n+O(\log n)}$ and vice versa (Ω_n can be obtained from $B(n+O(\log n))$.*

$\lhd$ Let us show how knowing Ω_n we can construct an integer T greater than $B(n-O(\log n))$. Generating the better and better approximations for Ω, we stop this process when the approximations reach Ω_n (i.e., the first n bits achieve their "true" values). Let T be the number of steps before this happens; this T can be algorithmically found when Ω_n is known. Let t be any number greater than T; let us show that the complexity of t is greater than $n-O(\log n)$. Indeed, id we know t n, we may perform t approximation steps and find Ω_n, that has prefix complexity at least $n-O(1)$ (Theorem 92) and therefore plain complexity at least $n-O(\log n)$.

Reverse direction: assume that $B(n)$ and n are known. How to find $\Omega_{n-O(\log n)}$? We claim that the current approximation for Ω found after $B(n)$ steps the first $n-O(\log n)$ bits are true (i.e., they coincide with the corresponding bits in Ω). If this is not the case then there exists a threshold β that is a finite binary fraction of size $2^{n-O(\log n)}$ bits that separates the current approximation and Ω. The complexity of β is at most $n-O(\log n)$. Knowing β, we can construct a number greater

than $B(n)$; just count the steps needed to get an approximation greater than β. For a large enough constant in $O(\log n)$ we get a contradiction. $\triangleright$

Therefore, recalling Theorem 15 we see that knowing $n+O(\log n)$ bits in Ω allows us to answer any question about the termination of a program of size at most n. Since the question about the membership in any enumerable set (e.g., questions whether a given statement of size n is provable in some fixed formal theory) have this form, we can follow Chaitin and call Ω "the number of wisdom" which contains information about many important things. (Sounds rather romantic, indeed.)

5.8 Effective Hausdorff dimension

{monothaus}

The notion of Hausdorff dimension is well known in measure theory (and became popular in connection with fractals). Here is the definition. Let $\alpha > 0$ be some real number. We say that a set A is an α-null set if for any $\varepsilon > 0$ there exists a sequence of intervals I_k that cover A such that

$$\sum_k \mu(I_k)^\alpha < \varepsilon.$$

This definition assumes that A is a subset of a space where a class of subsets called "intervals" is chosen and measure of intervals is defined. We restrict ourselves to the case of the set Ω. Here intervals are the sets Ω_x (where Ω_x is the set of all infinite sequences that start with a binary string x). The measure of the interval Ω_x equals $2^{-l(x)}$.

Let us start with a few simple remarks:

(1) Any subset of an α-null set is an α-null set.

(2) For $\alpha = 1$ we get the standard definition of a null set (set of measure zero).

(3) For $\alpha > 1$ any subset $A \subset \Omega$ is an α-null set. Indeed, one can cover A by 2^n intervals that correspond to 2^n strings of length n, and the sum of their α-measures tends to 0 as $n \to \infty$.

(4) Assume that $0 < \alpha < \alpha'$. Any α-null set is then an α'-null set (note that measure $\mu(I)$ of any interval I does not exceed 1 and therefore $\mu(I)^\alpha > \mu(I)^{\alpha'}$).

127 Give a natural definition for an α-null set of reals and show that a set $A \subset [0,1]$ is an α-null set if and only if the set of binary representations of all numbers in A is an α-null set according to the definition above.

[Hint: We need to verify that the more liberal notion of an interval in $\mathbb{R}$ where we do not require any alignment, does not change the class of null sets.]

Our remarks imply that for any set $A \subset \Omega$ there exists some threshold $d \in [0,1]$ with the following property: if $\alpha > d$, the set A is an α-null set; if $\alpha < d$ it is not. (For $\alpha = d$ the set may be an α-null set or not.) This threshold is called the *Hausdorff dimension* of the set A.

128 The *Cantor set* is the subset of $[0,1]$ that remains if we take out the middle third $(1/3,2/3)$, then take out the middle thirds of two remaining segments (i.e., $(1/9,2/9)$ out of $[0,1/3]$ and $(7/9,8/9)$ out of $[2/3,1]$ etc.). Prove that Cantor set is a compact set homeomorphic to Ω and has Hausdorff dimension $\log_3 2$.

[Hint: To get an upper bound for Hausdorff dimension one may consider the "standard" intervals, i.e., the intervals that remain untouched after several steps of the Cantor set construction. To

get a lower bound we need to consider an arbitrary cover that consists of open intervals. Then we (1) replace this cover by a finite one using compactness; (2) replace open intervals by closed intervals; (3) if some interval I from the cover intersects some deleted interval J but does not contain J, make I smaller (there is no need to cover J); (4) if some interval I from the cover contains some interval J that was deleted being the middle third of some interval I', we replace I by I'; (5) having only standard intervals, we note that they correspond to the covering of the binary tree and get the desired bound for their measures.]

129 Give a natural definition of Hausdorff dimension for the subsets of $\mathbb{R}^3$. Explain why the dimension equals 3 for solids, 2 for surfaces, 1 for curves and 0 for isolated points. Show that for any $d \in [0,3]$ there is a subset of $\mathbb{R}^3$ that has dimension d.

The effective version of Hausdorff dimension is defined in a natural way. (See [**?**, **?**].) A set $A \subset \Omega$ is an *effective α-null* set (for a given $\alpha > 0$) if there exists an algorithm that for any given $\varepsilon > 0$ enumerates a set $I_0, I_1, I_2, \ldots$ of intervals that cover A such that $\sum(\mu(I_k))^\alpha < \varepsilon$. (Here μ is the uniform measure on Ω).

As in the classical case, the property is monotone (remains true if α increases or A decreases). The main difference between the classical and effective case is shown by the following theorem:

{effective-

Theorem 94 *For any rational $\alpha > 0$ there exists the largest (with respect to inclusion) effectively α-null set.*

$\lhd$ The proof goes in the same way as for effectively null (=1-null) sets (Chapter 3). The countable union of α-null set (in the classical sense) is an α-null set. In the same way the union of an enumerable family of effectively α-null sets is an α-null set. On the other hand, if α is a rational number (or even a computable real), we can enumerate all effectively α-null sets (or, better, the algorithms that serve these sets) by enumerating all algorithms and changing them when too large intervals are generated. $\rhd$

The following result (A. Khodyrev) is not used in the sequel (for the definition of Hausdorff dimension rational α are sufficient) but is interesting in its own right. Let α be an arbitrary real number.

Theorem 95 *The largest effectively α-null set exists if and only if α is lower semicomputable.*

$\lhd$ Assume that α is lower semicomputable. This means that we can generate better and better approximations from below to α but do not know their precision. If we use these approximation (instead of true α) in the requirements for the covering in the definition of an effectively α-null set, we get stronger requirements. Consider the algorithm of the previous theorem that generates coverings of the largest effectively α-null sets and let it use rational lower approximations of α instead of α. Then modify the algorithm as follows: Do not reject permanently the intervals that violate these requirements but postpone them and check again when new approximation to α arrives. If a covering satisfies the requirement for the true α, all its intervals will be printed eventually.

On the other hand, let us assume that for some α there exists the largest effectively α-null set. Consider the algorithm that generates covers for it. This algorithm can be used to obtain lower

bounds for α. Indeed, if for some rational ε the algorithm produces a finite family of intervals (at some step) and β-powers of the measures of these intervals exceed ε, this means that $\beta < \alpha$.

It remains to prove that these bounds can be arbitrarily close to α. Assume that it is not the case and all of them are less than some $\alpha' < \alpha$. In this case every effectively α-null set would be at the same time α'-null set, which is not true (there exist sets of any effective Hausdorff dimension, see below Problem 130, p. 140 $\triangleright$

The *effective Hausdorff dimension* of a set $A \subset \Omega$ is now defined as the greatest lower bound of the set of all α such that A is an effective α-null set. This number belongs to $[0,1]$ and is obviously greater than or equal to the (classical) Hausdorff dimension. (Initially the definition of effective Hausdorff dimension was given in a different way, using computable martingales; see [**?**, **?**], where the properties of effective dimension were established. See also Section **??** about computable martingales.)

We have mentioned the paradox: the property of being an effectively null set depends only on the type of its elements (whether they are random or not); it is not important "how many" elements are in the set. A similar observation can be done for Hausdorff dimension:

{dimension-

Theorem 96 *The effective Hausdorff dimension of the set equals the least upper bound of the effective Hausdorff dimensions of its elements.*

(By effective Hausdorff dimension of a point $\omega \in \Omega$ we mean the Hausdorff dimension of the singleton $\{\omega\}$.)

$\triangleleft$ Obviously the (effectively Hausdorff) dimension of a set cannot be less than the dimension of any its element. It remains to prove the converse statement: if the dimensions of all singletons formed by elements of a set A are less than some rational number r, and $r' > r$ is another rational number, then the dimension of A does not exceed r'. This is a direct corollary of the previous statement: all singletons are subsets of the largest effectively null r'-set, so A is a subset of the same set and has dimension at most r'. $\triangleright$

Therefore we need to understand only what is the (effective Hausdorff) dimension of a singleton. It turns out that it has a simple description in terms of Kolmogorov complexity.

{hdim-formu

Theorem 97 *The effective Hausdorff dimension of a singleton $\{\omega\}$, where $\omega = \omega_0\omega_1\omega_2\ldots$, is equal to*

$$\liminf_{n\to\infty} \frac{KS(\omega_0\omega_1\ldots\omega_{n-1})}{n}.$$

(The statement uses plain Kolmogorov complexity of the prefixes of ω. However, this is not important: since the difference between different complexity versions is of order $O(\log n)$ for strings of length n, and we divide the complexity by n, we get a term $O(\log n)/n$ that does not change the limit.)

$\triangleleft$ We have to prove two inequalities: one for each direction.

Assume that the $\liminf$ is less than a rational number r. We have to verify that the set $\{\omega\}$ is an effectively r'-null set for any rational $r' > r$.

For each n we consider all n-bit strings that have complexity less than rn. There are at most $O(2^{rn})$ such strings. The condition about $\liminf$ guarantees that for infinitely many n the n-bit

prefix of ω is in the corresponding list. Consider all intervals Ω_z for all z in the list (for some fixed n), and compute the sum required in the definition of an effectively r'-null set: there are $O(2^{rn})$ terms and each is $(2^{-n})^{r'} = 2^{-r'n}$, so the sum is $2^{(r-r')n}$, which gives a converging geometric series

$$\sum_n 2^{(r-r')n}.$$

Deleting an initial part of this series (considering only strings of length N or more) we make the sum arbitrarily small (when N is large enough). At the same time our assumption (about lim inf) guarantees that remaining intervals still from a covering for ω. So one inequality is proved.

Going in the other direction, assume that $\{\omega\}$ has effective dimension less than r for some rational r. Let us show that the lim inf does not exceed r.

By definition, for each rational $\varepsilon > 0$ we can generate a sequence of intervals. We know that one of them contains ω and the sum of r-th powers of the measures does not exceed ε. Let us do this for $\varepsilon = 1, 1/2, 1/4, \ldots$. In this way we get a sequence of intervals that have finite sum of r-th powers of their measures, and infinitely many of them cover ω. In other terms, there exists a computable sequence of intervals $x_0, x_1, x_2, \ldots$ such that:

- $\sum 2^{-rl(x_i)} < \infty$;
- x_i is a prefix of ω for infinitely many i.

The first statement implies that $m(i) \geqslant c2^{-rl(x_i)}$ for some c and for all i (where m is the largest semimeasure on natural numbers considered in Chapter 4). Taking the logarithms, we get the bound for prefix complexity:

$$KP(x_i) \leqslant rl(x_i) + O(1)$$

for all i. Note also that the lengths of x_i tend to infinity (since the series in convergent), than x_i is a prefix of ω for infinitely many i and that the plain complexity does not exceed the prefix one. (The definition of lim inf guarantees that if a sequence has infinitely many terms that do not exceed r, its lim inf does not exceed r.) $\triangleright$

130 Prove the following corollary: for any real $\alpha \in [0,1]$ there exists a set (an even a singleton) that has effective Hausdorff dimension α. [Hint: The complexity of an initial segment can be increased by adding random bits and decreased by adding zeros.] {any-hausdo

131 Prove that for any real $\alpha \in [0,1]$ there exists a set that has (classical) Hausdorff dimension α.

[Hint: Consider the set of all sequences that have zeros at specified places.]

132 Prove that the definition of effective Hausdorff dimension of a set A can be reformulated in the following equivalent way: there exists a computable sequence of intervals that has finite sum of r-th powers of the measures and that covers each element of A infinitely many times.

We return to the notion of effective Hausdorff dimension in Section **??** where its relation to effective martingales is explained; we show there how to translate the proof of Theorem 97 into the martingale language.

5.9 Randomness deficiency using a priori complexity

{monotdef}

The ML-randomness criterion (for a computable measure P) can be reformulated in the following way. For each string x consider the difference

$$d_P(x) = -\log_2 P(\Omega_x) - KA\,(x).$$

The sequence ω is ML-random with respect to P if this difference is bounded (by a constant) for the prefixes of ω. So we may call this difference the *randomness deficiency* of a string x (with respect to computable measure P): a sequence is random if the deficiencies of its prefixes are bounded (by a constant).

The name "randomness deficiency" is quite general and may be understood in different ways in different contexts; see, e.g., Chapter **??**. However, in this section we study the properties of the function d_P defined in the above way.

[In would be nice to analyze these definitions systematically, including Levin–Gacs definition of deficiency for infinite sequences, classes of measures etc. This could be a special section in this chapter!]

If $P(\Omega_x) = 0$ for some x, we let $d_P(x) = +\infty$.

The randomness deficiency is always non-negative (up to a $O(1)$ additive term), see Theorem 81). The randomness criterion (Theorems 83 and 85) guarantees that for ML-random sequences the deficiency of their prefixes is bounded while for non-random sequences the deficiencies tend to infinity. This implies that the intermediate situation is not possible: there in no sequence such that deficiencies of its prefixes are not bounded but do not tend to infinity. This looks rather strange, and one may ask why this happens. The following theorem provides some explanation.

{monotdef-c

Theorem 98 *Let P be a computable measure on Ω. There exists a constant c such that for any string x and for any string y that has x as a prefix the inequality*

$$d_P(y) \geqslant d_P(x) - 2\log d_P(x) - c$$

holds.

Informally speaking, any continuation of a string with high deficiency has (almost as) high deficiency. Or: a prefix of a string that has small deficiency, has (almost as) small deficiency, so the deficiency function is quasi-monotonic.

$\lhd$ For each k consider the enumerable set of all finite sequences that have deficiency greater than k. All the infinite continuations of these sequences form an open set S_k, and P-measure of this set does not exceed 2^{-k}. Now consider the measure P_k on Ω that is zero outside S_k and is equal to $2^k P$ inside S_k. That means that for any set U the value $P_k(U)$ is defined as $2^k P(U \cap S_k)$. Actually, P_k is not a measure according to our definition, since $P_k(\Omega)$ is not equal to 1. However, P_k can be considered as a lower semicomputable semimeasure, if we change it a bit and let $P_k(\Omega) = 1$ (this means that the difference between 1 and the former value of $P_k(\Omega)$ is assigned to the empty string).

Now consider the sum

$$S = \sum_k \frac{1}{2k^2} P_k.$$

It is a lower semicomputable semimeasure (the factor 2 in the denominator is used to make the sum $\sum 1/2k^2$ less than 1); again, we need to increase S so that $S(\Omega) = 1$. Then we have

$$-\log S(x) \leqslant -\log P(x) - k + 2\log k + O(1)$$

for any string x that has a prefix with deficiency greater than k. Since S does not exceed the maximal lower semicomputable semimeasure on the binary tree (up to $O(1)$-factor), we get the desired inequality.

This argument assumes that deficiency of x is finite (i.e., $P(\Omega_x) \neq 0$); if $P(\Omega_x) = 0$, then $P(\Omega_y) = 0$ for any y that has prefix x, and the deficiency of y is also infinite. ▷

Let us note one more property of the deficiency function that is an immediately corollary of its definition:

133 Prove that for every string x the deficiency of at least one of the strings $x0$ and $x1$ does not exceed the deficiency of x. (We assume that a computable measure P used in the definition of the deficiency is fixed.)

This problem shows that we can start with any string and extend it bit by bit not increasing its deficiency. The randomness criterion guarantees that in this way we get a ML-random sequence with respect to the measure used in the definition of deficiency. (Recall that we assume that $P(\Omega_x) > 0$ for any x.)

Now we use the notion of deficiency (as defined in this section) to compare randomness with respect to different measures.

Let P be a computable probability distribution on Ω and let $f\colon \Sigma \to \Sigma$ be a continuous computable mapping. Consider the image of the measure P with respect to f, i.e., a measure Q on the set Σ such that

$$Q(U) = P(f^{-1}(U))$$

for any $U \subset \Sigma$. In other terms, Q is the probability distribution of the random variable $f(\omega)$ where ω is a random variable that has distribution P. In general case the distribution Q is not concentrated on Ω and may assign positive probabilities to finite sequences; in our terminology Q may be a semimeasure (and this semimeasure is lower semicomputable), not a measure.

Let us assume, however, that it is not the case and that Q is a measure on Ω. (It is easy to see that in this case Q is a *computable* measure.)

{random-ima

Theorem 99 (**a**) *For any sequence $\omega \in \Omega$ that is ML-random with respect to measure P, its image $f(\omega)$ is an infinite sequence that is ML-random with respect to measure Q.*

(**b**) *Any sequence τ that is ML-random with respect to Q can be obtained in this way, i.e., there exists a sequence ω that is ML-random with respect to P and $f(\omega) = \tau$.*

The intuitive meaning of this theorem can be explained as follows. Assume that we have a probabilistic machine that consists of a random bit generator and an algorithm that transforms random bits into an output sequence. Assume that the random bit generator has distribution P and the transformation algorithm defines a computable mapping $f\colon \Sigma \to \Sigma$.

Which sequences can appear as the output sequences of this machine? Or, better to say, for which sequences we can believe that they appeared at the output of this machine? There are two

possible answers. First, we can look inside the machine and say that these sequences are f-images of the ML-random (with respect to P) sequences. On the other hand, we can forget about the internal mechanisms and look at the output distribution only: then we expect the output sequence to be ML-random with respect to Q. Our theorem guarantees that these two approaches lead to the same class of sequences.

$\lhd$ First, let us prove that the f-image of a P-random sequence ω is infinite. If it is not the case and $f(\omega)$ is a finite string z, consider all infinite sequences ω such that $f(\omega) = z$ (i.e., the f-preimage of the set $\Sigma_z \setminus (\Sigma_{z0} \cup \Sigma_{z1})$.

The preimage of Ω_z is an effectively open set (the union of an enumerable set of intervals), the preimage of $\Sigma_{z0} \cup \Sigma_{z1}$ is another effectively open set that is a subset of the first one. To get the contradiction, we have to prove that the preimage of the difference (=the difference of the preimages) does not contain random sequences. This is a special case of the following general statement.

Lemma. Let P be a computable measure on Ω, and let $U \subset V$ be two effectively open sets such that $P(V \setminus U) = 0$. Then $V \setminus U$ is an effectively null set (=does not contain random sequences).

Proof of the Lemma. It is enough to consider one interval I in the set V (and replace U by its intersection with I). Enumerating the intervals that form the set U we cover more and more points in I. By continuity the measure of the covered part converges to the measure of the interval I (since $V \setminus U$ has zero measure). Therefore, we can wait until the remaining part of I has measure less than ε for any given ε and find a cover of $I \setminus U$ by a (finite) family of intervals with small total measure. Lemma is proved.

To finish the proof of (a) we have to show that the image $f(\omega)$ of a P-random sequence ω cannot be an infinite but not Q-random sequence. Indeed, assume that that $f(\omega)$ is infinite but does not form an effectively Q-null set. The preimages of the intervals that cover $f(\omega)$ cover ω, and we get an effectively open set that contains ω and has small measure (recall that P-measure of the preimage of an effectively open set is equal to Q-measure of the set itself). The statement (a) is proved.

Let us now prove the statement (b) using the notion of the deficiency. Assume that the sequence τ is ML-random with respect to the measure Q. This means that the deficiencies of its prefixes are bounded (by a constant). Then we apply the following lemma that can be considered as the "finitary version" of the statement (b):

Lemma. Let u be a string such that $Q(\Omega_u) > 0$. Then there exists a string v such that u is a prefix of $f(v)$ and $d_P(v) \leqslant d_Q(u) + O(1)$.

(The constant hidden in $O(1)$ may depend on f, P and Q but not on u.)

Proof of the Lemma. Consider the preimage $F_u = f^{-1}(\Sigma_u)$ of Σ_u. This is an effectively open subset of Σ. By definition, the P-measure of the set F_u (recall that the measure P is concentrated on infinite sequences) equals $Q(\Sigma_u)$. If the deficiency $d_Q(u)$ is small, $Q(\Sigma_u)$ cannot be significantly less than the a maximal (tree) semimeasure of u.

Now consider the a priori probability (on the binary tree) of the set F_u, i.e., the probability of the event "the output of an universal probabilistic machine M belongs to F_u". This event can be rephrased as follows: the output of the machine $f \circ M$ (that applies f to the output of M) starts with u. Comparing the machine $f \circ M$ and the universal one, we conclude that a priori probability

of the set F_u can be only constant times bigger that the a priori probability of Σ_u. The latter can be only $2^{d_Q(u)}$ times bigger than $Q(\Sigma_u)$, which is equal to the P-measure of the set F_u. Therefore we get an inequality between two measures of F_u (the a priori probability A and P):

$$\frac{A(F_u)}{P(F_u)} \leqslant O(2^{d_Q(u)})$$

Since the set F_u can be represented as the union of a (possibly non-enumerable) family of disjoint intervals, we conclude that the similar inequality is true for some interval I_v in this family:

$$\frac{A(\Sigma_v)}{P(\Sigma_v)} \leqslant 2^{d_Q(u)} \cdot O(1)$$

Since $\Sigma_v \subset F_u$, we get $f(v) \succcurlyeq u$, and the inequality implies that $d_P(v) \leqslant d_Q(u) + O(1)$. Lemma is proved.

Now we continue the proof of statement (b). Let $t_n = (\tau)_n$ be the prefix of a Q-random sequence τ that has length n. The randomness criterion guarantees that Q-deficiencies of t_i are bounded. Then Lemma says that there exists a sequence of strings $v_0, v_1, \ldots$ that have bounded P-deficiencies such that $f(v_i)$ is an extension of t_i.

A standard compactness argument shows that the sequence v_i has a subsequence that either consists of identical strings or converges to some infinite sequence ω. The latter means that any (finite) prefix of ω is a prefix of all but finitely many strings in the sequence.

In the first case the sequence τ is the image of the finite string v that appears infinitely often in the sequence v_i. (This can happen for a Q-random sequence τ if this sequence, i.e., the corresponding singleton, has a positive measure; τ is computable in this case.) Then we let ω be any P-random continuation of the string v (we know that it exists, since $P(\Omega_v) > 0$).

In the second case an infinite subsequence of the sequence v_i converges to ω. Note that in this case:

(1) Any prefix of ω is a prefix of some v_i, and these strings have bounded P-deficiencies. Therefore, Theorem 98 guarantees that P-deficiencies of all prefixes of ω are bounded. So the sequence ω is ML-random with respect to P.

(2) As we have proved in part (), the sequence $f(\omega)$ is infinite.

(3) The sequence $f(\omega)$ cannot have a prefix that is not a prefix of τ. Indeed, in this case ω would have a prefix u whose image is incompatible with τ; then the string u is a prefix of almost all strings in the subsequence that converges to ω, but images of v_i have increasing common prefixes with τ.

