

Chapter 1

Kolmogorov Complexity

By *Bruno Durand* and *Alexander Zvonkin*

The term “complexity” has different meanings in different contexts. *Computational complexity* measures how much time or space is needed to perform some computational task. On the other hand, the *complexity of description* (called also *Kolmogorov complexity*) is the minimal number of information bits needed to define (describe) a given object. It may well happen that a short description requires a lot of time and space to follow it and actually construct the described object. However, when speaking about Kolmogorov complexity, we usually ignore this problem and count only the description bits.

As it was common to him, Kolmogorov published, in 1965, a short note [10] that started a new line of research. Aside from the formal definition of complexity, he has also suggested to use this notion in the foundations of probability theory. His idea was quite simple:

An object is random if it has maximal possible complexity.

The definition of complexity uses the notion of an algorithm; this unexpected marriage of two *a priori* distant domains—in our case, probability theory and theory of algorithms—is also a typical trait of Kolmogorov’s work.

1.1 Algorithms

The notion of an algorithm is quite recent. In 1912 (when neither computers nor programming languages existed) Émile Borel (see [19]) used the phrase “a formal and precise automatic rule” describing an object which we would now call an

algorithm.⁽¹⁾ However, a mathematical theory of algorithms was developed only in the 1930ies (by Turing, Gödel, Post, Church, Kleene and others). The key observation was the existence of a universal algorithm (see below); it allows to prove easily that some problems (e.g., the so-called “halting problem” that asks whether a given algorithm terminates on a given input) are undecidable (cannot be solved by algorithms). Note that to prove the non-existence of an algorithm that solves a certain problem we need a mathematically precise definition of this notion. When appeared, this notion became a subject of the theory of algorithms, also called *theory of recursive functions* or *theory of computability*.

The remaining part of this section discusses some aspects of the notion of algorithm; the reader not interested in these details may skip it and proceed directly to Section 1.2.

It is rather difficult to give a mathematical definition that captures the intuitive idea of an algorithm in its full generality; instead, we may define a specific class of algorithms and claim that this class is representative, i.e., that any algorithm is equivalent to a certain algorithm in this class. (By the way, one of these classes was suggested by Kolmogorov.)

1.1.1 Models of computation

A model of computation formally describes some specific class of algorithms (the class of objects used as input/output data, how they are processed, etc.) Some computational models resemble programming languages while others look more as a hardware description. In any case, we assume that computational resources are unlimited (and forget that in real programming languages integers are usually bounded, processor architecture has a fixed word length, etc.).

(The study of resources (time and space) needed to solve a given problem is a different field called *computational complexity*. Let us note that an important notion in this field, *NP-completeness*, was introduced at the beginning of the 1970ies independently by three researchers, one of whom, Leonid Levin, is Kolmogorov’s student. The first publications by Levin were about Kolmogorov complexity [21]. His short biography and a brief story how Kolmogorov influenced him may be found in the book [17].)

¹The history of the term “algorithm” is interesting in itself. This word is a derivative of the name of a medieval Persian savant Al-Khwārizmī (787 – c. 850) who was the author of a book through which the Europeans learned the positional number system and the rules of arithmetic operations (addition, multiplication, etc.). The name of Al-Khwārizmī (which means “de Khorezm”, a town in Uzbekistan today called Khiva) was transliterated in Latin as *Algorithmus*. The term “algorithms” meant at the beginning “the rules of four arithmetic operations”. Then by extension it has got the meaning of any systematic method of computation. Leibnitz called “algorithms” the set of rules of computing differentials and integrals. It is only gradually that the word acquired its modern meaning; one hundred years ago this process was not yet finished. (Authors’ note)

Which computational model is “the best one”? This depends on our purposes. If we want to write real programs, it is natural to use a real computer and an appropriate programming language. On the other hand, if we want to prove theorems it would be more convenient to work with an abstract model of computation; a very simple model, with a small number of primitives, would then be better. However, there is no canonical model adapted for proofs since different models are more suitable for different results.

The most popular model is Turing machine. It is rather easy to prove the universality of this model; however, we have to deal with many details concerning tapes, symbols, representation of the transition table, etc. There are many versions of Turing machines; the most common one was, by the way, presented by Post and not by Turing.

Recursive functions “à la Church” give a more mathematical and attractive model though the proofs of certain basic theorems become somewhat discouraging if not frightening.

Markov algorithms are similar to rewriting systems for strings with termination conditions; this is a model difficult to manipulate (but well suited for the proof of the undecidability of word problems).

The RAM (random access machines) model resembles von Neumann-style computers. . .

Teaching the algorithms theory, one may choose a different approach and not fix any specific model but rely directly on the intuition of algorithms. More formally, it means that we have to accept some properties of algorithms used in the proofs as axioms. Then we do not need to go into cumbersome details of a specific computational model; the price is, however, that the list of axioms is open (e.g., if during the proof we need to establish the computability of some function, we just describe informally its computation and then add a new axiom saying that this function is computable).

1.1.2 All models of computation are equivalent

Why do we believe that this or that computational model correctly reflects the intuitive notion of an algorithm? This statement is usually called “the Church thesis” (for a given computation model): it claims that any computable function (computed by an algorithm in the informal sense) is computable in this model. This assertion is not a mathematical one; it is a belief concerning the notion of intuitive computability. On the other hand, we can prove that these assertions for different computation models are equivalent, since it turns out that the class of computable functions is the same for different existing models (Turing machines, recursive functions, etc.).

The name given to the thesis is rather inappropriate. Church claimed that all intuitively computable *total* functions are computable in his model. A long controversy followed, in which Gödel took sometimes surprising positions [1]. The first

equivalence theorem for two different models (recursive functions “à la Church” and Turing machines) was established by Turing in his seminal article, and the thesis in its most general form was formulated by Post. Therefore, a more appropriate name would be “Church–Turing–Post thesis”.

All this was done in the 1930ies, so why Kolmogorov might want to suggest a different computation model in the 1950ies? His motivation could be reconstructed as follows. Though all computation models mentioned above are equivalent, the translation between them sometimes replaces one step in one model by a long sequence of steps in another one. For example, an addition may be an elementary operation in some programming language while its implementation by Turing machine requires many steps.

Kolmogorov wanted to find a model whose steps are “elementary” in the sense that they do not allow natural decomposition into a sequence of simpler steps. On the other hand, he tried to find a most general (and natural) model among these models. This means that elementary steps of any other model (if they are indeed elementary according to our intuition) should not require further decomposition when translated into Kolmogorov’s model.

1.1.3 Kolmogorov–Uspensky machines

The model suggested by Kolmogorov was later called *Kolmogorov–Uspensky machines*. These machines are not related to Kolmogorov complexity, but they are related to Kolmogorov himself; hence we say a couple of words about them.

The configuration (state of the computation) of a Kolmogorov–Uspensky machine is a graph; some node of this graph is declared to be active. The program for the machine is a list of rules that say how this active part should be transformed and when the processing halts. So the computation step is indeed “local”; it deals with a finite size neighborhood of the active node. On the other hand, the “topological structure” of the computation can become rather complicated. This may be considered as a disadvantage of the model since it allows some actions that are hard to perform in a physical space. (For example, a Kolmogorov–Uspensky machine can create a labeled tree that provides an unreasonably fast access to an exponential amount of information.) So one may want to restrict somehow the class of allowed graphs [19, 8, 1]. Later a version of this model was considered by Schönhage (who used directed graphs with unlimited in-degrees). It seems pertinent to mention here the development of the GASM (Gurevich Abstract State Machines) which were inspired by Kolmogorov–Uspensky machines but have other goals and do not play a specific role in the classical computability theory. The first complete description of Kolmogorov–Uspensky machines may be found in [11]; a more modern presentation is given in [19].

1.1.4 Universality

Now we are accustomed to the idea that the same processor can be used to perform different tasks if provided with a suitable program. However, this idea of “universal computation” was a nontrivial and very important step in the development of the first real computers.

The same idea can be formally expressed as follows: there exists a *universal* computable function U of two arguments p and x . The universality means that we can obtain any computable function of x by fixing an appropriate first argument p (a *program* for this function).