This contradiction finishes the proof of part (b). $\triangleright$

134 Give a direct proof of (b) using the definition of an effectively null set. [Hint: For a given ε consider the family of intervals Z_ε that covers the largest effectively P-null set and has total P-measure less than ε; let F be the (closed) set of non-covered sequences. All sequences in F are random; it is enough to show that (for any ε) the complement to f-image of F can be effectively covered by intervals with small total Q-measure.

Let us call a string x "special" if the family Z_ε together with the f-preimages of all strings inconsistent with x covers the entire Ω. The set of special strings is enumerable (since every

covering of Ω has a finite subset that covers Ω due to compactness, we can effectively enumerate all special strings). If a sequence $f(\omega)$ has a special prefix x, then ω is covered by Z_ε, therefore the total Q-measure of all sequences having special prefixes is less than ε. On the other hand, any sequence τ that does not have special prefixes is an f-image of a sequence in F. Indeed, since for every k the k-bit prefix of τ is not special, there exists some sequence $\omega_k \in F$ whose image (being an infinite sequence, since $\omega_k \in F$) coincides with τ in the first k bits. Consider a convergent subsequence of a sequence $\omega_1, \omega_2, \ldots$; its limit ω belongs to F (since F is a closed set), so $f(\omega)$ is infinite and coincides with τ (continuity).]

135 Prove a statement that can be considered as a finitary version of the statement (a) of the last theorem: if u and v are binary strings such that $u \preccurlyeq f(v)$, then

$$d_Q(u) \leqslant d_P(v) + 2\log d_P(v) + O(1).$$

[Hint: the set of sequences having large Q-deficiency can be covered by a set of small Q-measure; therefore its preimages can be covered by a set of small P-measure and have large P-deficiency. Note that this statement is a generalization of Theorem 98.]

Theorem 99 has some (rather surprising) applications. Here are two examples:

136 Let ω be a ML-random sequence according to the Bernoulli distribution (independent coin tosses) where 1 has probability $1/3$. Prove that there exists a sequence ω' that is random with respect to the uniform distribution (1 has probability $1/2$) and can be obtained from ω by replacing some zeros by ones. [Hint: Consider a ML-random sequence of independent random reals uniformly distributed in $[0,1]$, or, better to say, the random sequence of bits placed in two-dimensional table where (infinite) rows are considered as infinite binary fractions. Then convert this sequence into bit sequence using threshold $2/3$. Theorem 99 guarantees that we get a ML-random sequence with respect to the $1/3$-Bernoulli distribution and that any ML-random sequence with respect to this distribution can be obtained in this way. Then we can change threshold to $1/2$.]

137 Consider a computable distribution on pairs of sequences (i.e., on the set $\Omega \times \Omega$) and the corresponding notion of ML-randomness on pairs. Show that if $\langle \xi, \eta \rangle$ is a random pair, then ξ is a ML-random sequence with respect to the distribution that is a projection of the distribution on $\Omega \times \Omega$ onto the first coordinate, and that any ML-random sequence is a first component of some random pair.

An important special case: a distribution on $\Omega \times \Omega$ is a product of two distributions, i.e., the components of the random pair are independent. In this case a stronger claim is true (known as van Lambalgen theorem, see [**?**]).

[Where exactly is this is Lambalgen's thesis? There is a lot of probably related material, but is this statement explicit somewhere?]

Let P and Q be two computable distributions on Ω. Consider the product $P \times Q$ which is a computable distribution on $\Omega \times \Omega$ (this space is isomorphic to Ω and the definitions of randomness can be easily extended onto it).

Theorem 100 *A pair of sequences $\langle \xi, \eta \rangle$ is ML-random with respect to the distribution $P \times Q$ if and only if the following conditions are both true:*

(1) ξ *is ML-random with respect to* P*;*
(2) η *is ML-random relative to* ξ (*with oracle* ξ) *with respect to* Q.

Speaking about relativized randomness, we mean that the algorithm that (given $\varepsilon > 0$) enumerates the intervals in the covering now has access to ξ as an oracle (so we get more enumerable sets, more non-random sequences and less random sequences). The use of oracle is crucial, since a pair $\langle \xi, \xi \rangle$ is rarely random with respect to $P \times P$ even when ξ is random with respect to P.

Note also that the conditions (1) and (2) are not symmetric with respect to ξ and η. Theorem implies that the condition (1) can be replaced by a stronger requirement: ξ is random relative to η. However, the non-symmetric version looks more natural. It can be read as: "to produce a random pair, first choose a random ξ and then choose a random η knowing ξ (random relative to ξ)".

$\lhd$ Let us prove first that conditions (1) and (2) are true for a random pair $\langle \xi, \eta \rangle$.

(1) If the sequence ξ is not random and may be covered by intervals of small measure, then the same intervals multiplied by Ω (along the second coordinate) become rectangles (products of intervals along both coordinates) that cover $\langle \xi, \eta \rangle$ and have small measure.

(2) Assuming that η is not random with oracle ξ. Then for each ε we can (using ξ as an oracle) enumerate intervals that cover η and have small Q-measure. This enumeration process can be run with any oracle and it will generates some intervals using finite amount of information about the oracle.

Therefore, we get (for a given $\varepsilon > 0$) a family of rectangles that is enumerable (without oracle) and has the following property: if the first coordinate is fixed to be ξ, the rectangles become a family of intervals with total Q-measure at most ε. This family can be easily converted into a family of rectangles for which all vertical sections (not only ξ-section) have the same property and all the sections where this inequality was true before the conversion remain untouched. This contradicts to the randomness of $\langle \xi, \eta \rangle$.

Now let us prove that if the pair $\langle \xi, \eta \rangle$ is not random, then one of the conditions (1) and (2) is false. Assume $\langle \xi, \eta \rangle$ is not random. Let U be the union of an enumerable family of rectangles in $\Omega \times \Omega$ of measure at most ε that covers $\langle \xi, \eta \rangle$. For each fixed value of the first coordinate x let U_x denote the x-section of U, i.e., the set $\{y | \langle x, y \rangle \in U\}$. Consider the values of x such that that Q-measure of U_x is greater than $\sqrt{\varepsilon}$. We get a set of P-measure at most $\sqrt{\varepsilon}$ that is an union of an enumerable family of intervals.

There are two possibilities: either ξ is covered by an enumerable family of intervals having total P-measure at most $\sqrt{\varepsilon}$ that we have constructed, or $\langle \xi, \eta \rangle$ is covered by a family V of rectangles such that the Q-measure of V_ξ does not exceed $\sqrt{\varepsilon}$. (Other section can have bigger measure, this does not matter.) In the second case η is covered by a ξ-enumerable family of intervals of total measure at most $\sqrt{\varepsilon}$.

We would like to apply this argument for every ε and conclude that either ξ is not random or η is not random with oracle ξ. The first conclusion can be drawn if for every ε the first possibility happens; the second one if the second possibility happens for every ε. But what should we do if both cases happen for different values of ε?

The following simple trick helps. For every $k = 1, 2, 3, \ldots$ we perform this construction for $\varepsilon = 2^{-2k}$. Then for each k we get a family $V(k)$ of intervals (along the first coordinate) that have total P-measure at most $2^{-k} = \sqrt{2^{-2k}}$. Now the two possibilities are:

(a) the family $V(k)$ covers ξ for infinitely many k;

(b) for sufficiently large k the family $V(k)$ does not cover ξ.

If (a) happens, for each K the union of $V(k)$ for all $k \geqslant K$ gives us an enumerable covering of ξ that has total measure $2 \cdot 2^{-K}$, so ξ is not random.

If (b) happens, then for each k greater than some K one can ξ-enumerate a family of intervals that covers η and has total Q-measure at most 2^{-k}, so η is not ξ-random. (We do not know the value of K, but this does not matter.) $\triangleright$

[One would like to generalize this theorem for the case of dependent variables, but first we need to develop theory of conditional probabilities. Or, better, to learn it.]

Let us return to the question that we have already discussed: A probabilistic machine is given; which sequences seem to be the plausible outputs of this machine (or, better to say, for which sequences we agree to believe that they are generated by this machine)? This question is meaningful for any machine, even for the machine that generates finite sequences with positive probability.

More formally, consider a computable probabilistic distribution P on the set Ω and computable continuous mapping $f : \Sigma \to \Sigma$. Together they generate some output distribution Q that is the image of P under f. Now we do not assume that Q is concentrated on infinite sequences, so we get an lower semicomputable semimeasure Q which is not necessarily a measure.

On the other hand, we can consider the images (under f) of sequences that are ML-random with respect to P. The interesting question is: Does this set depend only on the distribution Q? (This is true for measures, when this set coincides with the set of Q-random sequences.) The natural conjecture: this set coincides with the set of sequences such that the ratio A/Q (where A is the a priori probability on the binary tree) is bounded for their prefixes.

The authors don't know whether this is true. However, one may note that this conjecture implies the following theorem (proved in [**?**, **?**]):

{gacs-reduc

Theorem 101 *Let α be an arbitrary sequence of zeros and ones. Then there exists a sequence ω that is ML-random with respect to the uniform measure, and a computable mapping $f : \Sigma \to \Sigma$ such that $f(\omega) = \alpha$.*

Using the terminology of recursion theory, this statement can be reformulated as follows: any sequence of zeros and ones is Turing-reducible to some ML-random sequence with respect to the uniform measure. (We have already mentioned this result on p. 111.)

This theorem is an immediate corollary of the conjecture above: indeed, if f is the universal machine, then Q is (up to $O(1)$ factor) the a priori probability on the binary tree, and the ratio A/Q is bounded everywhere. However, we cannot use our previous arguments in this case (we may find the preimages of t_i that have small deficiency and even find their condensation point, but now the image of this condensation point can be finite.) So we need another construction.

$\triangleleft$ We prove a bit stronger statement and construct a computable continuous mapping f (the same for all α) such that the image $f(R)$ (where R is the set of all ML-random sequences with respect to the uniform measures) equals Ω.

Moreover, for any effectively open set U (i.e., the union of an enumerable family of intervals) of sufficiently small measure we will construct a computable mapping f such that $f(\Omega \setminus U)$ covers

entire Ω. Applying this construction to an effectively open set of small measure that covers R we get the result.

We enumerate intervals in U one at a time, and construct f watching the growth of U. At each step we ensure that the image of the complement of U (more precisely, the current approximation to U) is the entire Ω.

Initially the approximation to U is empty, so this goal is trivial: we may let f be the identity mapping. However, we should have in mind that later some parts of Ω become covered by U and therefore should be replaced by something else, so we need to reserve some space for future use. This reserve is easy to provide since the total measure of U is small, so most of the space will remain usable. However, we should be careful: it is not enough that at every stage of the construction a given sequence α has a preimage outside U; there must be a sequence ω that is outside U at all steps and is (ultimately) mapped to α. We achieve this goal, since the sequence of preimages has a limit (see below).

More specifically we choose some positive integers $k_0, k_1, k_2, \ldots$ and split the sequences in Ω into blocks of lengths $k_0, k_1, k_2, \ldots$. In other terms, we define f on the tree whose root has 2^{k_0} sons, each of these sons has 2^{k_1} sons etc. Vertices of these tree are strings of lengths $0, k_0, k_0+k_1, k_0+k_1+k_2$ etc. Formally speaking, the algorithm that implements the transformation f will read input blockwise, not bit by bit.

We may assume without loss of generality that the intervals of set U are also block-aligned (represent vertices in the subtree): U is an union of intervals Ω_x where each x has length $k_0+\ldots+k_i$ for some i. (This can be achieved by splitting every interval into smaller intervals; the total length remains unchanged.)

In this tree (with large branching factor) we select a binary subtree that will be mapped onto a standard binary subtree by f. To do this, we select two among 2^{k_0} sons of the root, X_0 and X_1 (say, the first two in the lexicographic ordering) and map them onto 0 and 1 respectively. This means that for any sequence ω that starts with X_0 (resp. X_1) the first bit of $f(\omega)$ equals 0 (resp. 1). For other strings ω (whose first block is neither X_0 nor X_1) the value $f(\omega)$ is not yet defined. Then for each string X_0 and X_1 we consider two its sons (say, the first two in the lexicographic ordering). We get four strings of length k_0+k_1 that we denote $X_{00}, X_{01}, X_{10}, X_{11}$; they are mapped onto strings 00, 01, 10 and 11 respectively, and so on. Formally, for each binary string u we consider a string X_u that consists of $l(u)$ blocks and is mapped onto u. If u is a prefix of v, then X_u is a prefix of X_v; if u is incompatible with v, then X_u is incompatible with X_v.

In this way we get a computable mapping that maps the binary subtree of the large tree onto entire Ω. If U is empty, this is the end of construction. If not, we have to react when a new interval in U covers some branches of our binary subtree making them unusable. The reaction is that we reroute our subtree in the (still) free zone and extend f on this zone, so the image of the current subtree is still entire Ω.

To describe this construction in more details, we need to introduce a notion of a "bad" vertex (i.e., a bad string of length 0, k_0, k_0+k_1, etc.) This notion changes during the process: the more intervals of U appear, the more bad vertices they create. A vertex x is bad in either of two cases:

• x belongs to some blacklisted interval (i.e., for one of intervals Ω_z in U the string z is a prefix of x);

• all the sons of x except (may be) one are bad.

As usual, this inductive definition says that we consider the minimal set of vertices that satisfies these two properties. This definition guarantees that any "good" (=not bad) vertex has at least two good sons. This property allows us to embed full binary tree in the subtree of good vertices.

Note that the sons of a bad vertices are also bad according to this definition.

Let us compute what minimal part of Ω needs to be in U to make the tree root a bad vertex. The root is bad if all its sons except one are bad. So we need a fraction

$$\left(1-\frac{1}{2^{k_0}}\right)$$

of bad vertices on the first level (the sons of the root). To get that many bad vertices, we need at least fraction

$$\left(1-\frac{1}{2^{k_0}}\right)\cdot\left(1-\frac{1}{2^{k_1}}\right)$$

of bad vertices at the second level. At some level all the bad vertices are inside U-intervals (since at any time we have a finitely many intervals in U). Therefore we get the following statement: *if the total measure of blacklisted intervals is less than the infinite product*

$$\left(1-\frac{1}{2^{k_0}}\right)\cdot\left(1-\frac{1}{2^{k_1}}\right)\cdot\left(1-\frac{1}{2^{k_2}}\right)\cdot\ldots,$$

that the root remains good at any stage of the construction.

Now the choice of k_i is clear: the product above should be positive which is equivalent to the convergence of the series $\sum 2^{-k_i}$. For example, we may let $k_i = 2\log i$ (for $i \geqslant 2$).

The set of bad vertices increases as time increases. If it does not damage the current binary subtree, then we leave the subtree unchanged. But if it does and some branch of the subtree becomes bad, then we substitute this bad vertex by its good brother (which exists since the father is good), and then grow the subtree from this good brother. Formally speaking, we modify the tree starting from the root replacing bad vertices by their good brothers and choosing good sons for new vertices.

While changing the tree we also extend the mapping f (leaving it unchanged where it was already defined). The process is effective so it is easy to check that f is computable.

It remains to prove that (for f constructed in this way) every sequence $\alpha \in \Omega$ has an f-preimage outside U. By definition, at any stage t of the construction there exists f-preimage ω_t that is not covered by the already discovered part of U. Moreover, as t increases, the points ω_t converge to some limit sequence ω (we prove the stabilization property at level i by induction over i; note that the number of possible changes on level i is bounded by 2^{k_i}). It remains to verify that ω does not belong to U and that $f(\omega) = \alpha$.

By way of contradiction, assume that ω is in U. Then ω belongs to some interval that is discovered on some step. After that the sequences ω_t do not belong to this interval, hence ω does not belong either.

Finally, let us verify that $f(\omega) = \alpha$. Let z be an arbitrary finite prefix of α; we have to show that $f(\omega)$ starts with z. Let k be the length of z. At every stage t there exists a k-block string in the

binary subtree that is mapped to z. When t increases, this string ultimately achieves its final value and therefore ω has a prefix that guarantees that $f(\omega)$ starts with z. $\triangleright$

138 Using this argument, prove that for any sequence α there exists a ML-random sequence ω such that α is Turing-reducible to ω and this reduction needs only $2n\log n$-bit prefix of ω to generate n-bit prefix of α.

[It seems that this bound can be improved (according to Laurent); one of the possibilities is to split α into blocks, too — but it is not clear whether this gives the best known result in this direction.]

One may speculate about the "meaning" of this theorem as follows: For any sequence α we can *a posteriori* explain how it could appear during an experiment: for a random ω this is the philosophical assumption, and the transformation f is computable and therefore can be implemented.

6 General scheme for complexities

{class}

6.1 Decision complexity

{class-deci

We started with plain Kolmogorov complexity KS and then considered also prefix complexity KP and monotone complexity KM. All three complexities were defined in terms of shortest descriptions, but the notion of description was different in each case. For plain complexity the description modes (decompressors) were just computable functions, for prefix complexity the description modes were computable continuous mappings of type $\Sigma \to \mathbb{N}_\perp$, for monotone complexity the description modes were computable continuous mappings of type $\Sigma \to \Sigma$.

To be uniform, we may use computable continuous mappings of type $\mathbb{N}_\perp \to \mathbb{N}_\perp$ for plain complexity. Recall that topology on the set $\mathbb{N}_\perp$ (and the set itself) was introduced in Section 4.4.3 (p. 79). It is easy to see that there are two possibilities for a continuous mapping $f\colon \mathbb{N}_\perp \to \mathbb{N}_\perp$: either $f(\perp)$ is some natural number (and not $\perp$), and the mapping is a constant one, or $f(\perp) = \perp$ and the values $f(n)$ for natural n can be arbitrary. The mapping of the second type are in a natural one-to-one correspondence with partial functions of type $\mathbb{N} \to \mathbb{N}$ if we use $\perp$ as a replacement for an undefined value. As before, computability is defined in the following natural way: the mapping $f\colon \mathbb{N}_\perp \to \mathbb{N}_\perp$ is *computable* if the set of pairs $\langle x,y\rangle$ such that $y \preccurlyeq f(x)$ is enumerable. All the constant mappings are computable, and for non-constant ones computability means that corresponding partial function is computable. (Recall that a partial function of type $\mathbb{N} \to \mathbb{N}$ is computable if and only if its graph is enumerable.)

So using this "new" definition of a description mode (decompressor) as a computable continuous mapping of type $\mathbb{N}_\perp \to \mathbb{N}_\perp$ we get the same plain complexity. Indeed, we add constant functions that map everything, including the element $\perp$, to some constant c, but they do not change complexity more than by $O(1)$. (A pedantic reader will stress that the function that maps everything to c should not be identified with the function that corresponds to a total function $\mathbb{N} \to \mathbb{N}$ that maps everything to c since the latter one still maps $\perp$ to $\perp$.)

All this formalism, however, is used only as a motivation for the following scheme that explains the origin of the complexities considered (see Figure 13): Each of the three complexities is obtained

	$\mathbb{N}_\perp$	Σ
$\mathbb{N}_\perp$	KS	?
Σ	KP	KM

Figure 13: KS, KP and KP revisited.

{class-1}

when we consider computable continuous mappings of the description space into the object space as description modes (decompressors).

This table has an empty cell; for this cell the description modes are computable continuous mappings of type $\mathbb{N}_\perp \to \Sigma$. Let us consider the corresponding definition in more details; we call

this complexity *decision complexity* and denote it by KR (the notation KD were used too, but now KD is used for the so-called "distinguishing complexity" so we use KR for decision complexity to avoid confusion).

Let us define decision complexity using some class of machines. Consider a machine that gets a binary string as an input (and some end-marker is written on the tape, so the machine knows where the input ends) and prints bits on the output tape (one by one). The machine is not obliged to stop, so for any input string x we obtain a finite or infinite bit sequence as machine's output. (If the output sequence is infinite, it obviously is computable.)

Any machine of the described type defines a mapping of the set of all binary strings (that can be identified with the natural numbers in $\mathbb{N}_\perp$) into a set Σ of all finite and infinite sequences. If M is a machine of this type, the complexity $KR_M(x)$ of a string x (with respect to decompressor M) is defined as the minimal length of a string y such that $M(y)$ (the output sequence for input y) starts with x.

139 Check that there exists an optimal decompressor M in the described class of decompressors (i.e., the decompressor M that leads to smallest KR_M up to $O(1)$ additive term).

140 Give the definition of computable continuous mappings $\mathbb{N}_\perp \to \Sigma$. What is the difference between this definition and the class of the machines described above and why it is not important for the definition of complexity? [Hint: a continuous mapping can map $\perp$ into some non-empty string.]

Therefore we can fill the empty cell in our table (Figure 14):

	$\mathbb{N}_\perp$	Σ
$\mathbb{N}_\perp$	KS	KR
Σ	KP	KM

Figure 14: Four complexities.

{class-2}

The following theorem lists the main properties of the decision complexity:

{decision-c

Theorem 102 (**a**) *If a string x is a prefix of a string y, then $KR(x) \leqslant KR(y)$.*

(**b**) *The complexity of prefixes of a sequence $\omega \in \Omega$ (which is a monotone function of the prefix length) is bounded (by a constant) if and only if the sequence ω is computable. (The limit of the complexity of prefixes may be called the decision complexity of the sequence ω. This complexity is finite for computable sequences and infinite for non-computable ones.)*

(**c**) $KR(x) \leqslant KS(x) + O(1)$ *for any string x.*

(**d**) $KR(x) \leqslant KM(x) + O(1)$ *for any string x.*

(**e**) $KM(x) \leqslant KR(x) + O(\log KR(x))$ *for any string x.*

(**f**) $KS(x|l(x)) \leqslant KR(x) + O(1)$ *for any string x.*

(**g**) *If $f\colon \Sigma \to \Sigma$ is a computable continuous mapping, then $KR(f(x)) \leqslant KR(x) + O(1)$ (the constant in $O(1)$ may depend on f but not on x).*

(**h**) *If $f\colon \Sigma \to \mathbb{N}_\perp$ is a computable continuous mapping, then $KS(f(x)) \leqslant KR(x)) + O(1)$ (the constant in $O(1)$ may depend on f but not on x).*

(**i**) *If $f\colon \mathbb{N}_\perp \to \Sigma$ is a computable continuous mapping, than $KR(f(x)) \leqslant KS(x) + O(1)$ (the constant in $O(1)$ may depend on f but not on x).*

(**j**) *Any prefix-free set of strings* (*none is a prefix of another one*) *that have decision complexity less than n, has cardinality less than 2^n.*

(**k**) *The function KR is upper semicomputable* (*enumerable from above*).

(**l**) *The function KR is the smallest* (*up to a constant*) *function satisfying last two conditions: if some function K is upper semicomputable and for every n the cardinality of every prefix-free set of strings x such that $K(x) < n$ for all elements of this set, is $O(2^n)$, then $KR(x) \leqslant K(x) + O(1)$.*

(**m**) *$KR(x) \leqslant KA(x) + O(1)$ for all strings x.*

$\lhd$ (a) An immediate corollary of the definition (description of a string is at the same time description of any its prefix).

(b) Assume that sequence ω is computable. Consider the machine that ignores its input and prints ω bit by bit, as a decompressor (description mode). All prefixes of ω have zero complexity with respect to this decompressor (since the empty strings is their description), and therefore they have $O(1)$ complexity (with respect to optimal decompressor), On the other hand, if the complexities of all prefixes of ω are bounded, some string has to be a description of infinitely many prefixes, therefore ω is computable.