Why does a universal function exist? Imagine an interpreter of an arbitrary programming language that considers its first argument p as a program and executes this program using x as its input.

1.1.5 Non-computable functions

The existence of a universal computable function immediately brings us to a paradox. Consider the function $F(p) = U(p, p) + 1$. This (unary) function is computable since U is. It should then have a program associated to it (since U is universal); let us denote this program by q . What happens if we apply program q to itself? By definition of U this gives $U(q, q)$. On the other hand, since q is a program for F , the same result must be equal to $F(q) = U(q, q) + 1$. So we get $U(q, q) = F(q) = U(q, q) + 1$, and this seems impossible.

The only way to explain this paradox is to recall that certain computations may never terminate, so a program may compute a non-total function. And the contradiction disappears if $U(q, q)$ is not defined.

A similar argument shows that the halting problem is undecidable: there is no algorithm that gets a program p and input x and tells whether $U(p, x)$ is defined (= whether the program p terminates on input x).

1.1.6 Back to algorithms

Returning to practice, let us note that the notion of a computable function captures only one aspect of algorithmic practice. For example, the behavior of a real-time algorithm (such as an operating system) is a more complicated thing than a mere function. The choice of a correct mathematical model for this class of algorithms (very important for practice) is a well studied but not fully solved problem of theoretical computer science.

1.2 Descriptions and sizes

Any information may be encoded as a bit string (a finite sequence of bits). For this reason, in what follows we assume that our algorithms deal with bit strings.

Binary strings are also called *words in the alphabet* $\mathbb{B} = \{0, 1\}$, and the set of all binary strings is denoted as \mathbb{B}^* . We identify \mathbb{B}^* with the set $\mathbb{Z}^+ \setminus \{0\} = \{1, 2, 3, \dots\}$ using the lexicographic order. (The empty word is associated with 1, then $0 \mapsto 2$, $1 \mapsto 3$, $00 \mapsto 4$, $01 \mapsto 5$, etc.: a string u is associated with a natural number that has binary representation $1u$. For example, the word 00 corresponds to the number 100_2 , i.e., 4.)

The length $|u|$ of a binary word u , i.e., the number of letters in it, is then equal to the integral part $\lfloor \log u \rfloor$ of the binary logarithm of the number associated with u . (Note that $|u|$ stands for the length of the word u and not for the absolute value of the corresponding integer.)

Definition 1.2.1. *Let $f : \mathbb{B}^* \rightarrow \mathbb{B}^*$ be a computable function. We define the complexity of $x \in \mathbb{B}^*$ with respect to f as*

$$K_f(x) = \begin{cases} \min |t| & \text{such that } f(t) = x, \\ \infty & \text{if such } t \text{ does not exist.} \end{cases}$$

In other terms, we call *descriptions* of x (with respect to f) all strings t such that $f(t) = x$; then the complexity $K_f(x)$ is defined as the length of the shortest description.

The main problem with this definition is that the complexity depends on the choice of f . It is unavoidable, but the theorem stated below (due to Kolmogorov but already present, in an informal way, in the paper of Solomonoff [18]) explains in which way this dependence can be limited. This theorem was later independently proved by Chaitin but does not appear in his first papers on the subject [2, 3]—the priority claims have provoked a long and futile controversy explained in [13].

Theorem 1.2.1 (Existence of an optimal function). *There exists a computable function f_0 (called optimal function) such that for any other computable function f there exists a constant C such that*

$$\forall x \quad K_{f_0}(x) \leq K_f(x) + C. \quad (1.2.1)$$

(Note that the constant C may depend on f but not on x .)

Proof. Let t be a shortest description of x with respect to f , i.e., $f(t) = x$. Then f_0 uses as a description of x the pair (p, t) where p is a program that computes the function f . In this pair p has $|p|$ bits and t has $|t|$ bits, so the total number of bits is $|p| + |t|$, i.e., $|p| + K_f(x)$. So we let $C = |p|$. \square

Remark 1.2.1. This argument needs some refinement. We cannot use the pair (p, t) directly; we need to encode it by a single string. Not any encoding will work. An appropriate encoding may encode p in a very inefficient way—this only increases the constant C . On the other hand, it is essential to be able to encode t without any loss of space since an encoding of t which demands, say, $\alpha|t|$ bits with $\alpha > 1$ leads to the complexity $\alpha K_f(x) + C$ instead of $K_f(x) + C$.

Corollary 1.2.1. *If f_1 and f_2 are two optimal functions then there exists a constant C such that*

$$\forall x \quad |K_{f_1}(x) - K_{f_2}(x)| \leq C. \quad (1.2.2)$$

Proceeding from this corollary, we choose some optimal function f_0 and fix it. The subscript f_0 in K_{f_0} is then suppressed. However, after doing this we still have in mind that in fact the Kolmogorov complexity is defined only up to a bounded additive term.

Definition 1.2.2. *The Kolmogorov complexity $K(x)$ is the complexity $K_{f_0}(x)$ with respect to some optimal function f_0 . The complexity $K(x)$ is defined up to a bounded additive term.*

Proposition 1.2.1.

$$K(x) \leq |x| + C, \quad \text{or, equivalently,} \quad K(x) \leq \log x + C. \quad (1.2.3)$$

Proof. It suffices to let $f(x) = x$ in (1.2.1), i. e., to use x itself as a description of x . \square

Proposition 1.2.2 (Distribution of complexities). *Consider all binary strings of length n . The fraction of strings x of length n such that $K(x) < n - k$ does not exceed 2^{-k} .*

Proof. The number of strings of length n is 2^n while the number of (potential) descriptions of length less than $n - k$ is

$$1 + 2 + \dots + 2^{n-k-1} < 2^{n-k}.$$

\square

There exist strings of length n whose complexity is at least n (they are often called *incompressible* strings). Indeed, there are 2^n strings of length n and at most $1 + 2 + \dots + 2^{n-1} = 2^n - 1$ potential descriptions of length less than n .

One may ask for an example of an incompressible string. However, it is not possible to find an incompressible string of length n effectively (having n as input). Indeed, if it were possible, a string generated by this algorithm would have complexity $\log n + c$ since we need to specify n (about $\log n$ bits) and the algorithm itself (constant number of bits), and $\log n + c$ is less than n for all sufficiently large n .

Incompressible strings are a useful tool in theoretical computer science (automata theory, formal languages, etc.).

Today everybody uses software for data compression and decompression; this was not the case in the 1960ies when Kolmogorov complexity was introduced.

However, the Kolmogorov complexity theory may still provide useful hints: for example, if a software advertisement claims that a latest version of the super-compressor compresses every file by a certain factor, you better avoid this product.

Finally, to prepare for the next section (on Gödel's incompleteness theorem), we present a variation on a well known theme of *busy beavers*. Initially the busy beaver numbers were defined as follows. Consider Turing machines that have at most n states and whose tape alphabet consists of two symbols (say, "blank" and "stroke"). We start such a machine on the blank tape. Some machines do not terminate at all. For the machines that terminate we count the number of steps; let $T(n)$ be the maximal number of steps among the terminating machines with at most n states.

Evidently, $T(n)$ is an increasing function of n since we consider all machines that have *at most* n states. It grows very fast; in fact, it grows faster than any computable function (does not have a computable upper bound). Indeed, if a computable upper bound $f(n)$ exists, it may be used to solve the halting problem, since we know that if a machine with n states does not terminate after $f(n)$ steps, it will never terminate. So no computable function, even a fast growing one, like $n!^{n!}$ ($n!$ levels), is an upper bound for $T(n)$.

But here we consider a different (but related) fast-growing function. Let us define $\delta(n)$ as the biggest integer that has complexity less than n . It exists since the number of descriptions of size less than n is finite. By definition we have $n \leq K(x)$ for any $x > \delta(n)$, e.g., for $x = \delta(n) + 1$. If the function δ were computable we would have $K(\delta(n) + 1) \leq \log n + C$ since n might serve as a description of $\delta(n) + 1$. The contradiction is evident. Hence, δ is not computable. In a similar way we can prove that δ grows faster than any computable function. (It suffices to replace $\delta(n)$ in the preceding inequalities by any computable upper bound for δ .)