(c) Any partial computable function whose arguments and values are binary strings can be considered as KR-decompressor (don't output anything before the computation is finished, then print the result bit by bit).

(d) Any continuous computable mapping $\Sigma \to \Sigma$ can be considered as KR-decompressor (after restriction on finite strings; we may say that we type the input string on the keyboard of a robust machine just after the computation starts and do not touch the keyboard anymore).

(e) Let $R : \mathbb{N} \to \Sigma$ be an optimal decompressor used in the definition of decision complexity. Consider a computable mapping $\hat{R} : \Sigma \to \Sigma$ defined as follows: $\hat{R}(\hat{x}u) = R(x)$, where $\hat{x}$ is a self-delimiting encoding of x (say, the x itself prepended by the binary encoding of $l(x)$ with duplicated bits and the separator 01) and u is any string (needed to ensure the monotonicity).

(f) Let again $R : \mathbb{N} \to \Sigma$ be an optimal KR-decompressor. Define the conditional decompressor S by letting $S(y,n)$ be the first n bits of the sequence $R(y)$ (if n exceeds the length of $R(y)$, then $S(y,n)$ is undefined).

(g) Consider a new KR-decompressor that is a composition of the optimal KR-decompressor and the mapping f and compare this new decompressor with the optimal one.

(h) Consider the composition of an optimal KR-decompressor and f as a KS-decompressor.

(i) Consider the composition of an optimal KS-decompressor and f as an KR-decompressor.

(j) Two inconsistent strings cannot share a description (since in this case they would be prefixes of some sequence, and the shorter string would be a prefix of the longer one). If all elements of a prefix-free set of strings have complexity less than n, then their descriptions are different strings of length less than n, and there exist less than 2^n such strings.

(k) Applying in parallel the optimal description mode to all strings, we get improving upper bounds for KR (finding new descriptions from time to time) that converge to the KR.

(l) This is a first interesting claim in this theorem (up to now we had only simple variation on known themes).

Let K be a function that satisfies (i) and (iii). Aiding a constant to K, we may assume without loss of generality that there are at most 2^n pairwise inconsistent strings x such that $K(x) < n$.

We construct a description mode that gives every string x such that $K(x) < n$ a description of length exactly n. This is done independently (and in parallel) for each n. Namely, we watch the decreasing upper bounds for K and fill the (increasing) list of strings x such that $K(x) < n$. Consider a subtree of a full binary tree that is formed by the strings in the list and all their prefixes. This is a growing subtree that has at any time at most 2^n leaves. (Indeed, the leaves are inconsistent strings x such that $K(x) < n$.) Let us attach a label to each leaf; this label is a string of length n. When the subtree grows by adding some new string, this string either extends one of the leaves (no more a leaf) or creates a new branch (being attached to some internal node). In the first case the new string is a leaf, and this leaf keeps the label of the superseded one. In the second case we provide a new label for the new leaf (which is possible since we have less than 2^n leaves).

Let us fix a label and look what happens with leaves carrying this label. Initially the label is unused. It is possible that the label remains unused forever (we do not need that many labels), but if it is not the case, the label is attached to some leaf and then moves up the tree (next position is a continuation of the previous one). So this label marks some branch of the tree (finite or infinite sequence of zeros and ones). In this way we get a function that maps strings of length n (i.e., labels) to Σ (the strings that are not labels are mapped to Λ, the empty sequence).

Combining these mappings for all n, we get a KR-description mode that provides complexity at most n for all strings x such that $K(x) < n$, just as we claimed.

(m) If x_i are pairwise inconsistent binary strings, then $\sum 2^{-KA(x_i)} \leqslant 1$ (since $2^{-KA(x_i)}$ equals a priori probability of the set Σ_{x_i} and these sets are disjoint). Therefore we have at most 2^n strings such that $KA(x_i) < n$ and may refer to the previous statement. $\triangleright$

141 Prove that $KR(x)$ can be defined as follows: for any computable function S of two arguments (the first is a binary string, the second is a natural number; values of f are zeros and ones) let $KR_S(x)$ for a string $x = x_0 \ldots x_{n-1}$ as the minimal length of the string y such that $S(y,i) = x_i$ for all $i = 0, 1, \ldots, n-1$. Then we choose an optimal function among all function of this class, and it defines decision complexity.

142 Show that the decision complexity of a string x equals (up to $O(1)$) the minimal value of $KS(p)$ for all programs p (in a given programming language, say, `Pascal`) that ignore their input and output the string x or any its continuation.

[If we replace here KS by KP, we get in the similar way an upper bound for monotone complexity. Will it be tight? Probably not, but a specific example is needed.]

6.2 Comparing complexities

{classcomp}

There are four complexities in our table (two options for the space of objects are combined with two options for the space of descriptions). The following diagram (Figure 15) shows the inequalities

between them (up to $O(1)$ additive term):

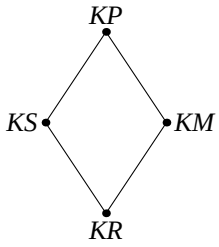

Figure 15: Inequalities between complexities. {class-3}

Some people would like to avoid references to topological notions like continuous mappings, though these notions are quite relevant here as the theory of abstract data types (Dana Scott lattices and related notion of f_0-spaces in the sense of Ershov), see [**?**]. These reader will appreciate the following simplified construction that is still enough to define the four complexities in the table.

Consider the set $\Xi = \mathbb{B}^*$ of all binary strings and two binary relations: $x = y$ means that strings x and y are equal; $x \asymp y$ means that x and y are consistent (one is a prefix of the other one). Let α and β be one of these two relations (so there are four combinations for the pair α, β).

A set $S \subset \Xi \times \Xi$ is called α-β-regular if the following condition is true for any strings x_1, x_2, y_1, y_2:

$$(x_1, y_1) \in S, (x_2 y_2) \in S, x_1 \alpha x_2 \Rightarrow y_1 \beta y_2$$

For example, $=$-$=$-regular binary relations are just graphs of functions.

143 (a) Show that every $\asymp$-$=$-regular relation determines a continuous mapping of type $\Sigma \to \mathbb{N}_\perp$. {computable

(b) Show that every $\asymp$-$\asymp$-regular relation determines a continuous mapping of type $\Sigma \to \Sigma$.

(c) Show that every $=$-$\asymp$-regular relation determines a continuous mapping of type $\mathbb{N}_\perp \to \Sigma$.

Now by α-β-*description mode* we mean an enumerable α-β-regular binary relation on $\Xi \times \Xi$. For any description mode S we define the complexity function K_S: let $K_S(x)$ be the minimal length of a description of x. i.e., the minimal value of $l(y)$ for all y such that $\langle y, x \rangle \in S$.

Theorem 103 *For any of the four combinations $\alpha, \beta \in \{=, \asymp\}$ there exists an optimal α-β-description mode* (*that provides minimal complexity function up to $O(1)$*) *and the corresponding complexity is one of the four known complexities KS, KP, KM, KR.*

$\lhd$ In all four cases enumerable α-β-regular relations correspond to computable continuous mapping of the corresponding sets (see Problem 143), that gives the same complexity function, and vice versa. $\rhd$

So we can provide new labels for rows and columns of our table (Figure 16):

144 Show how one can define for pairs of strings:

(a) monotone complexity (using computable continuous mappings $\Sigma \to \Sigma \times \Sigma$ as decompressors; such mappings are in one-to-one correspondence with pairs of computable mappings $\Sigma \to \Sigma$);

	$=$	$\asymp$
$=$	KS	KR
$\asymp$	KP	KM

Figure 16: α-β-complexities {class-4}

(b) a priori probability (using probabilistic machines that have two output tapes where bits are printed sequentially);

(c) decision complexity (using computable continuous mappings $\mathbb{N}_\perp \times \Sigma \times \Sigma$).

[There is almost no information about these notions: for example, is it true that $KM(x,y) \leqslant l(x)+l(y)$?]

Another classification scheme that goes back to [**?**]) defines each version of complexity as the smallest upper semicomputable function in some class (of functions that satisfy some restrictions). We have already considered these restrictions, so we just collect the results obtained and give the conditions for each complexity version:

- the number of strings x such that $K(x) < n$ is $O(2^n)$ (plain complexity KS, Theorem 8, p. 21);
- the series $\sum_x 2^{-K(x)}$ converges (prefix complexity KP, Theorem 56, p. 89);
- any prefix-free set of strings x such that $K(x) < n$ has $O(2^n)$ elements (decision complexity KR, Theorem 102, p. 152);
- $\sum_{x\in X} 2^{-K(x)} \leqslant 1$ for any prefix-free set X of binary strings (a priori complexity KA, Theorem 73, p. 116).

These scheme gives the same four complexities with one important exception: we get a priori complexity instead of monotone complexity. (There is no problem with prefix complexity, since it coincides with the negative logarithm of the largest lower semicomputable semimeasure on $\mathbb{N}$.)

Combining these two quadrilaterals, we get a pentagon: (Figure 17):

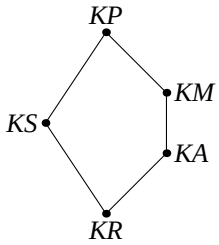

Figure 17: {class-5}

Let us recall basic results that relate complexities in this pentagon. First, all five complexities differ at most by $O(\log n)$ for strings of length n. Indeed, $KP(x) \leqslant KS(x) + O(\log KS(x))$ (Theorem 59, p. 92). On the other hand, $KS(x) \leqslant KS(x|l(x)) + KS(l(x)) \leqslant KR(x) + O(\log n)$. So the two most distant complexities in the pentagon (the upper one and the lower one) differ at most by $O(\log n)$ for strings of length n.

A more complicated picture arises if we want to bound the difference between two complexities in terms of the complexities itself, not the length (note that complexity can be much less than length). This is indeed possible for two lines that go in the north-east directions:

$$KP(x) \leqslant KS(x) + O(\log KS(x))$$

(see Theorem 59) and

$$KM(x) \leqslant KR(x) + O(\log KR(x))$$

(Theorem 102). (The similar inequality with KA instead of KM follows, as we have already mentioned in Problem 110, p. 123.) For "north-west" lines the situation is different: Indeed, KM and KR are bounded for prefixes of a computable sequence (e.g., for strings that contain only zeros) while KS and KP are not (the string of n zeros has the same complexity as the integer n, and can be of order $\log n$ for some n). n). We have already discussed this question in Theorem 79 and noted that the difference between KP and KM can be of order $\log n$ in both directions (for infinitely many n and for some x of length n). We also noted (without proof) Gacs theorem that says that the difference between KM and KA is not bounded (Theorem 80).

There are rather subtle results about relations between different complexities. For example, none of the results above guarantees that the difference between $KP(x)$ and $KS(x)$ tends to infinity as x goes to infinity (we consider here x as a natural number). This is indeed true, as the following theorem shows; this theorem was proved in R. Solovay manuscript written in 1970ies and not published (though circulated among experts).

{solovay-bo

Theorem 104

$$KS(x) \leqslant KP(x) - KP(KP(x)) + KP(KP(KP(x))) + O(1)$$

$\lhd$ As we have seen (Theorem 58, p. 91), the logarithm of the number of strings x such that $KP(x) < n$ does not exceed $n - KP(n) +$ for some c and for all n. We can enumerated all these strings (given n), so each of them can be reconstructed if we know n and the ordinal number of the string in the enumeration. This ordinal number can be written as a sequence of exactly $n - KP(n) + c$ bits (we add left zeros when needed). So when this ordinal number is given, we know already $n - KP(n)$ (the c is a constant, so we ignore it), and to restore n we need only to know $KP(n)$ that can be described by a self-delimiting code of length $KP(KP(n))$. So we concatenate this self-delimiting code and the ordinal number of x in the enumeration, and in this way we get (for any x such that $KP(x) < n$) the description of x that has length $KP(KP(n)) + n - KP(n) + c$. Therefore $KS(x) \leqslant n - KP(n) + KP(KP(n)) + O(1)$. Theorem is proved. $\rhd$

[Other results of Solovay should be added.

Any results about KM vs. KA?

Muchnik result about two sequences x_i and y_i such that $KS(x_i) - KS(y_i) \to +\infty$ but $KP(x_i) - KP(y_i) \to -\infty$.

Miller KS vs KP here?

Monotone complexity of pairs: is it true that $KM(x,y) \leqslant KM(x) + KM(y)$?

A priori complexity for pairs differs from monotone complexity?]

6.3 Conditional complexities

{classcond}

We have already considered several versions of conditional complexity (of one string relative to the other one). In Section 2.2 we have defined the conditional complexity $KS(x|y)$ as the minimal length of a string p that describes x when y is given, i.e., a string p such that $S(p,y) = x$. Here S is the conditional decompressor that is optimal in the class of all partial computable binary functions.

In Section 4.7 we defined the conditional prefix complexity $KP(x|y)$. In this definition we required S to be prefix-stable *with respect to* p: this means that is $S(p.y) = x$ for some p, then $S(p',y) = x$ for any string p' that has prefix p.

Finally, in the proof of Theorem 85 we mentioned the conditional monotone complexity $KM(x|y)$. For its definition a description mode (decompressor) is a computable family of computable continuous mappings $D_y \colon \Sigma \to \Sigma$ (indexed by string y). The computability of this family means that the set of triples $\langle y,u,v\rangle$ such that $v \preccurlyeq D_y(u)$ is enumerable.

The conditional decision complexity can be defined in a similar way.

In these four definition we consider conditions as terminated bit strings, and the behavior of the decompressor is unrelated for different conditions: if we know that p is a description of x relative to y, this gives us no information about the values of decompressor for any other y.

In other terms, a decompressor (say, for the conditional prefix complexity) can be considered as a computable mapping

$$D \colon \Sigma \times \mathbb{N} \to \mathbb{N}_\perp;$$

in the pair $\langle p,y\rangle \in \Sigma \times \mathbb{N}$ the string p is considered as a description (and D is monotone with respect to p) and y is a condition, and no monotonicity is required.

If we change this and consider conditions also as vertices of binary tree requiring monotonicity over conditions, we get four other versions of conditional complexity. These version are not widely used ([**?**] is one of the few exceptions).

In this way we get 8 versions of conditional complexities (for each of three components, i.e., conditions, descriptions and objects, we have two possibilities). The most non-technical definition of these complexities goes as follows. Let $\alpha,\beta,\gamma \in \{=,\asymp\}$ (see Section 6.2). An $(\alpha,\beta)|\gamma$-decompressor (description mode) is an enumerable set S of triples $\langle p,x,y\rangle$, such that

$$\langle p_1,x_1,y_1\rangle \in S, \quad \langle p_2,x_2,y_2\rangle \in S, \quad p_1 \alpha p_2, \quad y_1 \gamma y_2 \Rightarrow x_1 \beta x_2$$

The we define $K_S(x|y)$ as the minimal length of a string p such that $\langle p,x,y\rangle \in S$.

Theorem 105 *In all eight cases there exists an optimal decompressor S that gives the smallest complexity K_S (up to $O(1)$, among all the decompressors of that class).*

145 Give the detailed proof of this theorem (it follows the same scheme as in the case of plain or prefix conditional complexity).

In each of eight classes let us fix some optimal $(\alpha,\beta)|\gamma$-decompressor and denote the corresponding complexity by $K_{(\alpha,\beta)|\gamma}$. In this notation $KP(x|y)$ (as defined earlier) is $K_{(\succcurlyeq,=)|=}$ and $KS(x|y)$ is $K_{(=,=)|=}$.

146 Show that by replacing $=$ by $\succcurlyeq$ in the place of γ we may only increase the complexity. [Hint: This replacements adds more restrictions for a decompressor, so we get less decompressors. For the same reasons the plain complexity does not exceed the prefix one.]

[Problem (probably not very difficult): show that this increase is essential (greater than $O(1)$) and investigate related questions.]

Let us give an example of a statement that involves conditional complexities as they are defined above:

147 Prove that

$$KS(x) \leqslant K_{(=,=)|\succcurlyeq}(x|y) + KR(y) + O(\log KR(y)).$$

Let us now describe one more approach to the definition of the conditional complexity that goes bask to Kolmogorov's interpretation of logical connectives as operations on problems. The conditional complexity of x when y is known can be described as the complexity of the problem: "transform y into x"; moreover, this problem can be considered as a set of all functions that map y into x (any function that maps y to x is a "solution" of this problem).

More formally, let us consider the space $\mathbb{F}$ whose elements are all partial functions whose arguments and values are natural numbers. Let us introduce the following partial order on this set: $f_1 \preccurlyeq f_2$ if f_2 is an extension of f_1 (i.e., $f_1(y) = x$ implies $f_2(y) = x$). By *finite elements* of F we mean functions with finite domain. For each finite element $f \in F$ consider its cone, i.e., the set of all its extensions $\{y | f \preccurlyeq y\}$ (both finite and infinite). We call a continuous mapping $T\colon \mathbb{N}_\perp \to \mathbb{F}$ *computable* if the set of pairs $\langle a, f\rangle$ such that $a \in \mathbb{N}_\perp$, f is a finite element of $\mathbb{F}$ and $f \preccurlyeq T(a)$, is enumerable. Continuous computable mappings $\mathbb{N}_\perp \to F$ are used decompressors for function. For any function $f \in F$ we define the complexity of f (with respect to decompressor T as the minimal length of the string (or, better, the logarithm of the number — recall that we identify strings with natural numbers) a such that $f \preccurlyeq T(a)$.

148 Prove that there exists an optimal decompressor (in this sense) and that the complexity of the function $y \mapsto x$ (whose domain is a singleton $\{y\}$ and whose value is x) is $KS(x|y) + O(1)$.

We can give a similar interpretation of all eight conditional complexities defined above: for every two spaces $Y, X \in \{\mathbb{N}_\perp, \Sigma\}$ we define the space of functions $(Y \to X)$ and then consider computable mappings of the space of descriptions $P \in \{\mathbb{N}_\perp, \Sigma\}$ into the function space $(Y \to X)$. The definition of the function space is given in the spirit of Scott domain theory (or the theory of f_0-spaces in the sense of Ershov, see [**?**] for details).

A slightly different interpretation of (plain) conditional complexity as the complexity of the problem "transform y to x" is considered in Chapter **??**; it does not use the computability in function spaces.

The related notion of complexity for functions was considered by Schnorr[**?**, **?**]. Recall that a *numbering* (an important notion on recursion theory) is a mapping ν that maps each natural number n into some (partial) function ν_n whose arguments and values are natural numbers. A numbering ν is *computable*, if a (partial) function of two arguments

$$\langle n, x\rangle \mapsto \nu_n(x)$$

is computable. A numbering ν is called a *Gödel* numbering if for any other computable numbering μ there exists a computable function that *reduces* μ to ν in the following sense: $\mu_n = \nu_{h(n)}$ for every n. (In particular, the range of a Gödel numbering is the set of all computable functions.)

Following Schnorr, we make this condition stronger and require additionally that $h(n) = O(n)$ (in other terms, the length of the string $h(n)$ exceeds the length of string n at most by a constant, if we identify the natural number with binary strings). If such a function h exists for every computable numbering μ, the numbering ν is called *optimal*.

Theorem 106 *There are optimal numberings.*

$\lhd$ Consider any reasonable programming language for binary functions and let $\hat{u}v$ be a ν-number of the function obtained by fixing first argument equal to v in the function that has program u. (Here u is some self-delimiting encoding of u, i.e., u with doubled bits and 01 appended.) $\rhd$

Schnorr [**?**, **?**] has shown that any two optimal numberings ν_1 and ν_2 can be translated into each other by a computable permutation π that changes the size at most by $O(1)$ (in both directions): this means that $\nu_1(n) = \nu_2(\pi(n))$ for every n and that $\pi(n) = O(n)$ and $\pi^{-1}(n) = O(n)$.

6.4 Complexities and oracles

{classlim}

Relativization is a well known method in computability theory. We take some definition or statement that involves the class of computable function, and replace computable functions by functions that are computable with some oracle (computable *relative to* this oracle). Usually the oracle is a total function α whose arguments and values are natural numbers and/or binary strings, for example, a characteristic function of some set A. An algorithm is allowed to call an "external procedure" that computes the value $\alpha(n)$ for a given value of the parameter n. If α is a characteristic function of a set A, this means that we may ask whether some n belongs to A or not. If the function α is not computable, this permission to ask α-oracle increases our capabilities and we get a class of *α-computable* functions that contains all computable functions but also some non-computable ones (e.g., α).

Then we can develop the general theory of algorithms as usual and define, say α-enumerable sets, or α-computable real numbers, or (closer to our subject) α-lower-semicomputable semimeasures etc. And practically all the theorems of general theory of algorithms (and their proof) remain valid, we need just to add "α-" for all the notions. This procedure is called "relativization".

In particular, for a given set A we may define the notion of A-relativized Kolmogorov complexity allowing decompressors to use oracle A. This can be done for plain, prefix and all other

versions of complexity that we have considered (unconditional or conditional). The use of oracle is shown by a superscript, so, e.g., $KP^A(x)$ denotes prefix complexity relativized by oracle A.

In fact we can prove a bit more: instead of defining complexity for a given oracle A up to $O(1)$ additive term (by proving the existence of an optimal A-decompressor) we may define (with the same precision) the function of two arguments:

$$\langle A, x\rangle \mapsto K^A(x)$$

(here K is one of the complexity versions, say, KP or KM).

149 Show that this indeed can be done and that the resulting complexities coincide with the limits of conditional complexities defined in Section 6.3:

$$KP^A(x) = K_{\succeq,=}(x) = \lim_{n\to\infty} K_{(\succeq,=|\succeq)}(x|A_n),$$

where A_n is the prefix of length n of the characteristic sequence of the set A. (The similar statements are true for other complexity versions.)

Note that relativized complexity does not exceed the non-relativized one (up to $O(1)$), since the algorithm with an oracle is not obliged to use it in any way, so all decompressors are A-decompressors.

For some oracles A and some strings x the A-complexity of x can be much smaller than oracle-free complexity. For example, let A be the universal enumerable set This set is usually denoted by $\mathbf{0}'$. In other words, $\mathbf{0}'$-oracle is an oracle for the halting problem. We may send any program (with its input) to this oracle and it will tell us whether the program terminates at this input.

Using this oracle, we can find for every string x its shortest description (in the standard sense, without oracle) since the oracle tell us which computations terminate. Therefore, the function $|KS$ is $\mathbf{0}'$-computable (the same is true for KP, conditional complexities etc.), and the list of all strings of complexity less than n (that has $n+O(1)$ complexity without the oracle), as well as the numbers $B(n)$ and $BB(n)$ (see Section 1.2) now have $\mathbf{0}'$-complexity only $O(\log n)$.

On the other hand, most strings of length n have $\mathbf{0}'$-complexity $n-O(1)$, and therefore their $\mathbf{0}'$-complexity is close to their non-relativized complexity (and to their length).

6.4.1 Complexity with large numbers as conditions

Let us define a new type of conditional complexity, i.e., the complexity of a string x relative to the set A. Informally speaking, we want to measure the complexity of the following task "obtain x from any element of the set A".

This complexity has several equivalent (up to $O(1)$ definitions).

Here is one of them. Fix some reasonable programming language. (Formally speaking, "reasonable" means that the numbering corresponding to this language is a Gödel numbering, i.e., there exists a translation algorithm from any other programming language, see [**?**] for the details.) Now let us define the conditional complexity of an object x with condition A and the minimal (plain) Kolmogorov complexity of a program that maps *every* element of A into x. (A generalization of this definition is considered in Chapter **??**.)