1.3 Gödel's theorem

1.3.1 It is proved that one cannot prove everything

The function $K(x)$ is not computable. How can we use it? For example, to prove theorems. Maybe the most remarkable example is the proof of Gödel's incompleteness theorem. Roughly speaking, this theorem claims that not all the truths are provable. Mathematics has its intrinsic limits: there exist propositions that are true but impossible to prove.

We propose to you a more "concrete" form of a proposition that is true but unprovable; it was suggested by Gregory Chaitin [4].

Theorem 1.3.1 (Gödel's incompleteness theorem). *There exists a num-*

ber m such that for every x the proposition

$$\boxed{K(x) \geq m}$$

is unprovable.

Note that the set of all x such that $K(x) < m$ is finite. So the proposition $K(x) \geq m$ is *true* for infinitely many values of x . And all these truths have no proof.

Proof of Gödel's theorem. We use the same argument as in the previous section (when we proved that the busy beaver function $\delta(n)$ is non-computable) with some modifications.

Suppose that the statement is false, i.e., that for any integer m there exists x such that the proposition " $K(x) \geq m$ " is provable. Then consider an algorithm that finds this x given m :

- input: an integer m ;
- enumerate all the theorems (a theorem is a proposition which has a proof);
- as soon as a theorem " $K(x) \geq m$ " is found, return x .

Using this algorithm, we may consider its input m as a description of its output x . Therefore, according to (1.2.3), $K(x) \leq \log m + C$. But, on the other hand, $K(x) \geq m$ is a theorem (and therefore is true; we assume that all theorems are true, otherwise our notion of proof would be bad). So

$$m \leq \log m + C.$$

The constant C is "absolute": it depends neither on m nor on x . So we get a contradiction, since this inequality is false for sufficiently large m . \square

For a neophyte it is difficult to appreciate fully this simple argument. One should know, however, that Gödel's theorem had literally shattered the mathematical community at the beginning of the 1930ies. The projects and hopes of great mathematicians, such as David Hilbert, to get a complete formal theory as a framework for mathematics were reduced to nothing. Gödel's theorem became (and remains) one of the basic results and one of the gems of mathematical logic. (Numerous volumes are devoted to this theorem, including philosophical essays and popular expositions; the bestseller by Douglas Hofstadter [9] has 800 pages.) Generations of logicians tried to understand fully *why* and *how* mathematics is incomplete. Due to all their work, we are now able to explain the proof of this theorem in a single paragraph.

A philosopher once remarked that "every profound idea passes through three stages during its development: (1) it is impossible; (2) it is maybe possible but incomprehensible; (3) it is trivial". It seems that Gödel's theorem has already arrived to the third stage.

1.3.2 Formal systems

Of course, our account of the complexity proof of Gödel theorem is quite informal. An informed reader may be worried about this. He had probably heard the words *formal systems*. Indeed, we speak about proofs and theorems but do not say what the axioms or inference rules are (or any other proof machinery). It turns out, however, that in fact we do not need to go into these details. There is only one property of the proof system that is necessary.

Definition 1.3.1 (Formal system). *A proof system is an algorithm that generates statements, and all these statements are true.*

All usual proof systems (based on axioms and inference rules) are formal systems according to the above definition. Indeed, theorems can be enumerated in the following way: write all the strings of characters in a certain order; for each of them check whether it is a derivation (starts with axioms, follows inference rules, etc.); if yes, find the statement that has been derived (usually the last statement of the derivation) and output this statement.

This assumes that there is an algorithm that can distinguish derivations from other character strings, but this is true for all reasonable formal systems. Otherwise, how could we check that a proof is correct? (By a vote of members of a jury? By asking an oracle, a prophet or another sort of authority? By a tournament of knights like in Middle Ages? By drawing lots?) This is indeed the basic underlying idea of a formal system; the correctness of proofs should be verified “mechanically”, that is, by an algorithm.

1.3.3 Berry paradox

Gregory Chaitin, who suggested this remarkable proof, mentioned that this proof is a formalization of the “Berry paradox” published by Bertrand Russell in 1908. It considers

the smallest integer N whose description needs more than thousand words.

First of all, the integers that need more than one thousand words in order to be described, do exist—just because the number of shorter descriptions is finite. Therefore, the boxed sentence characterizes the integer N without ambiguity; in other words, it is a description of N . But it contains less than thousand words! Quite often a paradox appears since we refer to a notion not well defined. What is this notion here? The notion of the smallest element (in a set of positive integers) used in the phrase is well defined: the axiom of induction implies that every non-empty subset of \mathbb{N} has the smallest element. On the other hand, the notion of “description” is indeed not well formalized. Kolmogorov complexity provides a formal framework for this notion. Then, replacing words by bits, consider (for every m)

the smallest integer N such that $K(N) > m$.

Such an integer exists for every m ; however, this expression (for a given m) is not a description of N in Kolmogorov's sense, since there is no algorithm that finds this N . But if we change the sentence and say

the first integer N such that $K(N) > m$ is provable

(where "first" means "first in the sequence of generated proofs"), then it is indeed a description of complexity $\log m + c$, and the only way to avoid the contradiction is to conclude that for some m there are no proofs of statements of the form $K(N) > m$. As you see, we come to the proof of Gödel's theorem explained above.

1.3.4 Gödelian propositions: "concrete" examples

Good students often ask: is it possible to give a concrete example of an unprovable proposition?

This question sets a trap for us. Without doubt, we mean a true but unprovable proposition. But how, then, could we know that some proposition is true if it has no proof? Apparently, we should have other reasons, and very strong reasons, by the way, in order to *believe* that it is true.

Logicians know different ways to address this problem. For example, we can provide two statement "A" and "not A" that are unprovable. Then we know that at least one of them is true but unprovable. (But this probably cannot be considered as a "concrete" example.)

The other possibility is to consider different theories: a weak one (e.g., first-order arithmetic) and a stronger one (second-order arithmetic or set theory). Then we show a statement that is not provable in the weak theory, and this fact as well as the statement itself can be proved in the strong theory.

This is the approach found by Gödel himself. He proved that by using only (first-order) arithmetic it is impossible to prove that this theory (first-order arithmetic) is *consistent*, i. e., that it does not contain a contradiction. But the consistency of the (first-order) arithmetic can be proved in the set theory or second-order arithmetic (and, last but not least, it is confirmed by mathematical practice).

Kolmogorov complexity provides us with another procedure of producing *Gödelian* (that is, true but unprovable) propositions. Let us suppose that the number m in the Gödel theorem is, say, 100. (A careful reasoning can indeed provide some specific value for m . It depends on the formal system we use and the optimal function we choose in the definition of Kolmogorov complexity.)

Then we may toss up a coin, say, 500 times, and then claim that the complexity of the sequence of bits obtained is greater than 100. This statement will be impossible to prove, but we may be practically sure that it is true: the probability

of getting a false statement in this way is less than 2^{-400} (see the proposition about the distribution of complexities on p. 7). We thus obtain an arithmetic statement which we believe to be true for probabilistic reasons.

1.4 Definition of randomness

1.4.1 Questions, questions, questions...

The more we think about the notions of probability and randomness, the more difficult is to explain even the most “basic” things. Let us start by an example borrowed from the everyday life. Suppose that you see a car whose number on the licence plate is 7777 ZZ 77. This number seems rather extraordinary, doesn’t it? As to the number 7353 NY 42, it seems perfectly “normal”. Why?

We would like to say: because the first number has very small probability. Yet, this answer is not valid: the probability of the first number is exactly the same as that of the second. If we suppose that all digits and all letters are equiprobable and independent, then this probability is equal to $1/(10^6 \times 26^2)$. When we toss up a coin 1000 times, the probability to get 1000 heads is 2^{-1000} , but the probability of every other sequence of heads and tails is exactly the same! Why then does the sequence of 1000 identical tosses arouse a suspicion as to its random character? If we think more about this phenomenon, we finally understand that, in fact, while speaking about car “numbers”, we do not mean individual numbers but *sets* of “similar” numbers. The first number is a representative of the set of numbers where “the digits are repeated, and also the letters”. This set is simple to describe, and its probability is small. As to the second number, it is “just a number”. We are unable to outline its specific simple property which would describe a set of small probability. (And, if you *are*, this was not intended by the authors.) This is related to complexity: a simple property that is true only for few objects makes these objects simple.