The existence of a translation algorithm guarantees that this notion is well-defined, i.e., that the complexity defined in this way does not depend on the choice of a programming language (Gödel numbering).

One should not mix this complexity with a completely different notion: a conditional complexity of x with condition A given as a list of its elements. In our case we get not the list of all elements of A, but only one of them, and should be prepared to deal with *any* element of A. To stress this distinction, we use the notation $KS(x\|A)$ for the new complexity (while $KS(x|A)$ denotes the condition complexity of x if a finite set A is given as a list of its elements).

A different (but equivalent) definition of $KS(x\|A)$ can be given as follows. Let D (decompressor) be a computable partial function of two arguments. Let x be a binary string and let A be a set of binary strings. We define $KS_D(x\|A)$ as the minimal length of a string p such that $D(p,y) = x$ for every $y \in A$.

150 Prove that there exists a optimal decompressor in this class (that give the minimal function $KS_D(\cdot\|\cdot)$ up to $O(1)$ additive term). Prove that KS_D for optimal D coincides (up to $O(1)$-term) to the complexity defined above. {relative-c

For a singleton $A = \{a\}$ the complexities $KS(x|A)$ and $KS(x\|A)$ coincide with the standard conditional complexity $KS(x|a)$ up to $O(1)$-term (see Problem 23).

Now let A be the set of all integers greater than some (presumably) large number n. (As usually, we identify natural numbers with binary strings.) The complexity of a string x with respect to this set we denote by $KS(x\| \geqslant n)$. Obviously, thus complexity does not exceed $KS(x)$ and is a non-increasing function of n (and, more generally, $KS(x\|A)$ can only decrease if A becomes smaller; it becomes $O(1)$ for the empty set A). So there exists some limit as $n \to \infty$.

{large-numb

Theorem 107

$$\lim_{n\to\infty} KS(x\| \geqslant n) = KS^{\mathbf{0}'}(x) + O(1).$$

$\lhd$ Assume that the limit equals k. Then here exists a program p of complexity k that maps all sufficiently large numbers to x. If an oracle $\mathbf{0}'$ is available, this program can be considered as a $\mathbf{0}'$-description of x. Indeed, given this program, we search for N and y such that *p does not map any $n \geqslant N$ into an object that differs from y*. The emphasized property can be checked by using $\mathbf{0}'$-oracle since it has an enumerable negation. And our assumption guarantees that y equals x. Therefore,

$$KS^{\mathbf{0}'}(x) \leqslant \lim_{n\to\infty} KS(x\| \geqslant n) + O(1).$$

On the other hand, let y be a description of x with respect to a $\mathbf{0}'$optimal decompressor and let k be the length of y. Consider a following program that has additional input N: make N steps of the enumeration of the universal set $\mathbf{0}'$ and then use the set of enumerated elements as a oracle for decompression of y. This program can be constructed effectively given y, therefore its complexity does not exceed $KS(y) + O(1) \leqslant l(y) + O(1) = k + O(1)$. On the other hand, if N is large enough, this program generates x (since only finite number of oracle calls are performed during the decompression of y, for all sufficiently large N these questions get correct answers even if the oracle is replaced by its N-approximation). $\rhd$

It turns out that a similar question is true if we replace $KS(x\|\geqslant n)$ by $\sup_{m\geqslant n} KS(x|m)$. Note that

$$\sup_{m\geqslant n} KS(x|m) \leqslant KS(x\|\geqslant n),$$

since the optimal program in the right-hand side can be used for any m in the left-hand side. This is easy; the surprising result is that both sides have the same limit as $n\to\infty$ (up to $O(1)$ term):

{large-numb

Theorem 108

$$\limsup_{n\to\infty} KS(x|n) = KS^{\mathbf{0}'}(x) + O(1).$$

$\lhd$ We have to prove that if (for some string x and integer k)

$$KS(x|n) < k \text{ for any sufficiently large } n,$$

then $\mathbf{0}'$-complexity of x does not exceed $k+O(1)$. The difficulty here is that here (unlike the previous theorem) the program of length less than k can depend on n, and none of them works for all sufficiently large n.

Note that there is less than 2^k strings x with this property (for a given k). Indeed, if we have more of them, then for sufficiently large n we run out of programs of length less than k.

It would be enough to prove that the set of x that have this property is a $\mathbf{0}'$-enumerable set whose enumeration effectively depends on k (in other terms, it would be enough to prove that the function $x\mapsto \limsup KS(x|n)$ is $\mathbf{0}'$-enumerable from above). However, the natural description of this set,

$$\exists N\,(\forall n\geqslant N)\,[KS(x|n)<k],$$

shows only that it is a Σ_3-set (the condition in brackets is enumerable and two quantifiers precede it), so we choose an another approach.

Note that we do not really need that this set is $\mathbf{0}'$-enumerable. It is enough to show that it is a subset of an $\mathbf{0}'$-enumerable set that contains less than 2^k elements for a given k. This can be done as follows.

Consider the two-dimensional enumerable set of pairs $\langle n,x\rangle$ such that $KS(x|n)<k$. This set (for any k) is "thin" in the following sense: all vertical sections of this set (for fixed n) contain less than 2^k elements.

Consider some point $\langle n,x\rangle$. Let us try to add a horizontal ray that goes on the right from this point, to our set (i.e., add all pairs $\langle m,x\rangle$ for all $m\geqslant n$). The set may remain thin or not, and this two cases can be distinguished by an $\mathbf{0}'$-oracle. Indeed, the negation of being thin is an enumerable property (there exists a section that has at least 2^k different elements including the added one).

Let us perform this attempts (to add the horizontal ray starting from some pair $\langle n,x\rangle$) sequentially for all pairs in some order. (If some ray is added successfully, then its elements are taken into account for all subsequent attempts.) This process is $\mathbf{0}'$-computable and therefore the ordinates of all added rays form a $\mathbf{0}'$-enumerable set.

This set has less than 2^k elements (since we add rays only if the resulting set is still thin) and contains every x such that $\limsup K(x|n)<k$. Indeed, for such an x there is some ray that lies entirely is the initial set, and this ray can be added at any time. $\rhd$

(This proof is a simplified version of the proof given in [**?**].)

We can also obtain the results for prefix complexity that are similar to Theorem 107 and 108. However, the definition of a conditional prefix complexity with respect to a set is quite subtle, so we postpone its discussion and start with the second theorem.

{large-numb

Theorem 109

$$\limsup_{n\to\infty} KP(x|n) = KP^{\mathbf{0}'}(x) + O(1)$$

$\lhd$ Using a priory probabilities (conditional and unconditional), we rewrite the statement as follows:

$$\liminf_{n\to\infty} m(x|n) = m^{\mathbf{0}'}(x)$$

(the equality is understood up to a bounded factor in both directions).

Let us show first that the left-hand side does not exceed the right-hand side (or, more precisely, exceeds it at most by a $O(1)$ factor). Indeed, let us consider an $\mathbf{0}'$-oracle probabilistic machine whose output has distribution $m^{\mathbf{0}'}$. Then for any integer n we may run this machine with a changed oracle: instead of the entire oracle we use its approximation obtained after n steps. This, of course, changes the output distribution, however, the $\liminf$ of the probabilities to get some x using n-approximation to the oracle (as $n \to \infty$) is greater than or equal to the probability to get x with the entire oracle. Indeed, if x appears in a $\mathbf{0}'$-computation for some combination of random bits, then this computation depends only on some finite part of the oracle and therefore the same random bits will give the same output x if the approximation to the oracle is good enough (i.e., n is sufficiently large). (Note that $\liminf$ can be bigger than the probability to get x with a correct oracle, since approximate oracles can lead to output x for a combinations of random bits that do not generate x with a correct oracle.)

Now let us prove the reverse inequality. This proof resembles the proof of Theorem 108. We have a lower semicomputable family of semimeasures: for each n the function $x \mapsto m(x|n)$ is a semimeasure (i.e., $\sum_x m(x|n) \leqslant 1$ for each n). It follows that the function

$$m'(x) = \liminf_{n\to\infty} m(x|n)$$

is also a semimeasure, i.e., the sum $\sum_x m'(x)$ does not exceed 1. If this function were $\mathbf{0}'$-lower-semicomputable, this would finish the proof; however, we have the equivalence

$$r < \liminf_{n\to\infty} m(x|n) \Leftrightarrow (\exists q > r)\, \exists N\, (\forall n > N)\, [q < m(x|n)]$$

where the right-hand side has too many quantifiers (note that the property in the brackets is enumerable, not decidable). But again we may replace the function m' by any larger function, so it remains to construct an $\mathbf{0}'$-lower-semicomputable upper bound for m'.

To achieve this goal let us consider triples $\langle N, x, \varepsilon$ (where ε is a positive rational number). For a given triple we try to increase the values $m(\cdot|\cdot)$ up to ε on a ray that consists of pairs $\langle n, x\rangle$ for fixed x and for all $n \geqslant N$. This change is performed only if we get semimeasures (i.e., for every n the sum over all x does not exceed 1).

As before, we can check whether such an increase is possible using $\mathbf{0}'$-oracle. (Indeed, the violation is an enumerable event.) Let us consider sequentially all triples and perform the increase when possible (the increased values are taken into account on the subsequent steps). Then for each possible increase we keep the values of x and ε. In other words, we consider a function that on every x is equal to the upper bound of all ε that are used for increase together with that x. In this way we get a $\mathbf{0}'$-enumerable family of semimeasures that is an upper bound for m'. Indeed, if m' is greater than ε for some x, the function m is greater than ε on some ray, increase does not really change anything and therefore is possible. $\triangleright$

To formulate a similar statement for $KP\,(x\| \geqslant n)$ we should first of all define this prefix complexity relative to a set. Here we have several possibilities, and it is unclear which of the is "the right thing".

We may try to define $KP\,(x\|A)$ and the minimal prefix complexity of a program that outputs x when applied to any element of A. However Problem 79 (p. 94) shows that this definition does not match $KP\,(x|a)$ for singleton conditions, so probably this definition is not a good one.

Another definition is similar to the approach used in Problem 150. Consider an arbitrary computable function $\langle p,x\rangle \mapsto D(p,x)$ that is prefix-stable with respect to its first argument (if the second one is fixed). For any x and for any set A we then define $KP_D(k\|A)$ as the minimal length of a string p such that $f(p,n) = k$ for all $n \in A$. The difference (compared to plain complexity) is that we require the conditional decompressor to be prefix-stable with respect to the first argument. There exists an optimal decompressor in this class that gives the least function KP_D (up to $O(1)$ additive term). This function can be called prefix complexity $KP\,(x\|A)$.

151 Show that the same complexity (up to $O(1)$) is obtained if decompressors are computable continuous mappings $\Sigma \to \mathbb{F}$ (here Σ is the space of finite and infinite sequences of zeros and ones, and $\mathbb{F}$ is the space of partial functions from $\mathbb{N}$ to $\mathbb{N}$) and complexity is a shortest string that is mapped to some partial function that is equal to x on any element of A.

We can also define the prefix complexity with set condition using prefix-free functions instead of prefix-stable ones. Again, in the class of computable prefix-free functions there exists an optimal one (that gives the smallest complexity function $KP_f(x\|A)$). In this way we get the definition of some function $KP'(x\|A)$ that resembles the conditional complexity $KP'(k|n)$ and coincides with it (up to $O(1)$) if $A = \{n\}$.

Finally one can define a priori probability $m(x\|A)$. For that we consider some probabilistic machine that has input y and the measure of the set of all sequences $\omega \in \Omega$ that (being used as random bits) makes the machine transform any input $y \in A$ into x. Again, there exists an optimal machine that maximizes this probability (up to $O(1)$ constant factor) and for singletons this definition coincide with our definition of the conditional a priori probability.

The inequalities

$$-\log m(k\|A) \leqslant KP(k\|A) + O(1) \leqslant KP'(k\|A) + O(1),$$

can be proved as we did for conditional prefix complexity, but the argument that showed us that all three expression coincide does not work as before, and authors do not know whether these inequalities are strict. But it is easy to see that all three expressions are greater than

$$-\log \inf_{x\in A} m(k|x) = \sup_{x\in A} KP\,(x|a),$$

so any of them can be used in the theorem similar to Theorem 107. In particular for $KP(x\|A)$ (which seems to be most natural among all three) we get the following result:

{large-numb

Theorem 110

$$\lim_{n\to\infty} KP(x\| \geqslant n) = KP^{\mathbf{0}'}(x) + O(1).$$

[It would be nice to find out whether the inequalities indeed are strict.]

6.4.2 Limit frequencies and $\mathbf{0}'$-a-priori-probability

We conclude this section by a result from [**?**]; it relates the frequencies in computable sequences to the $\mathbf{0}'$-relativized prefix complexity.

Let $f(0), f(1), \ldots$ be a computable sequence of natural numbers. For a given n and k let us count the appearances of k among $f(0), \ldots, f(n-1)$ and divide the result by n. The ratio can be called the *frequency* of k among the first n terms of the sequence.

Now for a fixed k consider the limit inferior of this frequency as $n \to \infty$; we call it *lower frequency* of element k in the sequence f.

Let p_k be a lower frequency of k in a given sequence. It is easy to check that $\sum_k p_k \leqslant 1$. Indeed, if some partial sum of this series exceeds 1, then a finite sum of limit inferiors exceeds 1, and for sufficiently large n the sum of the frequencies among the first n terms of the sequence exceeds 1, which is impossible.

The following statement is true for any computable sequence f:

{zero-prime

Theorem 111 *The function $k \mapsto p_k$ is $\mathbf{0}'$-lower-semicomputable.*

(Here p_k is lower frequency of k; the definition of the lower semicomputable function is given in Section 4.1; now we consider $\mathbf{0}'$-relativized version of this definition.)

$\lhd$ Indeed, the statement $r < p_k$ (where r is some rational number) is equivalent to the following one:

> there exist a rational number $p > r$ and integer N such that *the frequency of k among the first n terms of f exceeds p for any $n > N$*.

The property printed in italics is co-enumerable (has an enumerable negation): if it is not true, we can establish it by showing the number n that violates it. Therefore this property is $\mathbf{0}'$-decidable (we apply the oracle to the algorithm that searches for that n). So the property $r < p_k$ is $\mathbf{0}'$-enumerable. $\rhd$

In fact we use the following general observation:

152 Let r_n be a computable sequence of rational numbers. Show that $\liminf r_n$ is a $\mathbf{0}'$lower-semicomputable real number and the corresponding $\mathbf{0}'$-algorithm can be effective found given an algorithm for r_n.

By the way, the reverse statement is also true:

153 Any $\mathbf{0}'$-lower-semicomputable real number is a limit inferior of a computable sequence of rational numbers. {liminf-cri

[Hint: This number is a supremum of a $\mathbf{0}'$-computable sequence of rational numbers r_n. Each r_n is an ultimate values of a stabilizing sequence $r_{n,k}$. Let s_k be the maximum of $r_{0,k},\dots,r_{t-1,k}$ where t is the smallest number such that $r_{t,k}\ne r_{t,k-1}$.]

It turns out that for an appropriate sequence f the function $k\mapsto p_k$ is a maximal $\mathbf{0}'$-lower-semicomputable semimeasure. This is a corollary of the following result:

{zero-prime

Theorem 112 *For any $\mathbf{0}'$-lower-semicomputable sequence $q_0,q_1,\dots$ of non-negative reals such that $\sum_i q_i\leqslant 1$ there exists a computable sequence $f(0),f(1),\dots$ such that lower frequency of any k in the sequence f is at least q_k.*

This allows us to give an equivalent definition of $\mathbf{0}'$-relativized prefix complexity of k: it is the negative logarithm of the lower frequency of k in the optimal sequence f (that gives maximal lower frequencies up to $O(1)$-factor).

$\lhd$ Since q_k is lower semicomputable, the set of pairs $\langle r,k\rangle$ where r is a rational number smaller than q_k is $\mathbf{0}'$-enumerable. As we know from the general computability theory (see, e.g., [**?**]), $\mathbf{0}'$-enumerable sets are Σ_2-sets, i.e., there exists a decidable property R such that

$$r<q_k\Leftrightarrow\exists u\,\forall v\,R(r,k,u,v)$$

We use a slightly different representation of Σ_2-sets: there exists a computable total function $\langle r,k,n\rangle\mapsto S(r,k,n)$ with 0/1-values such that $r<q_k$ if and only if the sequence $S(r,k,0),S(r,k,1)\dots$ has finite number of zeros. The sequence $S(r,k,0),S(r,k,1)\dots$ can be constructed as follows: we consider (sequentially) the values $u=0,1,2,\dots$ and for each u we search for v such that $R(r,k,u,v)$ is false. While searching, we extend the sequence adding zeros; when v is found, we add 1 to the sequence and switch to the next value of u. The number of zeros in the constructed sequence is finite if and only if the search was unsuccessful for some u, i.e., if $r<q_k$.

It is convenient to visualize this process as follows: from time to time the request "please make q_k greater than r" appears for some k and r, (and the previous request with the same k and r is canceled). Then we consider the requests that appear and are never canceled later; they correspond to pairs $\langle r,k\rangle$ such that $r<q_k$. (The moments when requests appear correspond to zeros in the sequence S.) This process is computable. We may also assume without loss of generality that at any given moment there is only finite number of requests exists. (This does not matter since only the limit behavior of the sequence is important.)

Recall that we need a computable sequence $f(0),f(1),\dots$ for which the lower frequency of k is at least q_k. To achieve this goal, it is enough to represent the given $\mathbf{0}'$-lower-semicomputable semimeasure as a limit inferior of a computable sequence of measures with rational values, i.e., to construct a two-dimensional table of rational numbers

$$\begin{array}{cccc} p_0^0 & p_1^0 & p_2^0 & \dots\\ p_0^1 & p_1^1 & p_2^1 & \dots\\ p_0^2 & p_1^2 & p_2^2 & \dots\\ \dots & \dots & \dots & \dots \end{array}$$

such that each row has only finite number of non-zero elements that have sum 1, and that limit inferior in the kth column is at least q_k. Indeed, let us assume that such a table is constructed.

Without loss of generality we may suppose that in the ith row all the numbers are multiples of $1/i$ (we can take approximation with precision $1/i$ not changing the limit). Then the sequence f can be constructed as follows: first we use the first row as the table of frequencies, then switch to the second row and use it much longer (to make the influence of the first row negligible), then use the third row even longer (to make the influence of the first and second rows negligible) etc.

So it remains to construct a table p^i_j with the following property: if some request "please make q_k greater than r" appears at some moment and is not canceled later, then that kth column has limit inferior at least q_k. This is done as follows: constructing nth row (at time n), we try to satisfy all current requests (that have been appeared and are not canceled) according to their age (the oldest request is treated first). For each request we increase the corresponding p_k up to a given r if this is possible (does not make the sum greater than 1). We may assume that there are many requests and at some point the sum becomes greater than 1; at that moment we cut the last request (so the sum is 1) and this finishes the construction of nth row.

Why this helps? Imagine that $r < q_k$ is true. Then the request "please make q_k greater than r" at some moment appears and is never canceled later. (It need not to be the first appearance of this request.) Let us look at all requests that appear before this one. Some of them are canceled later (while others are "final"). Let us wait until all these cancellations happen. After that only "true" requests (that are never canceled later) are older than our request, and for these true requests we have $r' < q_{k'}$. Their sum therefore does not exceed 1 together with our request, so the requests with high priority at that time will not interfere with our request. $\triangleright$

154 Prove that there exists a computable sequence where the lower frequencies coincide with q_k.

[Hint: combine the proof of this theorem with the solution of Problem 153.]

155 Prove that theorem 112 remains true if we consider partial computable functions f from $\mathbb{N}$ to $\mathbb{N}$ instead of sequences: for any partial computable function f from $\mathbb{N}$ to $\mathbb{N}$ there exists a (total) computable sequence $g(0), g(1), \ldots$ that have the same (or bigger) lower frequencies: for any k the lower frequency of k in g is at least its lower frequency in $f(0), f(1), \ldots$ (which is defined as the limit inferior of the number of appearances of k among $f(0), \ldots, f(N-1)$ divided by N). [Hint: for every N the frequencies in the initial segment of length N form a lower semicomputable semimeasure (it was a measure for total sequences); the construction used in the proof of Theorem 109 allow to find an upper bound for the limit frequencies by a $\mathbf{0}'$-lower-semicomputable semimeasure. Then we apply Theorem 112.]

[Here could be the argument about oracles that do not change KP and new proof of the existence of nonT-complete enumerable sets]

7 Shannon entropy and Kolmogorov complexity

{entropy}

7.1 Shannon entropy

{entropy-de

Consider an alphabet A that contains k letters $a_1, \ldots, a_k$. We want to encode each letter a_i by a binary string c_i. Of course, we want all c_i to be different to avoid confusion. But this is not enough if we write codewords without any separator. Example: letters A, B and C are encoded by strings 0, 1 and 01. All three codes are different, but two strings ABAB and ABC have identical codes 0101. So additional precautions are needed to guarantee unique decoding.

We want the code to allow unique decoding. At the same time we want it to be space-efficient. It is good to have the strings c_i as short as possible (without violating the unique decoding property). And if we cannot make all codewords short, the priority should be given to the frequent letters. (Similar considerations were taken into account when Morse code was designed.)

7.1.1 Codes

{prefix-coc

Let us give formal definitions now. A *code* for a k-letter alphabet $A = \{a_1, \ldots, a_k\}$ consists of k binary strings $c_1, \ldots, c_k$. These strings are called *codewords* (for the code considered); letter a_i has *encoding* c_i. Any A-string (finite sequence of letters taken from A) has an *encoding*; to get it we encode each letter and write these codes one after another (without separators).

A code is *injective* if different letters have different codes. A codes is *uniquely decodable* if any two different A-strings have different codes. A *prefix code* is a code where no codeword is a prefix of another codeword. (This is a traditional term; however, the more logical name "*prefix-free* code" is also used.)

Theorem 113 *Every prefix code is uniquely decodable.*

$\lhd$ The first codeword (the encoding of the first letter) is determined uniquely (due to the prefix property), so we can separate it from the rest. Then the second codeword is determined, etc. $\rhd$

156 Show that there exist uniquely decodable codes which are not prefix codes. [Hint. Consider a "suffix" code.]

157 Construct an explicit bijection between the set of all infinite sequences of digits $0, 1, 2$ and the set of all infinite sequences of digits $0, 1$. [Hint. Use the prefix code $0 \mapsto 00$, $1 \mapsto 01$, $2 \mapsto 1$.]

158 Consider two prefix codes $c_1, \ldots, c_k$ (for a k-letter alphabet) and $d_1, \ldots, d_l$ (for a l-letter alphabet). Show that strings $c_i d_j$ (concatenations of codewords from these two codes) form a prefix code for a kl-letter alphabet.

Before asking which of two codes is more space-efficient, we should fix frequencies of the letters. Let $p_1, \ldots, p_k$ be non-negative reals such that $p_1 + \ldots + p_n = 1$. The number p_i will be called *frequency* or *probability* of letter a_i. For each code $c_1, \ldots, c_k$ (for alphabet $a_1, \ldots, a_k$) its *average length* is defined as

$$\sum_i p_i l(c_i)$$

Now we can formulate our goal: for given $p_1, \ldots, p_k$ we want to find a code of minimal average length inside some class of codes, e.g., an uniquely decodable code of minimal average length.

159 Which injective code has minimal average length (among injective codes) for given $p_1, \ldots, p_n$? [Hint: Put all letters in the decreasing frequency order, and all binary strings in the increasing length order.]

7.1.2 The definition of Shannon entropy

{entropy-co

Shannon entropy provides a lower bound for the average length of a uniquely decodable code. It is defined (for given non-negative p_i such that $\sum_i p_i = 1$) as

$$H = p_1(-\log p_1) + p_2(-\log p_2) + \ldots + p_k(-\log p_k)$$

(We assume that $p \log p = 0$ for $p = 0$ making function $p \log p$ continuous at the point $p = 0$.)

Some motivation for this definition: letter a_i appears with frequency p_i, and each occurrence of a_i carries $-\log p_i$ "bits of information", so the average number of bits per letter is H. But then we should explain also why we believe that each occurrence of the letter that has frequency p_i carries $-\log p_i$ bits of information. OK, imagine that somebody has in mind one of 2^n possible numbers and you want to guess this number by asking yes or no questions. Then you need n questions, and each answer gives you one bit of information; so when event having probability $1/2^n$ happens it brings us n bits of information.

Of course, the previous paragraph is just a mnemonic rule for the definition of entropy. The formal reason to introduce this notion is given by the following theorem:

{prefix-coc

Theorem 114 *Let $p_1, \ldots, p_n$ be non-negative reals such that $p_1 + \ldots + p_n = 1$.*

(**a**) *The average length of every prefix code $c_1, \ldots, c_k$ is at least H (the entropy):*

$$\sum_i p_i l(c_i) \geqslant H.$$

(**b**) *There exists a prefix code such that*

$$\sum_i p_i l(c_i) < H + 1.$$

$\lhd$ Note that this theorem deals only with the lengths of codewords (but not the codewords itself). So it is important to know when given integers $n_1, \ldots, n_k$ could be lengths of codewords in a prefix code. Here is the criterion:

Lemma (Kraft inequality). *Assume that non-negative integers $n_1, \ldots, n_k$ are fixed and we want to find binary strings $c_1, \ldots, c_k$ of these lengths ($l(c_i) = n_i$) that form a prefix code (i.e., c_i is not a prefix of c_j for $i \neq j$). This is possible if and only if*

{kraft-lemm

$$\sum_i 2^{-n_i} \leqslant 1.$$

This statement already appeared, see lemmas in the proofs of Theorems 50 (p. 82) and 52 (p. 83). It is easy to explain: if c_i is never a prefix of other string c_j, then the corresponding

intervals of lengths 2^{-n_i} are disjoint, and the sum of their lengths does not exceed 1. (Using the probabilistic language: a random string of 0s and 1 has prefix c_i with probability 2^{-n_i}; these k events are disjoint, so the sum of probabilities does not exceed 1.)

Going in the opposite direction, we can use a simpler argument that was used before (see the proof of Theorem 52). The simplification is possible since we have only a finite number (k) of integers and they are given in advance. We can simply place the corresponding intervals of lengths 2^{-n_i} inside $[0,1]$ from left to right going in decreasing length order. Then each interval is properly aligned and corresponds to a binary string of length n_i.

Let us prove the theorem now. Without loss of generality we may assume that all p_i are strictly positive (since null values change neither Shannon entropy nor average code length). The part (a) of our theorem says that if n_i are non-negative integers and $\sum_i 2^{-n_i} \leqslant 1$, then $\sum p_i n_i \geqslant 1$. It is true for any reals n_i (even if they are not integers). Indeed, let q_i be equal to 2^{-n_i}. In these coordinates the statement reads as follows: if $q_i > 0$ and $\sum q_i \leqslant 1$, then

$$\sum p_i(-\log q_i) \geqslant \sum p_i(-\log p_i).$$

This inequality is sometimes called *Gibbs inequality*. To prove it, we rewrite the difference between right-hand side and left-hand side as

$$\sum_i p_i \log \frac{q_i}{p_i} \qquad (*)$$

Then we use the convexity of the logarithm function: the weighted sum of logarithms does not exceed the logarithm of the weighted sum, $\sum p_i \log u_i \leqslant \log(\sum_i p_i u_i)$ (if u_i are positive). In our case we see that $(*)$ does not exceed

$$\log\left(\sum_i p_i \frac{q_i}{p_i}\right) = \log\left(\sum q_i\right) \leqslant \log 1 = 0.$$

The item () is proven.

Let us mention also that the non-negative number

$$\sum_i p_i \log \frac{p_i}{q_i}$$

is called *Kullback – Leibler distance* between two probability distributions p_i and q_i (so we assume that $\sum q_i = 1$), or *Kullback – Leibler divergence*; the latter name is better since this 'distance" is not symmetric. The convexity of logarithm (its second derivative is negative everywhere) guarantees that this distance is always non-negative and equals zero only if $p_i = q_i$ for all i.

To prove item (b), consider the integers $n_i = \lceil -\log_2 p_i \rceil$ (where $\lceil u \rceil$ is a minimal integer greater than or equal to u). Then

$$\frac{p_i}{2} < 2^{-n_i} \leqslant p_i$$

The inequality $2^{-n_i} \leqslant p_i$ allows to use the lemma, so there exist codewords of corresponding lengths. The inequality $p_i/2 < 2^{-n_i}$ implies that n_i exceeds $(-\log p_i)$ less than by 1, and this

remains true after averaging: the average code length ($\sum p_i n_i$) exceeds $H = \sum p_i(-\log p_i)$ less than by 1. $\triangleright$

In a sentence the idea of the proof can be explained as follows: if we forget that code-lengths should be integers and allow any n_i such that $\sum_i 2^{-n_i} \leqslant 1$, the optimal choice is $n_i = -\log p_i$ (convexity of the logarithm function); making n_i integers, we lose less than 1.

Theorem 115 *The entropy of the distribution $p_1, \ldots, p_n$ (with n possible values) does not exceed* $\log n$. *It equals* $\log n$ *only if all p_i are equal.*

$\triangleleft$ If n is a power of 2, the inequality $H \leqslant \log n$ follows from Theorem 114 (consider a prefix code where n codewords have length $\log n$. In general case we use Gibbs inequality for $q_i = 1/n$ (for all i) and recall the this inequality becomes an equality only if $p_i = q_i$. $\triangleright$

7.1.3 Huffman code

{huffman-er

We have shown that the average length of an optimal prefix code (for given $p_1, \ldots, p_k$) is somewhere between H and $H+1$. But how can we find this optimal code?

Let $n_1, \ldots, n_k$ be the lengths of codewords for and optimal code (for given $p_1, \ldots, p_k$). Rearranging the letters, we may assume that

$$p_1 \leqslant p_2 \leqslant \ldots \leqslant p_k.$$

It this case

$$n_1 \geqslant n_2 \geqslant \ldots \geqslant n_k.$$

Indeed, if a letter has longer code than another letter that is less frequent, the codewords exchange (between these two letters) decreases the average length of code.

One can not also that $n_1 = n_2$ for an optimal code (the two less frequent letter have the same code-length). Indeed, if $n_1 > n_2$, then n_1 is greater than all n_i. So the first term in the sum $\sum_i 2^{-n_i}$ is smaller than all other terms, and the inequality $\sum_i 2^{-n_i} \leqslant 1$ cannot be an equality (all terms except the first one are multiples of the second term) and the difference between its two sides is at least 2^{-n_1}. Therefore, we can decrease n_1 by 1 and still do not violate the inequality $\sum_i 2^{-n_i} \leqslant 1$. This means that the code is not optimal (in contrary to our assumption).

So we can look for an optimal code among codes that have $n_1 = n_2$; this optimal code minimizes the sum

$$p_1 n_1 + p_2 n_2 + p_3 n_3 + \ldots + p_k n_k = (p_1 + p_2)n + p_3 n_3 + \ldots + p_k n_k$$

(here n is the common value of n_1 and n_2). In the last expression the minimum should be taken over all sequences $n, n_3, \ldots, n_k$ such that

$$2^{-n} + 2^{-n} + 2^{-n_3} + \ldots + 2^{-n_k} \leqslant 1.$$

This inequality can be rewritten as

$$2^{-(n-1)} + 2^{-n_3} + \ldots + 2^{-n_k} \leqslant 1,$$

and the expression that is minimized can be rewritten as

$$(p_1+p_2)+(p_1+p_2)(n-1)+p_3n_3+\ldots+p_kn_k.$$

The term (p_1+p_2) is a constant that does not influence the minimal point, so the problem reduces to finding an optimal prefix code for $k-1$ letters that have probabilities $p_1+p_2, p_3, \ldots, p_k$.

So we obtain the recursive algorithm that finds the optimal prefix code as follows:

• combine the two most rare letters into one (adding their probabilities);

• find the optimal prefix code for the resulting probabilities (a recursive call);

• replaces codewor x for a "virtual" combined letter by two codewords $x0$ and $x1$ which are one bit longer (note that this replacement keeps the prefix property).

The optimal code constructed by this algorithm is called *Huffman code* for a given distribution $p_1, \ldots, p_n$.

7.1.4 Kraft – McMillan inequality

{kraft-mcmi

So far we have studied prefix codes. It turns out that they are as efficients as general uniquely decodable codes, as the following theorem shows.

{mcmillan-i

Theorem 116 (McMillan inequality) *Let $c_1, \ldots, c_k$ be a code words of an uniquely decodable code and $n_i = l(c_i)$ be their lengths, Then*

$$\sum_i 2^{-n_i} \leqslant 1.$$

Therefore (recall the lemma above) for any uniquely decodable code there is a prefix code with the same code-lengths.

$\lhd$ Let us use letters u and v instead of digits 0 and 1 when constructing codewords. (E.g., the code 0, 01 and 11 is now written as u, uv, vv.) Now take a formal sum $(c_1+\ldots+c_k)$ of all codewords and compute its Nth power (for some N that we choose later). Then we open the parentheses without changing the order of factors u and v (as if u and v were two non-commutative variables). For example, the code above gives (for $N=2$) the expression

$$(u+uv+vv)(u+uv+vv) = uu+uuv+uvv+uvu+uvuv+uvvv+vvu+vvuv+vvvv.$$

Each term in the right-hand side is a concatenation of some codewords. The unique decoding property guarantees that all the terms are different. Now we let $u=v=1/2$. The left-hand side $(c_1+\ldots+c_k)^N$ becomes $(2^{-n_1}+\ldots+2^{-n_k})^N$. For the right-hand side we have an upper bound: if it consisted of *all* strings of length t, it would contain 2^t terms equal to 2^{-t} (each), so the sum would be equal to 1 (for each length t). Therefore, the right-hand side does not exceed the maximal length of strings in the right-hand side, which equals $N\max(n_i)$.

If $\sum 2^{-n_i} > 1$, we immediately get a contradiction, since for large enough N the left-hand side grows exponentially in N while the right-hand side is linear in N. $\rhd$

This proof looks as an extremely artificial trick (though a nice one). A more natural proof (or, better to say, a more natural version of the same proof) is given below, see p. 178.

7.2 Pairs and conditional entropy

{entropy-pa

7.2.1 Pairs of random variables

{entropy-pa

Dealing with Shannon entropies, we use the terminology which is standard for probability theory. Let ξ be a random variable which takes finitely many values $\xi_1, \ldots, \xi_k$ with probabilities $p_1, \ldots, p_k$. Then the *Shannon entropy* of a random variable ξ is defined as

$$H(\xi) = p_1(-\log p_1) + \ldots + p_k(-\log p_k)$$

This definition allows us to consider the entropy of a pair of random variables ξ and η (that have a common distribution, i.e., are defined on the same probability space). Indeed, this pair is also a random variable with a finite range. The following theorem says that the entropy of a pair does not exceed the sum of entropies of its components:

{entropy-pa

Theorem 117

$$H(\langle \xi, \eta \rangle) \leqslant H(\xi) + H(\eta)$$

We consider random variables with finite ranges, so this is just some inequality involving logarithms. Let is write this inequality. Assume that ξ has k values $\xi_1, \ldots, \xi_k$ and η has l values $\eta_1, \ldots, \eta_l$. Then the maximal possible number of values for the pair $\langle \xi, \eta \rangle$ is kl and these values are $\langle \xi_i, \eta_j \rangle$ (some of them may never appear or have probability 0). The distribution for $\langle \xi, \eta \rangle$ is therefore a table that has k rows and l columns. The number p_{ij} (ith row, jth column) is the probability of the event "$(\xi = \xi_i)$ and $(\eta = \eta_j)$" (here $i = 1, \ldots, k$ and $j = 1, \ldots, l$). All p_{ij} are non-negative and their sum equals 1. (Some of p_{ij} can be equal to 0.)

Adding the numbers in each row, we get the probability distribution for ξ: the probability of value ξ_i equals $\sum_j p_{ij}$. We denote this sum by p_{i*}. Similarly, η takes value η_j with probability p_{*j} which equals the sum of all numbers in jth column.

Therefore, the theorem in question is an inequality that is applicable to any matrix with non-negative elements and sum 1:

$$\sum_{i,j} p_{ij}(-\log p_{ij}) \leqslant \sum_i p_{i*}(-\log p_{i*}) + \sum_j p_{*j}(-\log p_{*j})$$

(here p_{i*} p_{*j} are rows' and columns' sums).

This inequality again is a consequence of the convexity of logarithm, but it is useful to understand its intuitive meaning. Let us forget for a while that entropy is not exactly equal to the length of the shortest prefix code (and ignore the difference that does not exceed 1). Then this inequality can be proven as follows. Assume that space-efficient prefix codes for ξ and η are given that have codewords $c_1, \ldots, c_k$ and $d_1, \ldots, d_l$ respectively. Then consider a code for $\langle \xi, \eta \rangle$ that assigns to the value $\langle \xi_i, \eta_j \rangle$ the string $c_i d_j$ (concatenation of c_i and d_j without separator). We get a prefix code (indeed, to separate codeword that starts an infinite sequence, we first find prefix c_i and then prefix d_j in the remaining part; both operation can be performed uniquely). The average length of this code equals the sum of the average lengths of its components. This code may be non-optimal (which is natural, since the inequality could be strict), but provides an upper bound for the length of the optimal code.

$\lhd$ Let us transform this informal argument into a proof. Recall the proof of Theorem 114 (p. 170). We have seen there that the entropy is a minimal value of $\sum_i p_i(-\log_2 q_i)$ taken over all tuples of non-negative reals q_i that have sum 1. In particular, the entropy of the pair (left-hand side) is the minimal value of

$$\sum_{i,j} p_{ij}(-\log q_{ij})$$

taken over all tuples q_{ij} of non-negative reals that sum up to 1. Let us restrict our attention to "rank 1" tuples that have the form

$$q_{ij} = q_{i*} \cdot q_{*j}$$

for some tuples of non-negative reals q_{i*} q_{*j} (both tuples have sum 1). Then $(-\log q_{ij})$ can be decomposed into sum $(-\log q_{i*}) + (-\log q_{*j})$, and the entire sum is decomposed into two parts, which after partial summation over one coordinate become equal to

$$\sum_i p_{i*}(-\log q_{i*})$$

and

$$\sum_j p_{*j}(-\log q_{*j})$$

respectively. The minimal values of the two parts are $H(\xi)$ and $H(\eta)$.

Therefore, the left-hand side of our inequality is the minimum over all tuples and the right-hand side is the minimum over rank 1 tuples, and the inequality is proven. $\rhd$

7.2.2 Conditional entropy

{conditiona

Recall the definition of conditional probability. Let A and B be two events. The *conditional probability* of B with condition A (denoted as $\Pr[B|A]$) is defined as the ratio $\Pr[A \text{ and } B]/\Pr[A]$. This definition assumes that $\Pr[A] > 0$. The motivation is clear: we look at the fraction of outcomes when B happened but restrict our attention to the case when A happened.

Let A be an event (that has non-zero probability) and let ξ be a random variable with finite range $\xi_1, \ldots, \xi_k$. Then we may consider the *conditional distribution* of ξ when A happens. We get a new random variable: now ξ_i has probability $\Pr[(\xi = \xi_i)|A]$ instead of $\Pr[\xi = \xi_i]$. The entropy of this distribution is called *conditional entropy of ξ with condition A* and is denoted by $H(\xi|A)$. (The distribution itself could be denoted by $(\xi|A)$.)

160 Show that $H(\xi|A)$ can be greater than $H(\xi)$ and can be less then $H(\xi)$. [Hint: the distribution $(\xi|A)$ has not much in common with the distribution for ξ, especially if A has small probability.]

Informally speaking, $H(\xi|A)$ is the minimal average code length if average is taken only over the cases when A happens.

Now let us consider two random variables ξ and η (as it was done in the previous section). Let as assume that each value of both ξ and η has non-zero probability (zero-probability outcomes could be ignored). For each value η_j (for η) consider the event $\eta = \eta_j$. (Its probability was denoted by p_{*j}.) Consider the conditional entropy of variable ξ having this event as the condition. In other

terms, consider the entropy of the distribution $i \mapsto p_{ij}/p_{*j}$. Then we average these entropies, using probabilities of the events $\eta = \eta_j$ as weights. The resulting average is called *conditional entropy of ξ with condition η*. It is denoted by $H(\xi|\eta)$. So by definition

$$H(\xi|\eta) = \sum_j \Pr[\eta = \eta_j] H(\xi|\eta = \eta_j)$$

or, using the notation above,

$$H(\xi|\eta) = \sum_j p_{*j} \sum_i \frac{p_{ij}}{p_{*j}} \left(-\log \frac{p_{ij}}{p_{*j}}\right).$$

The following theorem sums up the basic properties of conditional entropy (that are true for any random variables ξ and η):

{conditiona

Theorem 118 **(a)** $H(\xi|\eta) \geqslant 0$;

(b) $H(\xi|\eta) = 0$ *if and only if* $\xi = f(\eta)$ *with probability* 1 *for some function* f (*in other terms, we ignore the cases that have zero probability*).

(c) $H(\xi|\eta) \leqslant H(\xi)$

(d) $H(\langle \xi, \eta \rangle) = H(\eta) + H(\xi|\eta)$

$\lhd$ The item (a) is evident: all $H(\xi|\eta = \eta_j)$ are non-negative, so the same is true for their weighted sum.

(b) If the weighted sum of non-negative terms equals zero, then all the terms that have non-zero weights are equal to zero. So for each value η_j the restricted variable $(\xi|\eta = \eta_i)$ has zero entropy, and therefore has only one value if we ignore values that have probability 0.

The statement (c) can be explained as follows: $H(\xi|\eta)$ is the average length of an optimal code for ξ if we allow different codes for ξ for different values of η (for each value of η we consider the code that is optimal with respect to conditional distribution). This provides some additional freedom (compared to the case when the same code should be used for all values of η), and this freedom can only decrease the optimal code length.

The same argument made formal: for each j the value of $H(\xi|\eta = \eta_j)$ is the minimal value of the sum

$$\sum_i \frac{p_{ij}}{p_{*j}}(-\log q_{ij})$$

taken over all non-negative values of the variables $q_{1j} + q_{2j} + \ldots + q_{kj} = 1$ (we use different variables for each j). Therefore, $H(\xi|\eta)$ is the minimal value of the sum

$$\sum_j p_{*j} \sum_i \frac{p_{ij}}{p_{*j}}(-\log q_{ij})$$

taken over all tables that contain non-negative reals q_{ij} and each column has sum 1. If we restrict ourselves to tables where all columns are equal ($q_{ij} = q_i$), the sum turns into

$$\sum_j p_{*j} \sum_i \frac{p_{ij}}{p_{*j}}(-\log q_i) = \sum_j \sum_i p_{ij}(-\log q_i) = \sum_i p_{i*}(-\log q_i)$$

and its minimum is $H(\xi)$. Therefore $H(\xi|\eta) \leqslant H(\xi)$.

Finally, item (d) is just an exercise in transformation of logarithms:

$$\sum_{i,j} p_{ij}(-\log p_{ij}) = \sum_j p_{*j} \sum_i \frac{p_{ij}}{p_{*j}}(-\log \frac{p_{ij}}{p_{*j}} - \log p_{*j}) =$$
$$\sum_j p_{*j} \sum_i \frac{p_{ij}}{p_{*j}}(-\log \frac{p_{ij}}{p_{*j}}) + \sum_j p_{*j} \sum_i \frac{p_{ij}}{p_{*j}}(-\log p_{*j}) =$$
$$\sum_j p_{*j} H(\xi|\eta = \eta_j) + \sum_j p_{*j}(-\log p_{*j}) = H(\xi|\eta) + H(\eta).$$

Theorem is proven. ▷

This theorem implies Theorem 117 (p. 174). We see also that entropy of the pair of random variables cannot be less than the entropy of any of variables (since conditional entropy is non-negative). Thus we easily obtain the following statement:

{no-new-ent

Theorem 119 *Let ξ be a random variable with a finite range and let f be a function defined on that range. Then*

$$H(f(\xi)) \leqslant H(\xi),$$

where $f(\xi)$ is a random variable that is a composition of f and ξ (i.e., f is applied to the value of ξ).

In terms of distribution the transition from ξ to $f(\xi)$ means that we combine several values into one summing up the corresponding probabilities.

◁ Indeed, the random variable $\langle \xi, f(\xi) \rangle$ has the same distribution as ξ, and its entropy cannot be less than the entropy of the second coordinate. ▷

161 Provide an interpretation of this result in terms of minimal average length of codes, and the direct proof.

162 When the inequality of Theorem 119 becomes an equality?

7.2.3 Independence and entropy

{independer

The notion of independent random variables could be easily expressed in terms of entropy. Recall the variables ξ and η are called *independent* if the probability of the event "$\xi = \xi_i$ and $\eta = \eta_j$" is equal to the product of probabilities of the events $\xi = \xi_i$ and $\eta = \eta_j$. (A reformulation: the conditional distribution of ξ with condition $\eta = \eta_j$ coincides with the unconditional distribution. Also we can exchange ξ and η and say that conditional distribution of η with condition $\xi = \xi_i$ coincides with the unconditional distribution.)

In the notation used above the independence can be written as $p_{ij} = p_{i*}p_{*j}$ (probability matrix has rank 1).

{independer

Theorem 120 *Random variables ξ and η are independent if and only if*

$$H(\langle \xi, \eta \rangle) = H(\xi) + H(\eta).$$

In other words, we get an independence criterion: the inequality of Theorem 117 becomes an equality. Using Theorem 118, we can rewrite this criterion as $H(\xi) = H(\xi|\eta)$ (or, symmetrically, $H(\eta) = H(\eta|\xi)$).

$\lhd$ Let us use once more that logarithm is a strictly convex function: the inequality

$$\log\left(\sum p_i x_i\right) \geqslant \sum p_i \log x_i,$$

holds for all positive weights p_i with sum 1 and all positive x_i. This inequality becomes an equality only if all x_i are equal.

Therefore, for positive p_i with sum 1 the expression

$$\sum p_i(-\log q_i)$$

(where q_i are positive and sum up to 1) takes its minimal value only at the point $q_i = p_i$.

Now recall the proof of Theorem 117 above. The minimum over rank 1 matrices (that makes the right-hand side equal to the sum of entropies) was achieved for

$$q_{ij} = p_{i*} \cdot p_{*j}$$

If this minimum coincides with the minimum taken over all matrices q_{ij} (the latter is achieved for $q_{ij} = p_{ij}$), then we have

$$p_{ij} = p_{i*} \cdot p_{*j}$$

and variables ξ and η are independent. $\rhd$

163 Provide an another (though similar) proof using Theorem 118.

164 Prove that three random variables α, β, γ are independent (this means that the probability of the event $(\alpha = \alpha_i, \beta = \beta_j, \gamma = \gamma_k)$ equals the product of three probabilities for each of the variables) if and only if

$$H(\langle \alpha, \beta, \gamma \rangle) = H(\alpha) + H(\beta) + H(\gamma).$$

Theorems 117 and 120 show that the difference $H(\xi) + H(\eta) - H(\langle \xi, \eta \rangle)$ is always non-negative and equals zero if and only if ξ and η are independent. So we can take this difference for a quantitative measure of dependence between ξ and η. This difference is denoted by $I(\xi : \eta)$ and called the *mutual information* of two random variables ξ and η. Theorem 118 allows us to rewrite the definition for $I(\xi : \eta)$ in the following way:

$$I(\xi : \eta) = H(\eta) - H(\eta|\xi) = H(\xi) - H(\xi|\eta).$$

(mutual information shows how much the knowledge of one random variable decrease the entropy of the other one).

To see all these notions in action, let us return to the McMillan inequality. Now we change the order and prove first that a uniquely decodable code for a random variable ξ has the average length of the codeword at least $H(\xi)$. {macmillan-

First note that for an injective code where all codewords have length less than c the average length is at least $H(\xi) - \log c$. Indeed, if n_i are the lengths of the codewords, the sum of 2^{-n_i} does not exceed c (for every fixed length the sum does not exceed 1). Therefore, the inequality of theorem 114 is violated at most by $\log c$.

This is not enough, and to get a tight bound we consider N independent identically distributed copies of random variable ξ. We get a random variable that could be denoted by ξ^N. Its entropy is $NH(\xi)$. Let us use our code for each of N coordinates and then concatenate all the strings. The unique decoding property guarantees that this is an injective code. Its average length is N times greater than the average length of initial code for ξ (linearity of expectation). And the maximal length does not exceed cN where c is an upper bound for the length of the codewords of the uniquely decodable code we started with. So the previous paragraph gives us

$$N \cdot (\text{average length of the uniquely decodable code}) \geqslant NH(\xi) - \log(cN)$$

Now we divide over N and take $N \to \infty$. Since $\log(cN)/N \to 0$ as $N \to \infty$, this gives us the bound $H(\xi)$ for the average length of an uniquely decodable code.

Now the McMillan inequality is easy. Assume that uniquely decodable code has code-lengths $n_1, \ldots, n_k$ and $\sum 2^{-n_i} > 1$. We start with probabilities $p_i = 2^{-n_i}$ and then proportionally decrease all of them making their sum equal to 1. Consider the random variable that has the distribution p_i (obtained in this way) and its coding by means of our uniquely decodable code. The average length is $\sum p_i n_i$ which is less than $H = \sum p_i(-\log p_i)$ (recall that $n_i < -\log p_i$ since we have decreased the values p_i).

165 Look closely at this proof and trace the correspondence between it and the proof given above.

7.2.4 "Relativization" and basic inequalities

{entropy-re

All the statements about entropy have "relativized" (conditional) versions. For example, we could add some random variable α as a condition in the inequality

$$H(\langle \xi, \eta \rangle) \leqslant H(\xi) + H(\eta)$$

and get its conditional version

$$H(\langle \xi, \eta \rangle | \alpha) \leqslant H(\xi|\alpha) + H(\eta|\alpha)$$

The conditional version is an easy consequence of the unconditional one. Indeed, for each fixed value α_i of a random variable α we have

$$H(\langle \xi, \eta \rangle | \alpha = \alpha_i) \leqslant H(\xi|\alpha = \alpha_i) + H(\eta|\alpha = \alpha_i)$$

(Theorem 117 is applied to conditional distributions of ξ and η with condition $\alpha = \alpha_i$). The we sum up all these inequalities with weights $\Pr[\alpha = \alpha_i]$.

So we get a conditional inequality as a consequence of the unconditional one. Now, going in the opposite direction and using the equation

$$H(\beta|\gamma) = H(\langle\beta,\gamma\rangle) - H(\gamma),$$

we can express all conditional entropies in terms of unconditional ones.

After canceling some terms we get the following inequality:

{basic-shar

Theorem 121 (basic inequality)

$$H(\xi,\eta,\alpha) + H(\alpha) \leqslant H(\xi,\alpha) + H(\eta,\alpha)$$

We use a simplified notation and write $H(\xi,\eta,\alpha)$ instead $H(\langle\xi,\eta,\alpha\rangle)$ (or even more formal $H(\langle\langle\xi,\eta\rangle,\alpha\rangle)$).

The similar "relativization" (adding random variables as conditions) can be applied to the mutual information. For example, we can naturally define $I(\alpha:\beta|\gamma)$ as

$$H(\alpha|\gamma) + H(\beta|\gamma) - H(\langle\alpha,\beta\rangle|\gamma).$$

The basic inequality (Theorem 121) says that $I(\alpha:\beta|\gamma) \geqslant 0$ for all random variables α,β,γ.

166 Prove that $I(\langle\alpha,\beta\rangle:\gamma) \geqslant I(\alpha:\gamma)$

167 Prove that

$$I(\langle\alpha,\beta\rangle:\gamma) = I(\alpha:\gamma) + I(\beta:\gamma|\alpha).$$

If $I(\alpha:\gamma|\beta) = 0$, the random variables α and γ are called *independent relative to* β (when β is known). Experts in probability theory say in this case that α,β,γ form a *Markov chain* where the dependence between the "past" (α) and the "future" (γ) is caused only by the "current state" (β).

168 Prove that in this case $I(\alpha:\gamma) \leqslant I(\alpha:\beta)$, and therefore $I(\alpha:\gamma) \leqslant H(\beta)$.

To prove all these (and similar) statements one could use the diagrams that are similar to the diagrams for Kolmogorov complexity discussed in Chapter 2. The diagram for two variables consists of three regions. Each region carries a non-negative value. The sum of these values for two left regions is $H(\alpha)$ and for two right regions is $H(\beta)$ (see Fig.18).

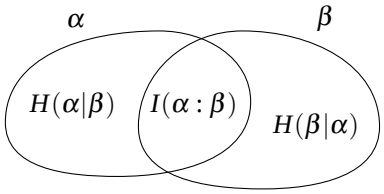

Figure 18: Entropies of two random variables.

{entropy.1}

For three variables α,β,γ we get a more complicated diagram (Fig. 19). The central region carries a number that is denoted by $I(\alpha:\beta:\gamma)$. It can be defined as $I(\alpha:\beta) - I(\alpha:\beta|\gamma)$, or,

equivalently, as $I(\alpha:\gamma)-I(\alpha:\gamma|\beta)$ etc. In terms of unconditional entropies we get the following expression:

$$I(\alpha:\beta:\gamma)=H(\alpha)+H(\beta)+H(\gamma)-H(\alpha,\beta)-H(\alpha,\gamma)-H(\beta,\gamma)+H(\alpha,\beta,\gamma)$$

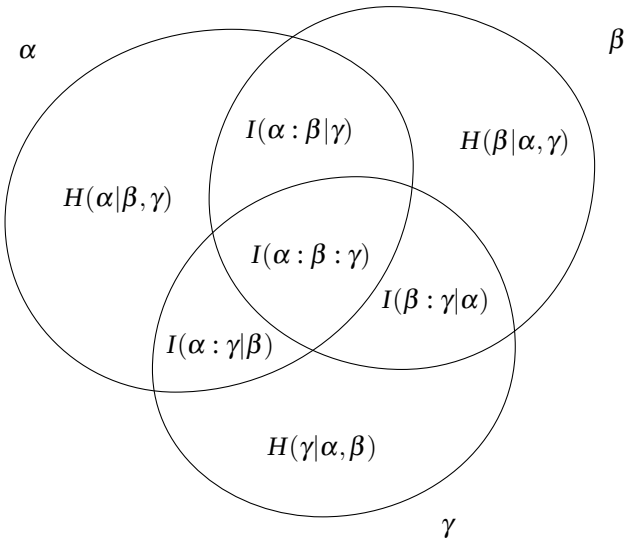

Figure 19: Entropies of three random variables. {entropy.2}

Note that (unlike other six values shown) the value of $I(\alpha:\beta:\gamma)$ can be negative. For example, this happens if variables α are β independent, but still are dependent when γ is known.

169 Construct three variables α,β,γ with this property. [Hint. Following the example given on p. 46, consider uniformly distributed independent variables α and β with range $\{0,1\}$ and let $\gamma=(\alpha+\beta) \bmod 2$.]

170 (Fano inequality) Prove that if the random variables α and β differ with probability at most $\varepsilon<1/2$, and α takes at most a values, then

$$H(\alpha|\beta)\leqslant\varepsilon\log a+h(\varepsilon),$$

where $h(\varepsilon)$ is the entropy of a random variable with two values and probabilities ε and $1-\varepsilon$. [Hint. Let γ be a random variable with two values; $\gamma=0$ when $\alpha\neq\beta$ and $\gamma=1$ when $\alpha=\beta$. Then $H(\alpha|\beta)\leqslant H(\gamma)+H(\alpha|\beta,\gamma)$. The first term is $h(\varepsilon)$, nd the second one can be rewritten as

$$\Pr[\gamma=0]H((\alpha|\beta)|\gamma=0)+\Pr[\gamma=1]H((\alpha|\beta)|\gamma=1),$$

i.e.,

$$\Pr[\alpha\neq\beta]H((\alpha|\beta)|\alpha\neq\beta)+\Pr[\alpha=\beta]H((\alpha|\beta)|\alpha=\beta),$$

which does not exceed $\varepsilon\log a+0$.]

171 Assume that $H(\alpha|\beta,\gamma)=0$ and $I(\beta:\alpha)=0$. Prove that $H(\gamma)\geqslant H(\alpha)$.

This problem has the following interpretation. If a spy wants to send to the headquarters a secret message α as a plain text β using a key γ (that is agreed in advance) and wants the adversary who does not know γ to get no information about α, then the entropy of key γ cannot be less than entropy of the message α. This statement is sometimes called *Shannon theorem on perfect cryptosystems*.

172 Prove that {condit-tri

$$2H(\alpha,\beta,\gamma) \leqslant H(\alpha,\beta)+H(\beta,\gamma)+H(\alpha,\gamma)$$

for any three random variables α,β,γ. [Hint: see the proof of the corresponding statement about Kolmogorov complexity, Theorem 26, (p. 44).]

7.3 Complexity and entropy

{complexity

As you surely have noticed, the properties of Shannon entropy (defined for random variables) resemble the properties of Kolmogorov complexity (defined for strings, see Chapter 2). Is it possible to formalize this similarity by converting it into exact statements?

This question has two interpretations. First, one can prove that Kolmogorov complexity and Shannon entropy have similar properties (in particular, the same linear inequalities are true for them, see Section **??**, p. **??**). On the other hand, one may compare the numeric values for complexity and entropy, and this is what we do in this section.

The problem here is that Kolmogorov complexity is defined for strings while Shannon entropy is defined for random variable, so how could one compare them? However, sometimes this comparison is possible, as we shall see. Let us start with a very vague and philosophical description of the results below: Shannon entropy takes into account only frequency regularities while Kolmogorov complexity takes into account all algorithmic regularities, so in general the latter is smaller. On the other hand, if an object is generated by a random process in such a way that it has only frequency regularities, entropy is close to complexity with high probability.

Let us give now some specific results that illustrate this general statement.

7.3.1 Complexity and entropy of frequencies

{frequencie

Consider an arbitrary finite alphabet A which may contain more than two letters. Kolmogorov complexity for A-strings can be defined in a natural way. (Note that we have never used that objects whose complexity is defined are *binary* strings. However, it is important that *binary* strings are considered as descriptions: complexity measured in bytes would be eight time less than complexity measured in bits!)

Let x be an A-string of length N and let $p_1,\ldots,p_k$ be the frequencies of letters in x. All these frequencies are fractions with denominator N and integer numerators. The sum of frequencies equals 1. Let $h(p_1,\ldots,p_k)$ be the Shannon entropy of corresponding distribution.

{complexity

Theorem 122

$$\frac{KS(x)}{N} \leqslant h(p_1,\ldots,p_k)+\frac{O(\log N)}{N}.$$

Here $O(\log N)$ means something that does not exceed $c \log N$, where constant c does not depend on N, x and frequencies $p_1, \ldots, p_k$. However, this constant may depend on k (we consider an alphabet of a fixed size).

$\lhd$ In fact this is a purely combinatorial statement. Indeed, $KS(x|N, p_1, \ldots, p_k)$ does not exceed $\log C(N, p_1, \ldots, p_k) + O(1)$, where

$$C(N, p_1, \ldots, p_k) = \frac{N!}{(p_1 N)!(p_2 N)! \ldots (p_k N)!}$$

is the number of A-strings of length N that have frequencies $p_1, \ldots, p_k$. (Each string with given frequencies can be determined by its ordinal number in this set if the parameters $N, p_1, \ldots, p_k$ are known, and this ordinal number has $\log C(N, p_1, \ldots, p_k)$ bits.)

The number $C(N, p_1, \ldots, p_k)$ can be estimated using Stirling's approximation. Ignoring factors bounded by a polynomial in N (that appear due to the term $\sqrt{2\pi k}$ in Stirling's approximation formula $k! \approx \sqrt{2\pi k}(k/e)^k$), we get exactly $2^{Nh(p_1,\ldots,p_k)}$. This computation was performed (for $k = 2$) when we proved the strong law of large numbers (Theorem 27, p. 51). The general case (for arbitrary k) can be treated in the same way.

Finally, note that we need about $k \log N$ bits to specify $N, p_1, \ldots, p_k$ (we need to specify k integers whose sum is N), so by deleting the condition in $KS(x|N, p_1, \ldots, p_k)$ we increase the complexity by $O(\log N)$ (and the constant in $O(\log N)$-notation is close to k). $\rhd$

Another proof uses the upper bound for monotone complexity (Theorem 81, p. 124). Consider a probability distribution on infinite A-sequences that corresponds to independent trials with probabilities $p_1, \ldots, p_k$ in each trial.

The event "a sequence with prefix z appears" where z is a A-string of length N that has frequencies $q_1, \ldots, q_k$, equals

$$p_1^{q_1 N} \ldots p_k^{q_k N}$$

(letter a_i has probability p_i and appears $q_i N$ times). The binary logarithm of this probability is equal to

$$-N \cdot (q_1(-\log p_1) + \ldots + q_k(-\log p_k)).$$

For the special case $q_i = p_i$ we get $-Nh(p_1, \ldots, p_k)$, therefore monotone complexity has upper bound $Nh(p_1, \ldots, p_k)$. (Recall also that monotone complexity differs from other complexity versions by a term $O(\log N)$ for strings of length N.)

In fact, this argument is flawed, When we proved the upper bound for monotone complexity, we have assumed that distribution is fixed. The constant term, therefore, may depend on the distribution. And now we try to estimate $KM(x)$ using measure that depends on the letter frequencies in the string x. So formally Theorem 81 is not applicable. But if we recall its proof, we see that it provides a bound for "conditional" monotone complexity when $p_1, \ldots, p_k$ are given. The difference between this conditional complexity and the unconditional one is $O(\log N)$, so we indeed get another proof for Theorem 122.

173 What is a value of a constant hidden in $O(\log N)$ (as a function of k)? [Hint: both proofs give $k(1 + o(1)) \log N$.]

174 Show that when all frequencies $p_1, \ldots, p_k$ are not very close to 0, the statement of the previous problem could be improved up to $(k/2 + O(1)) \log N$. [Hint. In the first proof one should

take into account the square roots in Stirling's approximation; most of them are in the denominator. The second proof can also be modified: instead of exact values of frequencies one can consider approximate frequencies with error of order $O(1/\sqrt{N})$. This gives a weaker bound, but the difference is bounded by a constant. (Recall that a smooth function is quadratic near its minimum.) In this way we can save half of the bits when specifying $p_1, \ldots, p_k$.]

Note that the inequality provided by Theorem 122 may be very far from equality. Indeed, if A has two letters and they alternate in a string x, then the right hand size equals 1 and the left-hand size is of order $(\log N)/N$. This is not surprising and fits well into the general picture: the complexity is small since it reflects all the regularities (not only frequencies). In the next sections we prove that the complexity of a randomly generated string is close to the entropy with high probability.

7.3.2 Expected complexity

Let us fix k, a k-letter alphabet A and k positive numbers $p_1, \ldots, p_k$ whose sum is 1 (for simplicity we assume that all p_i are rational numbers).

Consider a random variable ξ, whose values are letters of A and probabilities are $p_1, \ldots, p_k$. For each N consider a random variable ξ_N consisting of N independent identically distributed copies of ξ. Its values are A-strings of length N. Now we may ask a question: what is the expected complexity of a string generated according to this distribution?

{expected-c

Theorem 123 *The expected values of $KP(\xi^N|N)$ is $NH(\xi)+O(1)$ (the constant in $O(1)$ may depend on ξ but not on N).*

Note that (for positive p_i) all A-strings of length N are among the values of ξ^N. Some of them have complexity much greater than NH (except for the case of uniform distribution), but others have complexity much less than NH.

$\lhd$ For each A-string of length N (i.e., for any value of ξ_N) consider its shortest description (with respect to some fixed prefix decompressor). These descriptions form a prefix code (in the sense of Section 7.1.1). The average length of the codeword is exactly the expected value of $KP(\xi^N)$. Therefore Theorem 114 (p. 170) guarantees that this expected value cannot be less than $H(\xi^N) = NH(\xi)$. The lower bound is proved (and even the $O(1)$-term can be omitted).

The same theorem is useful for the upper bound, too. Indeed, it guarantees that there exist prefix codes that have average length of a codeword at most $H+1$. Such a code can be constructed by an algorithm if N (and numbers p_i, which are fixed) is given. For example, one may use the construction used in the proof of Theorem 114, or use Huffman code, or even just try all codes until a good one is found.

Anyway, the constructed code can be used as a conditional decompressor (with N as the condition) such that average length of the shortest description of ξ^N does not exceed $H(\xi^N)+1 = NH(\xi)+1$. Replacing this decompressor by an optimal one, we increase the average length by $O(1)$. $\rhd$

175 Show that one can slightly improve the upper bound and prove that the average value of monotone complexity $KM(\xi^N)$ does not exceed $NH(\xi)+O(1)$. [Hint. Apply Theorem 81 to the distribution of ξ^∞.]

We assumed that $p_1, \ldots, p_k$ are fixed rational numbers. One may wish to get a uniform bound that is true for all tuples $p_1, \ldots, p_k$. To this end we should add $p_1, \ldots, p_k$ in the condition and prove bounds for the expected value of $KP(\xi^N | N, p_1, \ldots, p_k)$ instead of $KP(\xi^N | N)$. The lower bound is not affected at all, since it is true for any prefix code, and for the code construction the information in the condition is sufficient. (We assume that p_i are rational numbers; this is not very important, since one may replace arbitrary reals by their approximations with sufficiently small error.)

176 Formulate the exact statement and prove it.

This theorem says that *average* complexity equals entropy though individual values of complexity could be much smaller or much larger. In fact, a stronger statement it true: *most* values of ξ_N have complexity close to $NH(\xi)$. More formally, the event "the complexity of string ξ^N differs significantly from $NH(\xi)$" has small probability. This statement could be considered as an algorithmic version of the Shannon theorem on (noiseless) channel capacity, and we will return to this question in Section 7.3.4.

7.3.3 Prefixes of random sequences and their complexity

{complexity

In this section we consider infinite Martin-Löf random sequences and compare complexities of their prefixes with the entropy of a generating distribution. Let A again be an alphabet that has k letters and let $p_1, \ldots, p_k$ be a probability distribution on A. We assume that $p_1, \ldots, p_k$ are computable positive reals.

Consider the space A^∞ of infinite A-sequences and the probability distribution on this space that corresponds to independent identically distributed variables with distribution $p_1, \ldots, p_k$. This is a computable probabilistic measure on A^∞, so Martin-Löf definition of randomness can be used. (In fact, we have considered two-letter alphabet, but essentially the same definition can be used for any finite alphabet.)

{complexity

Theorem 124 *Let ω be a Martin-Löf random sequence with respect to this distribution. Let $(\omega)_N$ be its prefix of length N. Then*

$$\lim \frac{KS((\omega)_N)}{N} = H,$$

where H is the Shannon entropy, i.e., $H = \sum p_i(-\log p_i)$.

177 Prove this statement for the uniform distribution this statement as an immediate consequence of the randomness criterion (Theorem 82, p. 125). (It is a rare occasion when the uniform case is really special.)

The statement refers to plain complexity KS; however, this is not important, since different versions of complexity differ only by $O(\log N) = o(N)$. So we may use monotone complexity in the proof, and this is convenient.

$\lhd$ The Schnorr–Levin randomness criterion (Theorem 82, p. 125) says that complexity of a prefix of a random sequence is close to the minus logarithm of probability that this prefix appears. The probability refers to the distribution on A^∞ considered above, and the minus logarithm equals $N\sum q_i(-\log p_i)$ where q_i is the frequency of ith letter in $(\omega)_N$. It remains to use the Strong Law of Large Numbers that guarantees that q_i tends to p_i as $N \to \infty$ for a random sequence. $\rhd$

Looking at this proof we see that the difference between the complexity (per letter) and entropy has three reasons: first, "the randomness deficiency" from Schnorr–Levin theorem that gives $O(1)/N$ difference; second, the difference between the plain and monotone complexities (of order $O(\log N/N)$) and, finally, the difference between frequencies and probabilities which is the most important term. (The law of iterated logarithm says that this leads to a difference that is a bit larger than $O(\sqrt{N})/N$.)

We have assumed that p_i are computable reals, otherwise the notion of Martin-Löf randomness cannot be used. If they are not computable, we can still consider the set of sequences such that complexity of their prefixes (per letter) do not have entropy as limit. Then we can prove that this set has measure zero (with respect to the corresponding distribution).

178 Prove this statement. [Hint. For an upper bound we can use some approximations for p_i; the precision $1/N^2$ is enough if we consider prefixes of length N. The additional information needed to specify these approximate values is of size $O(\log N)$. The lower bound does not use at all the algorithmic properties of p_i; for example, we can get a bound for relativized complexity with any oracle A that makes all p_i computable.]

7.3.4 The complexity deviation

{complexity

Theorem 124 is asymptotic. One may look for a bound of difference between complexity and entropy of frequencies for finite sequences. (This follows the example provided by the probability theory that has Strong Law of Large Numbers for the limit values as well as large deviation bounds for finite sequences.)

Let A be a k-letter alphabet and let $p_1, \ldots, p_k$ be a distribution on A. Again we assume for simplicity that p_i are rational (or at least computable). Consider the product distribution on A^N that corresponds to N independent trials with probabilities $p_1, \ldots, p_k$. So each A-string of length N has certain probability (and certain complexity). We already know from Theorem 123, that the average value of complexity is NH, where $H = \sum p_i(-\log p_i)$. But we want to know also how far this complexity deviates from its average value.

The simplest case of two equiprobable letters (which is quite untypical, as we shall see) gives a uniform distribution on all binary strings of length N. We know that all these strings have complexity at most $N + O(1)$ and the (overwhelming) majority of strings has complexity close to N: the fraction of strings that have complexity less than $N - c$ is at most 2^{-c}. So in this case the significant difference between complexity and entropy has exponentially small probability.

The case of uniform distribution on k-letter alphabet is similar. However, if not all the letters have the same probability, the situation changes significantly.

Here is the key observation. For any string x of length N we compare probabilities p_i with "empirical frequencies" $q_i(x)$ (frequencies of letters in x). It turns out that with high probability the complexity of a random (with respect to our distribution on A^N) string is close to $k(x) = N\sum_i q_i(x)(-\log p_i)$. Indeed, Theorem 81 (p. 124) says that monotone complexity can exceed $k(x)$ by at most $O(1)$. On the other hand, the argument used in the proof of Levin–Schnorr theorem (p. 125, Lemma 1) shows that for any c the probability of the event $KM(x) < k(x) - c$ (according to the distribution considered) does not exceed 2^{-c}.

Therefore, the question about the complexity reduces to the question about the distribution of empirical frequencies. This question has been studied in the probability theory for centuries. It is known (Moivre–Laplace theorem) that this distribution is close to a normal (Gaussian) one: the expectation of frequency equals the probability, and the average of the deviation square is proportional to $1/N$. This is the main term, since it is much larger than terms caused by $O(\log N)$ difference between different complexity versions and by using N as a condition, etc. This argument (made precise) gives us the proof of the following statement:

{square-roo

Theorem 125 *Let ξ be a random variable with k values. For each positive $\varepsilon > 0$ there exists c such that for all N the probability of the event $NH(\xi) - c\sqrt{N} < KS(x) < NH(\xi) + c\sqrt{N}$ is at least $1 - \varepsilon$.* (*Probability is taken over the distribution where N copies of ξ are independent.*)

In fact our arguments assumed that p_i are computable. However, this assumption can be dropped if we replace p_i by their approximations with sufficiently small error (the precision $1/N^2$ is enough and requires only $O(\log N)$ additional bits).

7.3.5 Shannon coding theorem

{shannon-co

The theorem of the last section is a natural translation of classical Shannon results into the complexity language. These results deal with the length of a code that allows us to transmit N-letter blocks with high probability (according to the given distribution).

Let ξ be (again) a random with k values (letters of A) and some fixed distribution. Let N be a positive integer. By ξ^N we denote a random variable with range A^N that is formed by N independent copies of ξ. We want to encode values of ξ^N by m-bit strings (see Figure 20):

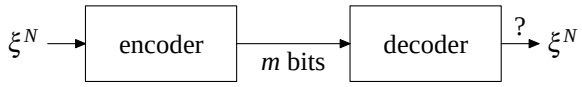

Figure 20: Using m bits for transmission of ξ^N.

{entropy.3}

Here "coder" is any mapping of type $A^N \to \mathbb{B}^m$, and "decoder" is any mapping of type $\mathbb{B}^m \to A^N$. A given value of ξ^N causes *an error* if the input and output A-strings (of length N) differ. The probability of error is measured according to the distribution of ξ^N. The question is: what conditions on m and N guarantee the existence of a code that has small error probability? First, let us make the following evident remark:

Theorem 126 *For given N, m and $\varepsilon > 0$ the code with error probability at most ε exists if and only if the 2^m most probable values of ξ^N have total probability at least $1 - \varepsilon$.*

$\lhd$ Indeed, when m bit are used for encoding, one may transmit (without errors) at most 2^m values. To minimize the error probability, we should choose 2^m most probable values. $\rhd$

In the next theorem the alphabet A and the random variable ξ are fixed.

Theorem 127 *For each $\varepsilon > 0$ there exists a constant c such that:*

(**a**) *The values of ξ^N can be encoded/decoded with $NH(\xi) + c\sqrt{N}$ bits with error probability at most ε;*

(**b**) *Any code for ξ^N of length at most $NH(\xi) - c\sqrt{N}$ has error probability at least $1 - \varepsilon$ (i.e., the probability of correct decoding is at most ε).*

$\lhd$ (a) As we know, for a suitable c the value of random variable ξ^N has complexity less than $m = NH(\xi) + c\sqrt{N}$ with probability at least $1 - \varepsilon$. So for these values one can use shortest descriptions (see the definition of plain complexity) as codes. (Formally speaking, we get strings not of length m, but of length less than m, but there are s less than 2^m of them and they can be replaced by strings of length m.)

Note that coding is not performed by an algorithm, but the theorem (as stated) does not say anything about that, it claims the existence of a code mapping.

(b) Here we need to use some trick. If there exists a code of given length, then such a code can be constructed algorithmically using the previous theorem (or just by an exhaustive search). Then the decoding function for this code can be considered as a conditional decompressor (where conditions are p_i and N). Therefore, all values of ξ^N that are decoded without error, have complexity at most $NH(\xi) - c\sqrt{N} + O(\log N)$ (the latter term corresponds to the complexity of parameters and can be omitted if we increase c). As we know (Theorem 125, p. 187), the probability of this event is at most ε. $\rhd$

179 As before, we assume that probabilities p_i are known exactly, and if p_i are not computable, we get some problems. Correct the argument replacing p_i by their approximation with sufficient precision.

180 Give a statement and proof for a similar result about conditional coding and conditional entropy. [Hint. Assume that two dependent random variables ξ and η are given. We make n trials, the value of η^N is known both to the sender and the receiver, and the sender wants to send m bits in such a way that receiver could reconstruct the value of ξ^N. How large should be m?]

7.4 Markov chains

for the viewpoint of Kolmogorov complexity and entropy (what is known here? Andrey?) lempel-ziv? how to compute the entropy of a Markov chain?

8 Some applinations

{appl}

8.1 There are infinitely many primes

{appl-prime

Let us start with a toy example and prove that there are infinitely many primes.

Assume that there are only m different prime numbers $p_1, \ldots, p_m$. Then every positive integer x has prime decomposition of the form

$$x = p_1^{k_1} p_2^{k_2} \ldots p_m^{k_m}$$

and can be described by the list of powers $k_1, \ldots, k_m$. Each of k_i does not exceed $\log x$ (since the base is at least 2) and has complexity at most $O(\log\log x)$ (its binary representation has $O(\log\log x)$ bits). Since m is fixed (i.e., the same for different x), the complexity of the tuple $\langle k_1, k_2, \ldots, k_m\rangle$ is $O(\log\log x)$ and therefore the complexity of x (that can be obtained from that tuple) is $O(\log\log x)$. But for a "random" (incompressible) $n$bit integer x the complexity is close to n and is not $O(\log n)$ as this formula says (the logarithm of a n-bit number does not exceed n). Euclid's theorem is proven.

What should one say about this argument? It is a real application of Kolmogorov complexity or just cheating? A skeptical observer would say that we just retell some counting argument in terms of Kolmogorov complexity. This counting argument is as follows: if there are only m prime numbers, then there are t most $(\log x)^m$ different integers between 1 and x, since any integer in this range is determined by the m powers in its decomposition, and each power is less than $\log x$. This immediately leads to a contradiction, since $x > (\log x)^m$ for large x.

This argument is indeed true: our reasoning using Kolmogorov complexity is a direct translation of this argument (and is a bit more cumbersome due to asymptotic notation). However, such a translation may still have sense, since new language provokes new intuition, and this intuition may be useful even if later the same argument can be translated into the standard language.

We return to this discussion after looking at other applications.

8.2 Moving information along the tape

{appl-tape}

The other toy example is a well knows result saying that duplication of a $n$bit string on the tape of a Turing machine (with one tape only) requires εn^2 steps in the worst case. This classical result was obtained in 1960ies using the so-called "crossing sequences"; our proof is just a translation of this argument into the language of Kolmogorov complexity. (We assume that the reader is familiar with the basic notions related to Turing machines, see, e.g., [**?**]).

Consider a zone of size b on a tape of an one-tape Turing machine; this zone is considered as a "buffer" and we want to transmit information through this zone, say, from left (L) to right (R), see Figure 21.

Initially the buffer zone and R are empty (filled with blanks) are empty, and L is arbitrary. We want to give an upper bound for the complexity of R after t steps. The upper bound is $(t\log m)/b + O(\log t)$ where m is the number of states that our Turing machine has and b is the width of the buffer zone. Informally the argument is quite simple: each state of the TM carries $\log m$ bits of

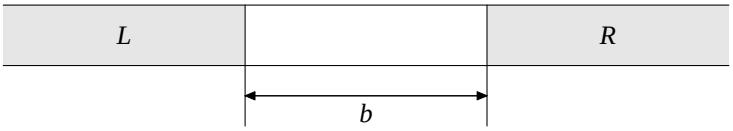

Figure 21: A buffer zone of size b.

{tape-buffe

information, and during one computation step this information can be moved to the neighbor sell, so moving it at the distance b requires b more time.

Now we have to convert this intuitive explanation into a formal argument.

Theorem 128 *Let M be a Turing machine that has m states. Then there exists a constant c such that for any b and for any computation that starts with empty buffer zone of size b and empty tape on the right of the buffer zone the complexity of the contents $R(t)$ of the right part of the tape after t steps of computation does not exceed*

$$\frac{t\log m}{b} + 4\log t + c.$$

$\lhd$ Let us consider some line between cell inside the buffer zone as a "border", and let us write down the state of M when it crosses the border from left to right (as it was done in the times of iron curtain). The sequence of states is called the *crossing sequence*. Knowing the crossing sequence, we can reconstruct the behavior of M "abroad" (on the right of the border) not using the contents of the tape on the left. Indeed, we should artificially put the machine into the first state of the crossing sequence and let it go abroad. When M returns back, we put it in the second state of the crossing sequence and let it go abroad again. In this way we correctly reconstruct the abroad behavior of the machine (since it does not remember anything except its state when crossing the border). In particular, at some moment t' the tape on the right of the buffer zone contains $R(t)$. Note that t' may be different from t since we do not take into account the time M spends on the left of the border, but t' cannot exceed t. Therefore, to reconstruct $R(t)$ we need to now the crossing sequence, t' and the distance between the border and R-zone. So there exists a constant c (depending on M but not on b and t) such that for any crossing sequence S and any b and t we have

$$KS\,(R(t)) \leqslant l(S)\log m + 4\log t + c.$$

Here we multiply the length $l(S)$ of the crossing sequence by $\log m$ since S is a string in a m-letter alphabet and each letter carries $\log m$ bits. To add b' and t' in a self-delimiting encoding we need at most $2\log b + 2\log t$ bits. We may assume that $t > b$, otherwise $R(t)$ is empty since the head never visited R. The constant c appears when we switch to the optimal decompressor.

This inequality is true for any contents of L and for any placement of the border. Now if for a given contents of L we consider the shortest crossing sequence, the length of this sequence is less then t/b (there is $b+1$ possible positions of the border, and at each step only one of the positions is crossed, so the sum of the lengths of crossing sequences does not exceed t). In this way we get the inequality stated by the theorem. $\rhd$

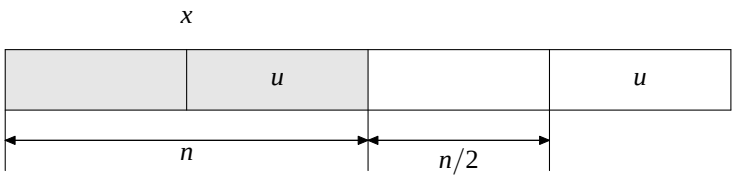

Figure 22: Buffer zone for duplication {tape-copy-

181 Show that this bound can b e improved by replacing b in the denominator by $2b$. [Hint: The return trips need almost the same time (the difference is at most b).]

The quadratic lower bound for the duplication of a n-bit string immediately follows.

Assume that a one-tape Turing machine M duplicates its input: if initially the tape contains a binary string x (followed by blanks), at the end of the computation the tape has a second copy of x (i.e., contains xx).

Theorem 129 *There exists a constant $\varepsilon > 0$ such that for every n there exists a n-bit string that requires at least εn^2 steps to duplicate it.*

$\lhd$ For simplicity let us assume that n is even, and let x be a string whose second half u has complexity close to its length (i.e., to $n/2$). Then apply the inequality we have proven considering the zone of size $n/2$ on the right of x as the buffer (Figure 22).

Assume that duplication takes t steps. Then the complexity of R zone after t steps (which is at least $n/2$) does not exceed $t\log m/b + 4\log t +$, where b is the size of the buffer zone, i.e., $n/2$. Therefore,

$$\frac{n}{2} \leqslant \frac{t\log m}{n/2} + 4\log t + c,$$

We may assume without loss of generality that $t < n^2$ (otherwise the statement is trivial). Then we replace $4\log t$ by $8\log n$ and conclude that

$$t \geqslant \frac{n^2}{4\log m} - O(n\log n);$$

the second term is small compared to the first one when n for large n (we may then formally extent the result to every n by decreasing the coefficient ε). $\rhd$

Is the Kolmogorov complexity essential in this proof? The skeptical observer may say again that we in fact just counted the number of different strings that can be copied in a limited time (using the fact that different string should have different crossing sequences, otherwise the behavior of the machine at the right of the boundary would be identical). Indeed, the original proof follows this scheme (in fact, it deals with palindrome recognition, not the duplication, but the technique is the same). Does the language of complexity make the proof more intuitive and easy to understand? Probably this is a matter of taste.

Many bounds in the computational complexity theory can be proven in the same way, using the string of maximal complexity as the "worst case" and proving that the violation of the bound

would imply this string to be compressible. Many applications of this type (and further references) are given in the classical textbook [**?**]; its authors, Ming Li and Paul Vitanyi, played an important role in development of this approach, called "incompressibility method". Note that in many cases the historically first proof was obtained using Kolmogorov complexity.

In the next section we consider one more application of the incompressibility method. Then we switch to other applications. The most interesting thing in these applications is not the statements in itself but the various methods that allow us to apply Kolmogorov complexity to prove statements that do not mention it.

8.3 Finite automata with several heads

A *finite automaton with k heads* is similar to the ordinary one (we assume that the reader is acquainted with basic notions related to finite automata, see, e.g., [**?**]) but has k one-way read-only heads. Here "one-way" means that the head can only move from left to right.

Initially all k heads observe the leftmost character of the input string. At each step the behavior of the automaton is determined by its state and k symbols it observes (under k heads): automaton changes the state and instructs some heads (at least one) to move to the right. Then the automaton performs the next step, etc.

The input string is followed by a special marker; the automaton terminates if all the heads observe this marker. (We assume that the head that sees the marker does not move to the right.) Automaton *accepts* the string if it gets into an *accepting* state after processing this string. We say that automaton *recognizes* the set of all accepted strings.

Example. Consider the language (=set of strings) $x\#x$ where x is any binary string. It is well known that this language cannot be recognized by a standard (one-head) automation. However, it is easily recognized by a two-head automaton. Indeed, we should send one head to look for the separator #, when the separator is found, two heads move synchronously and check that they read the same symbol.

So two heads are better than one (more languages can be recognized). It turns out that the same is true for more heads: $k+1$ heads are (strictly) better than k heads.

Theorem 130 *For every k there exists a language that can be recognized by a* $(k+1)$*-head automaton but not by a k-head one.*

$\lhd$ For each $m \geqslant 1$ consider th language L_m that consists of all strings

$$w_1\# \ldots w_m\# w_m\# \ldots w_1$$

(for any binary strings $w_1, \ldots, w_m$). Each w_i is repeated twice, and in the right half the strings w_i go in the reversed order (this is crucial for the argument).

A k-head automaton can recognize this language as follows: one of the heads goes to the right half, and remaining $k-1$ heads are placed before $w_1, \ldots, w_{k-1}$. Then each of these $k-1$ heads checks its string while the first head passes by its copy. After that the first $k-1$ strings are checked, the first head is of no use (it is at the end of the input string), but remaining $k-1$ heads are useful since they are on the left of the remaining strings $w_k, w_{k+1}, \ldots$. Now we repeat the same trick: one

of $k-1$ heads is sent across the right half, $k-2$ check next $k-2$ strings etc. Repeating this, we can check

$$(k-1)+(k-2)+\ldots+1=\frac{k(k-1)}{2}=C_k^2$$

string. (Note that m is fixed, so the search for a substring with a given number the finite memory is enough.)

Therefore, *the language L_m can be recognized by a k-head automaton if $m \leqslant C_k^2$.*

It remains to show that *if $m > C_k^2$, the language L_m cannot be recognized by a k-head automaton.* Assume that is not the case and some k-head automaton M recognizes this language. To get a contradiction, let us consider independent random string $w_1, \ldots, w_m$ of sufficiently large length N. More formally, consider a string of length mN and complexity at least mN and split it into m strings of length N denoted by $w_1, \ldots, w_m$. By assumption, the string

$$W = w_1 \# \ldots w_m \# w_m \# \ldots w_1$$

is accepted by M; we get a contradiction by showing that either $w_1 \ldots w_m$ is compressible or the automaton does not recognize L_m.

Let us say that a given pair of heads of M *visited* w_i if at some moment (while processing W by M) these heads were simultaneously inside two copies of w_i. A key observation: *a given pair of heads cannot visit both w_i and w_j for $i \neq j$.* Indeed, consider the moment when w_i was visited. After that the left heads reads only w_j with $j > i$ and the right head visits only w_j with $j < i$.

By our assumption $m > C_k^2$; therefore there exists i such that w_i is not visited by any pair of heads. Let us show that either this string is compressible or one of its copies can be counterfeited in such a way that M will still accept the string (so M does not work correctly).

Let us observe the actions of M on W. A special attention is needed when one of the heads enters or leaves w_i (any of two copies): we write down the positions of all heads and the state of M at these moments. The obtained "log-file" P has complexity $O(\log N)$ where the hidden constant depends on k, m and the number of states in M but not on N. Indeed, there are at most $4k$ moments to consider (4 per head) and at each moment we record the state of the automaton and head positions, which is $O(\log N)$ bits.

Let us show that (if M recognizes L_m correctly) the string w_i can be uniquely reconstructed if all other w_j (with $j \neq i$) and P are given. This implies that the complexity of the string $w_1 \ldots w_m$ does not exceed $(m-1)N$ (the number of bits in other w_j) plus $O(\log N)$ (the complexity of protocol) plus $O(1)$, which is less than mN for large N, so we get a desired contradiction.

The reconstruction goes as follows: we place all strings of length m in place of w_i (keeping w_j with $j \neq i$ intact). For each candidate we run M on the resulting string and check whether we get the same protocol P. There are three possible cases:

(1) If (for some w) M rejects (does not accept) the string, then M does not recognize our language.

(2) M accepts all these strings (for all candidates) and the protocol P appears only once, for $w = w_i$. Then the reconstruction is possible (and $w_1 \ldots w_m$ is compressible).

(3) M accepts all these strings and P appears both for w_i and for some $w \neq w_i$. Let us show that in this case M accepts a string not in L_m, i.e., the string W' that has w_i in the left half while in the right half w_i is replaced by w.

Indeed, the are two accepting computation of M: one if w_i is used on both sides and the other one for w. Let us split both of them into parts; the splitting points are moments when one of the head enters or leaves w_i (or w). The positions of all other heads and the states of M are recorded in P so they are the same for both computations. (Note that the moments of time can be different since they are not recorded. In fact, we may add them also, but this is not needed.) So we can glue the computation intervals for both cases; let us show that we can get an accepting computation of M on a bad string (the left half has w_i while the right half has w).

By our assumption during the processing of W there is no moment when both copies of w_i carry some heads; since the border crossings for both copies is recorded in P, the same is true when w_i is replaced by w. So for each interval between two protocol events related to w_i/w there are three possibilities: (a) there is a head in the ith string on the left; (b) there is a head in the ith string on the right; (c) none of the above. Then we can copy-paste the intervals into a new computation: for (a)-parts we use the computation of M on W; for b-parts we use the computation of M of changed input (where w_i is replaced by w); for (c)-parts we can use either of two (they are the same). Then we get a computation of M on a mixed string W', so M does not work properly. $\triangleright$

8.4 Laws of Large Numbers

{appl-lln}

The Strong Law of Large Numbers was proven in Section 3.2 (Theorem 27, p. 51) without any references to Kolmogorov complexity, by a straightforward counting. We consider (mainly) the uniform case. In the case the SLLN says that the set of all sequences $\omega = \omega_0\omega_1\ldots$, such that the sequence

$$p_n = \frac{\omega_0 + \omega_1 + \ldots + \omega_{n-1}}{n}$$

has limit $1/2$ as n tends to infinity, has full measure (with respect to the uniform Bernoulli measure on Ω). In other words, SLLN says that the complement of this set (i.e., the set of sequences ω such that p_n either has no limit or has limit not equal to $1/2$) is a null set. Later (Theorem 32 (p. 60) we have shown that this null set is in fact an effectively null set; this implies that for any ML-random (with respect to the uniform measure) sequence ω the sequence p_n converges to $1/2$ (Theorem 33, p. 60).

However, we can go in the other direction. Namely, we may first prove that for any ML-random sequence the frequencies converge to $1/2$ using the randomness criterion in terms of complexity (Theorem 82, p. 125). This criterion says that for a ML-random (with respect to the uniform Bernoulli measure) sequence ω the monotone complexity of its prefix $(\omega)_n$ of length n is $n + O(1)$. This property implies that the frequency of 1s in $(\omega)_n$ (i.e., p_n) converges to $1/2$. Indeed, Theorem 122 says that the complexity of ω_n does not exceed $nh(p_n, 1-p_n) + O(\log n)$, so $h(p_n, 1-p_n) = 1 + O(\log n/n)$ for any ML-random sequence. (Note that the difference between plain and prefix complexity of ω_n is $O(n)$, so any of them can be used.) This implies that $p_n \to 1/2$ as $n \to infty$ (see the graph of entropy function, Figure 8, p. 52). So the SLLN is true for all ML-random sequence, which form a set of full measure.

The skeptical observer would say that this is not a different proof, or we have just repeated the same arguments using different language. And he is probably right: If we recall the proof of Theorem 122, we see that it uses the same estimate (based on Stirling's approximation) that was

used for the proof of SLLN. (Another argument, where monotone complexity is bounded by a negative logarithm of the measure, Theorem 81, also has a direct translation in the probabilistic language; it was discussed in Section 3.2 after the proof of Theorem 27 on p. 51).

So why do we get by using the complexity language? First, we can find a broader class of sequences that satisfy SLLN:

{lln-comple

Theorem 131 *Let ω be a binary sequence such that $KS((\omega)_n) = n + o(n)$. Then the sequence p_n (frequency of ones in $(\omega)_n$) converges to $1/2$.*

◁ The proof remains essentially unchanged: in this case $h(p_n, 1-p_n)$ is still $1+o(1)$. ▷

Second, we can not only prove that $p_n \to 1/2$ but also give some estimates for the convergence speed. The corresponding result in probability theory is called the *Law of Iterated Logarithm*, and Kolmogorov complexity can be used to give a (rather simple) proof of the upper bound provided by this law.

{iterated-l

Theorem 132 *Let ω— be a ML-random sequence with respect to the uniform measure. Let p_n be the frequency of ones in $(\omega)n$. Then for any $\varepsilon > 0$ the inequality*

$$|p_n - 1/2| \leqslant (1+\varepsilon)\sqrt{\frac{\ln\ln n}{2n}}$$

holds for any sufficiently large n.

◁ Let us look which bound is obtained by the argument above (that uses Kolmogorov complexity). We know that

$$n - O(1) \leqslant KM((\omega)_n) \leqslant nh(p_n, 1-p_n) + O(\log n),$$

therefore

$$h(p_n, 1-p_n) \geqslant 1 - O(\log n/n)$$

The function

$$p \mapsto h(p, 1-p) = p(-\log p) + (1-p)(-\log(1-p))$$

has maximum at $p = 1/2$, and the second derivative at this point is non-zero (equals $-4/\ln 2$). Therefore, Taylor expansion gives us

$$h(1/2+\delta) = 1 - \frac{2}{\ln 2}\delta^2 + o(\delta^2)$$

as $\delta \to 0$, and for $\delta_n = p_n - 1/2$ we have

$$\delta_n^2 = O(\log n/n),$$

i.e.,

$$|p_n - 1/2| = O\left(\sqrt{\frac{\log n}{n}}\right).$$

So we get at least something, though the bound we need is much stronger. (Let us mention that in the probability theorem the final bound was obtained in many steps. First Hausdorff (1913) has proven the bound $O(n^\varepsilon/\sqrt{n})$; then Hardy and Littlewood (1914) have improved it to $\sqrt{\log n}$; then Steinhaus (1922) came with the bound $(1+\varepsilon)\sqrt{(2\ln n)/n}$, and only later Khinchin (1924) got the final result. So we are now on the level of Hardy and Littlewood in this respect — not that bad.)

Let us think about possible improvements for the upper bound that we had for $KM((\omega)_n)$. This upper bound was obtained by comparing $KM((\omega)_n)$ and the negative logarithm of the probability of the prefix $(\omega)_n$ with respect to the Bernoulli measure with parameter p_n. This logarithm is exactly $nh(p_n, 1-p_n)$, but the Bernoulli measure used for comparison depends on n, so the construction used in the proof of Theorem 81 needs an additional term that is $KP(p_n)$ (we start with a self-delimiting encoding of p_n). Here $KP(p_n)$ does not exceed $(2+\varepsilon)\log n$, since both numerator and denominator of the fraction p_n do not exceed n. Altogether we get the bound

$$\frac{2}{\ln 2}(p_n - 1/2)^2 \approx 1 - h(p_n, 1-p_n) \leqslant (2+\varepsilon)\log n/n,$$

which is still not good enough.

What else can we do? Note that we may already know that p_n is rather close to $1/2$: with denominator n the numerators differs from $n/2$ by $\sqrt{n}$ or a bit more. So (when the denominator n is knows) we can use less bits to describe the numeration, and this allows us to replace 2 by 1.5 in the right-hand side. But this is still not enough for us.

The crucial idea is to use approximations for p_n. Let us assume that $p_n = 1/2 + \delta_n > 1/2$. Instead of p_n we use (while constructing the Bernoulli measure used to get an upper bound for complexity) its approximation $1/2 + \delta'_n$ where δ'_n is an approximation to δ_n from below with a small (fixed) relative error. For example, let us take δ'_n such that $0{,}9\delta_n < \delta'_n \leqslant \delta_n$. Such a δ'_n can be founded among the geometric sequence $(0{,}9)^k$, and its complexity is about $\log k$, i.e., about $\log(-\log\delta_n/\log 0{,}9) = \log(-\log\delta_n) + c$. Note that if $\delta_n < 1/\sqrt{n}$ then we have nothing to prove, do the complexity of δ'_n can be upper-bounded by $(1+\varepsilon)\log\log n$ (for every ε this bound holds for all sufficiently large n).

This is good news; the bad news is that we have a more complicated bound for the complexity of $(\omega)_n$. Now instead of $h(p_n, 1-p_n)$ we have

$$p_n[-\log p'_n] + (1-p_n)[-\log(1-p'_n)], \qquad (*)$$

where $p'_n = 1/2 + \delta'_n$; recalling our discussion of entropy, we may say that a sequence $(\omega)_n$ where frequencies of zeros and ones are p_n and $1-p_n$ is encode by a code that is based on simplified frequencies p'_n and $1-p'_n$. The expression $(*)$ can only increase if we replace p_n by p'_n: since $p_n > p'_n > 1/2$, the second expression in brackets is greater than the first one, and increasing its weight by decreasing p_n increases the entire expression $(*)$.

Finally we get the bound

$$n - O(1) \leqslant nh(p'_n, 1-p'_n) + (1+\varepsilon)\log\log n$$

for every $\varepsilon > 0$ (the inequality holds for all sufficiently large n). As before, it implies

$$\delta'_n \leqslant (1+\varepsilon)\sqrt{\ln 2 \cdot \log\log n/2n}.$$

For a "true" δ_n we get a slightly bigger bound ($1/0.9$ times bigger); since 0.9 can be replaced by any number less than 1 we get the desired statement (the factor $\ln 2$ is used to convert the binary logarithm to the natural one, while the replacement of the second binary logarithm by the natural one can be compensated by a change of ε in the factor $(1+\varepsilon)$). $\triangleright$

182 Show that this argument can be used to prove the statement of Theorem 132 not only for ML-random sequence but for every sequence ω such that $n - KM((\omega)_n) = o(\log\log n)$.

8.5 Forbidden substrings

{appl-llll}

The statement we prove in this section is interesting as an example of a non-trivial application of Kolmogorov complexity (that cannot be directly translated into a counting argument).

{no-forbidd

Theorem 133 *Let $\alpha < 1$ be a positive real numbers. Assume that for each n some binary strings are called* forbidden *strings and there are at most $2^{\alpha n}$ forbidden strings for any length n. Then there exists some c and an infinite sequence of zeros and ones that does not have forbidden substrings of length c or more.*

For example, we can declare strings of length n and (plain) complexity less than αn as forbidden strings. Then we get the following corollary:

{no-simple-

Theorem 134 *Let $\alpha < 1$ be a positive real number. There exists an infinite sequence of zeros and ones such that any its substring of sufficiently large length n has complexity at least αn.*

It is instructive to compare this statement with the randomness criterion for the uniform measure (Theorem 86, p. 129). In this criterion we considered only the prefixes of the sequence (instead of all substrings); on the other hand the lower bound for complexity was $n - O(1)$ instead of a weaker bound αn that we have now. (The bound $n - O(1)$ was for the monotone complexity; it implies $n - O(\log n)$ bound for plain complexity that we use now). The following problem shows that such a strong bound cannot be true for all the substrings.

183 For any infinite sequence ω of zeros and ones there exist $\alpha < 1$ and infinitely many substrings that have complexity per letter (the ratio complexity/length) at most α. [Hint: Consider two cases: if the string has *all* binary strings as substring, the claim is evident. If it does not contain some string u of length k, we can split long substrings into blocks of length k and use efficient coding that takes into account that block u is never used and does not need a code; this gives complexity per letter at most $(\log(2^k - 1))/k$.]

The proof of Theorem 133 goes in two steps. First we prove its special case, Theorem 134. Then it turns out (surprisingly) that the general case follows from this special one.

$\triangleleft$ To prove Theorem 134 let us consider an intermediate β such that $\alpha < \beta < 1$. Using Theorem 65 (p. 102) we find a number N with the following property: to each string x we can append N bits (on the right) in such a way that prefix complexity of the string increases at least by βN.

Let us use this property iteratively starting from the empty string. We get an infinite sequence of N-bit blocks; the prefix complexity increases at least by βN when the next block is appended.

This implies that the complexity of any group of consecutive k blocks is at least $\beta kN - O(1)$. Indeed, appending this group we increase complexity by βkN at least, but the inequality $KP(xy) \leqslant KP(x) + KP(y) + O(1)$ shows that $KP(y) \geqslant KP(xy) - KP(x) - O(1)$.

This implies that for every substring u (not necessarily block-aligned) the complexity of u is at least $\beta l(u) - 1$ since the change in complexity and length due to boundary effects (by cutting the incomplete block on the border) is $O(1)$. It remains to note that we have some reserve due to the difference between α and β, and this reserve is enough to compensate both the boundary effects and the difference between plain and prefix complexities. $\triangleright$

184 Give a similar argument that uses plain complexity instead of prefix one. [Hint: Use Problem 34, p. 38.]

$\triangleleft$ Now let us prove Theorem 133; the simplest approach in to use relativized complexity. Let us consider the set F of forbidden strings as an oracle; this means that we consider algorithms that can ask (for free) whether a given string is forbidden or not. As usually, this relativization goes smoothly both in the statement of Theorem 134 and its proof, and this theorem is true for F-relativized complexity.

Note that now all forbidden strings of length n have F-complexity at most $\alpha n + O(\log n)$, since each forbidden string can be determined by n and by its ordinal number in the list of all forbidden strings of length n. In fact the stronger bound $\alpha n + O(1)$ is valid since we can use the list of all forbidden strings in the order of increasing length, but this does not matter much since a small change in α covers this difference. $\triangleright$

One can also make the following (rather unexpected) observation: Theorem 133 can be derived from Theorem 134 directly, without any relativization, by using the following statement:

Theorem 135 *If for some rational α and some set F of forbidden strings the statement of Theorem 133 is false* (F *has less than $2^{\alpha n}$ forbidden strings for any n, but there is no infinite sequence without long forbidden strings*)*, then the same happens for some* decidable *set F.*

(Note that for a decidable F the relativization does not change anything; the restriction to rational α is also not important, since we can increase α to a greater rational number.)

$\triangleleft$ Assume that for some $\alpha < 1$ and some set F the statement of Theorem 133 is false. Then for each c we may find a set F_c in such a way that

(a) F_c contains only string of length greater than c;

(b) F_c contains at most $2^{\alpha k}$ strings of length k (for every k);

(c) any infinite sequence contains at least one substring that belongs to F_c.

(Indeed, we can let F_c be the set of all strings in F that have length greater than c.)

The standard argument (compactness, König's lemma) shows that any sufficiently long string has at least one substring in F_c, so one can find *finite* F_c with the same properties. Moreover, such a finite set can be found by an exhaustive search, so we get F_c that has these properties and can be found effectively when c is given.

(Why do we need first switch to finite sets? to make the search possible.)

Now we construct the sequence c_i such that c_{i+1} is greater than the lengths of all strings in F_{c_i}. The union of all F_{c_i} is a decidable set that violates the statement of Theorem 133. $\triangleright$

Note the structure of our arguments: knowing that object with some property exists, we perform an exhaustive search and effectively find (may be, different) object with the same property. This observation is often useful when dealing with Kolmogorov complexity.

[Here the argument in the reverse direction can be added: LLLL, application to twodimensional sequence, the corollary about subsequences (Rumyantsev)]

8.6 A proof of an inequality

As we have said (see p. 15), the inequalities for Kolmogorov complexity have quite unexpected consequences. In this section we explain one of them (this topic will be continued in Chapter **??**).

{triple-fur

Theorem 136 *Let X, Y, and Z be finite sets. Let $f\colon X\times Y\to\mathbb{R}$, $g\colon Y\times Z\to\mathbb{R}$, and $h\colon X\times Z\to\mathbb{R}$ be functions with non-negative values. Then*

$$\left(\sum_{x,y,z} f(x,y)g(y,z)h(x,z)\right)^2 \leqslant \left(\sum_{x,y} f^2(x,y)\right)\cdot\left(\sum_{y,z} g^2(y,z)\right)\cdot\left(\sum_{x,z} h^2(x,z)\right)$$

$\lhd$ It looks very suspicious, but this inequality in fact is a corollary of the inequality

$$2\,KP\,(x,y,z) \leqslant KP\,(x,y)+KP\,(y,z)+KP\,(x,z)+O(\log n)$$

for prefix complexity (Theorem 26, p. 44). We wrote the last inequality for prefix complexity, not the plain one, but this does not matter since the difference is $O(\log n)$. (For prefix complexity this inequality is true up to $O(1)$-precision, see Problem 84, p. 101; for now the $O(\log n)$-precision is enough.)

It is convenient to assume that elements of the finite sets X, Y, Z are binary strings. It is enough to show that if the sums in the right-hand side of the inequality do not exceed 1, the same is true for the left-hand side. (Indeed we can multiply f by any constant c, and both sides of the inequality are multiplied by the same factor, so we can "normalize" f; the same for g and h.)

Now assume that $\sum_{x,y} f^2(x,y)=1$ and that the same is true for two other sums. We have to show that $\sum_{x,y,z} f(x,y)g(y,z)h(x,z)\leqslant 1$.

The idea is simple: the function f^2 is a probability distribution on pairs (x,y), so $KP\,(x,y)\leqslant -\log f^2(x,y) = -2\log f(x,y)$ (we temporarily ignore the constant in the comparison of this distribution and the a priori one). Similarly, $KP\,(y,z)\leqslant -2\log g(y,z)$ and $KP\,(x,z)\leqslant -2\log h(x,z)$. Then we apply the inequality for $KP\,(x,y,z)$ (temporarily ignoring the logarithmic term) and get

$$KP\,(x,y,z)\leqslant -\log f(x,y)-\log g(y,z)-\log h(x,z).$$

Since the sum of $2^{-KP\,(x,y,z)}$ over all triples x,y,z does not exceed 1 (Theorem 51, p. 82), we get the desired inequality.

This argument is, of course, too simple to be valid: all our bounds are of asymptotic nature so we have to switch somehow from individual strings to sequences of strings. Let us show how it can be done.

We start with a simple remark: it is enough to prove the inequality for functions f, g, h with rational values (by continuity argument).

Let N be an arbitrary natural number (later we take the limits as N tends to infinity). Consider the sets X^N, Y^N, and Z^N whose elements are N-tuples (of elements of X, Y, Z respectively). Consider a probability distribution on $X^N \times Y^N = (X \times Y)^N$ that corresponds to N independent copies of distribution f^2 on $X \times Y$. (Formally speaking, the probability of a point $\langle\langle x_1,\ldots,x_N\rangle,\langle y_1,\ldots,y_N\rangle\rangle$ is the product $f^2(x_1,y_1)\ldots f^2(x_N,y_N)$.) We get a family of distributions that computably depends on N. Therefore, there exists a constant c such that

$$KP\left(\langle x_1,\ldots,x_N\rangle,\langle y_1,\ldots,y_N\rangle|N\right) \leqslant 2\sum_i(-\log f(x_i,y_i))+c$$

for all N and foe all $x_1,\ldots,x_N,y_1,\ldots,y_N$ (we compare our distribution with a priori probability). We can delete the condition N in the left-hand side replacing c by $c\log n$ in the right-hand side. Then (as before) we add three inequalities if this type and apply the inequality for complexities. Then we get

$$\begin{aligned}KP\left(\langle x_1,\ldots,x_N\rangle,\langle y_1,\ldots,y_N\rangle,\langle z_1,\ldots,z_N\rangle\right) \leqslant \\ \leqslant \sum_i(-\log f(x_i,y_i))+\sum_i(-\log g(y_i,z_i))+\sum_i(-\log h(x_i,z_i))+c\log N\end{aligned}$$

for some constant c and for all N, $x_1,\ldots,x_N$, $y_1,\ldots,y_N$, $z_1,\ldots,z_N$ (note that total length of all the strings x_i,y_i,z_i for $i=1,\ldots,N$ is $O(N)$, so all logarithmic terms are absorbed by $c\log N$). Combining this bound with the inequality $\sum_u 2^{-KP(u)} \leqslant 1$, we conclude that for every N the sum

$$\sum\prod_i f(x_i,y_i)g(y_i,z_i)h(x_i,z_i)$$

(over all tuples $x_1,\ldots,x_N$, $y_1,\ldots,y_N$, $z_1,\ldots,z_N$) does not exceed $2^{O(\log N)}$, i.e., is bounded by a polynomial in N. But this sum is Nth power of the sum

$$\sum_{\langle x,y,z\rangle\in X\times Y\times Z} f(x,y)g(y,z)h(x,z),$$

so the polynomial growth is possible only is the latter sum does not exceed 1, and this ends the proof. $\triangleright$

185 Show that this inequality implies the bound for the volume of a three-dimensional body in terms of its two-dimensional projections mentioned on p. 15. [Hint: we can let f,g,h be the characteristic functions of the projections. This works for the discrete case; for the continuous case we should either approximate the body using a cube grids or approximate the integral by finite sums.]

For comparison let us give two other proofs of the same inequality. Here is the first one (rather simple) that uses Cauchy inequality that says that $(u,v)^2 \leqslant \|u\|^2\cdot\|v\|^2$, or, in coordinates,

$(\sum u_i v_i)^2 \leqslant (\sum u_i^2)(\sum v_i^2)$). We can argue as follows:

$$\left(\sum_{x,y,z} f(x,y)g(y,z)h(x,z)\right)^2 \leqslant$$

$$\leqslant \left(\sum_{x,y} f^2(x,y)\right)\left(\sum_{x,y}\left(\sum_z g(y,z)h(x,z)\right)^2\right) \leqslant$$

$$\leqslant \left(\sum_{x,y} f^2(x,y)\right)\sum_{x,y}\left(\left(\sum_z g^2(y,z)\right)\left(\sum_z h^2(x,z)\right)\right) =$$

$$= \left(\sum_{x,y} f^2(x,y)\right)\left(\sum_{y,z} g^2(y,z)\right)\left(\sum_{x,z} h^2(x,z)\right)$$

Another proof uses Shannon entropy (and can be considered as a translation of Kolmogorov complexity argument into the probabilistic version). Assume that $\sum f^2 = \sum g^2 = \sum h^2 = 1$. We want to prove that $\sum_{x,y,z} p(x,y,z) \leqslant 1$, where $p(x,y,z) = f(x,y)g(y,z)h(x,z)$. Assume that is not the case and this sum equals $c > 1$. Then we can multiply it by $1/c$ and get a probability distribution p' on $X \times Y \times Z$:

$$p'(x,y,z) = \frac{1}{c} f(x,y)g(y,z)h(x,z).$$

The corresponding random variable (whose range is $X \times Y \times Z$) is denoted by ξ. It can be considered as a triple of (dependent) random variables ξ_x, ξ_y, ξ_z. One can also consider the joint distributions $\xi_{xy} = \langle \xi_x, \xi_y \rangle$ etc. For example, the random variable ξ_{xy} takes value $\langle x,y \rangle$ with probability $\sum_z p'(x,y,z)$.

Recall that by definition the Shannon entropy of the distribution $(p_1, \ldots, p_k)$ equals $\sum p_i(-\log p_i)$; it does not exceed $\sum p_i(-\log q_i)$ for any other distribution $q_1 + \ldots + q_k = 1$. Therefore the entropy $H(\xi_{xy})$ can be bounded (from above) by using $f^2(x,y)$ as the "other" distribution:

$$H(\xi_{xy}) \leqslant \sum_{x,y}\left(\sum_z p'(x,y,z)\right)(-2\log f(x,y)).$$

Then we write similar bounds for two other projections and apply the inequality

$$H(\xi) = H(\xi_x, \xi_y, \xi_z) \leqslant \frac{1}{2}(H(\xi_{xy} + H(\xi_{yz}) + H(\xi_{xz})),$$

(Problem 172, p. 181). We conclude that

$$H(\xi) \leqslant \sum_{x,y,z} p'(x,y,z)(-\log f(x,y) - \log g(y,z) - \log h(x,z)) =$$

$$= \sum_{x,y,z} p'(x,y,z)(-\log p(x,y,z)).$$

By definition $H(\xi) = \sum_{x,y,z} p'(x,y,z)(-\log p'(x,y,z))$, and we get a contradiction, since p' is c times smaller than p (and therefore $-\log p'$ exceeds $-\log p$ by $\log c$).

8.7 Lipschitz transformations are not transitive

In this section we apply Kolmogorov complexity to analysis of the properties of infinite sequences. Let us start with the following definition related to the Cantor(metric) space Ω of infinite binary sequences.

A mapping $f\colon \Omega \to \Omega$ is a *Lipschitz* one if

$$d(f(\omega_1), f(\omega_2)) \leqslant cd(\omega_1, \omega_2)$$

for some constant c and for all $\omega_1, \omega_2 \in \Omega$. Here d is the standard distance in the Cantor space defined as 2^{-k} where k is the first place where two sequences differ.

Informally speaking, Lipschitz property means that the first n digits of the sequence $f(\omega)$ are determined by $n + O(1)$ first digits of ω. In particular, all mappings defined by local rules (each bit in $f(\omega)$ is determined by some its neighborhood in ω) have Lipschitz property.

We are interested in the following property of a mapping f: for every two sequences ω_1, ω_2 and for every $\varepsilon > 0$ there exists a number N and sequences ω_1' and ω_2' such that

$$\omega_2' = f(f(f(\ldots f(\omega_1')\ldots))) \quad (N \text{ iterations})$$

and

$$d(\omega_1, \omega_1') < \varepsilon, \quad d(\omega_2, \omega_2') < \varepsilon.$$

(In other terms, for any two open neighborhoods there exists an orbit that starts in the first one and get inside the second one.) We call this property "transitivity" of f (in this section).

It is easy to check that left shift (that deletes the first bit of the sequence) is transitive: if we need a sequence that starts with x_1 and is transformed (after several shifts) into a sequence that starts with x_2, just take a sequence that starts with x_1x_2.

Now the question: does the left shift remains transitive if we change the definition and replace Cantor distance d by the so-called *Besicovitch* distance:

$$\rho(\omega_1, \omega_2) = \limsup_{n\to\infty} d_n(\omega_1, \omega_2)/n,$$

where d_n is a number of discrepancies among the first n terms, i.e., the number of $i < n$ such that ith terms of ω_1 and ω_2 differ.

It turns out that in this case the left shift is no more transitive (is not "Besicovitch-transitive"). Moreover, the following statement is true:

{durand-cer

Theorem 137 *No Lipschitz mapping can be Besicovitch-transitive.*

(Speaking about the Lipschitz property, we have in mind the original definition using Cantor distance.)

The reason is quite simple: the Lipschitz mapping does not increase significantly the complexity of the prefixes of a sequence, since n bits of the output sequence are determined by $n + O(1)$ bits of the input sequence (we assume that transformation rule is computable; if not, we have to

relativize complexity by a suitable oracle). On the other hand, if two sequences are Besicovitch-close, then their prefixes have almost the same complexities (a change in a small fraction among the first n bits can be encoded by a short string compared to n).

$\lhd$ For a formal proof it is convenient to use the notion of effective Hausdorff dimension of a sequence (which is equal to $\liminf KS(\omega_0 \ldots \omega_{n-1})/n$ for a singleton $\{\omega\}$), see Theorem 97 in Section 5.8, p. 139).

Lemma 1. A computable Lipschitz mapping does not increase the effective Hausdorff dimension of a sequence.

(Speaking about computability of a Lipschitz mapping $f\colon \Omega \to \Omega$, we mean that n first bits of $f(\omega)$ are effectively determined by $n+c$ first bits of ω for some c.)

Indeed, if $f(\omega_1) = \omega_2$, then the complexity of n-bit prefix of ω_2 does not exceed (up to $O(1)$) the complexity if $n+c$ bit prefix of ω_1, and for the dimension these constants are not important.

Lemma 2. If Besicovitch distance $\rho(\omega_1, \omega_2)$ is less than ε, then effective Hausdorff dimensions of ω_1 and ω_2 differ at most by $H(\varepsilon)$.

(Here $H(\varepsilon)$ is the Shannon entropy of a random variable with two values that have probabilities ε and $1-\varepsilon$.)

Indeed, if the first n terms of ω_1 and ω_2 differ in k places, then the complexities differ at most by the complexity of the bitwise XOR of these two sequence (since knowing one sequence and the XOR we easily get the other one). And any sequence of length n that has k ones has complexity at most $nH(k/n) + O(\log n)$ (see Section 7.3.1, Theorem 122, p. 182). Lemma 2 is proven.

So if we take a sequence of a zero dimension (say, a computable sequence), then any sequence that is Besicovitch-close to it has small dimension, and computable Lipschitz mapping does not increase this dimension, so we can get only sequences of small effective Hausdorff dimension. On the other hand, any sequence that is Besicovitch-close to a random sequence (that has dimension 1) has dimension close to 1 (Lemma 2 again).

So we have proven our theorem for *computable* Lipschitz mappings. It remains to note that all our arguments are relativizable and that every Lipschitz mapping is computable relative to some oracle. $\rhd$

[What is the right name for transitivity? What are the correct references? Laurent knows for sure.]

8.8 Ergodic theorem

Vyugin's proof? needs to be reconstructed

9

( ) 0′-?

10 , ,

,

, ,
(,) , (-? ?)

[ ]