Now, let us go further: what is *probability*?

Despite what one might believe, probability theory (whose rigorous mathematical foundation was provided by Kolmogorov himself in 1933) does not answer this question. This theory formulates, in a form of axioms, the properties of probabilities. It also permits to calculate probabilities of certain events when probabilities of other events are known. Thus, it treats probability theory just as any other branch of mathematics, without bothering much about “useless” philosophical questions. People were quite satisfied by this situation—except for Kolmogorov. How would you explain to an intelligent person with no mathematical background what probability is? You claim that when one tosses a coin, the probability to get a head is $1/2$. Then he starts to question you:

- I don’t understand the word “probability” in your sentence.
- I mean that the chances to get a head or a tail are equal.

— Hm... you've replaced the word "probability" by "chance", but what does it mean?

— OK, OK. I would only like to say that in, say, a thousand of coin tosses you get approximately half of heads and half of tails.

— Ah... It seems that I begin to understand something. For the moment, I won't ask you how precise this approximation is. But please tell me: do you really guarantee that the fraction of heads is *always* close to one half?

— Alas, no. It is not always the case, but it is true with a very high probability. However, there remain extremely small chances to get (for example) only the heads.

— "With a very high probability"! Again this word! You started to explain what probability is, but now you use the same notion in a much more complicated context, that of 1000 tosses instead of one. Frankly, all that is not very serious.

— But wait, wait! I can give you axioms which describe the properties of probabilities...

— To know the properties of something is certainly very important. But it would also be good, before speaking about properties, to understand what is the object whose properties we want to study. *The sepulkas are used for sepulation, one puts them in a sepulkary, they can be assembled in beads, and they are able to whistle:*⁽²⁾ do you understand anything here?

— We've been talking for a long time already, but there still remains an approach by which I could try to convince you. You see, the property of having the proportion of 0's and of 1's close to 1/2 is true not only with a large probability; it is also true for *all* random sequences. The sequences which do not satisfy this property are just not random.

— Is the sequence of alternating zeros and ones random according to you?

— No, it is not. It is obviously too regular to be random.

— Then I don't understand at all what you are speaking about. What does the word *random* mean?

— Mmm...

— Are there at least any axioms which would describe the properties of the objects you call random?

— Mmm...

1.4.2 Random sequences

The approach based on Kolmogorov complexity permits to define the notion of an *individual* random sequence formally without any references to probabilities. For infinite sequences of bits it provides a sharp boundary between random and non-random sequences. For finite sequences (binary strings) we have no hope to

²See Stanisław Lem, *Memoirs of a space traveller*, London, 1982. (Authors' note)

achieve this sharp division. (Indeed, changing one bit cannot make a random sequence non-random, but a sequence of changes can.)

For a finite sequence, to be random is a synonym of having a complexity close to the length. In other words, the best (or close to the best) way to describe such a sequence is to present it literally. Then we can prove that in a random sequence the frequency of zeros (and ones) is close to $1/2$. For example, consider a sequence of 1000 bits that contains, say, 300 ones and 700 zeros. This fact significantly reduces its complexity, and therefore the sequence is not random. (Indeed, we can say that this is a sequence that has number N in the list of all sequences that have 300 ones and 700 zeros, and one can see that the bit length of N is much smaller than 1000: $N \leq \binom{1000}{300}$ hence $\log N \leq \log \binom{1000}{300} < 877$.)

So the random sequence should contain approximately equal number of zeros and ones. However, if we push the same reasoning a little further, we see that if a sequence had exactly the same number of zeros and ones, we would also have some nontrivial information about it, so it could not be perfectly random. For a truly random sequence, zeros and ones must be slightly unbalanced (the difference should be proportional to the square root of the length).

As we have said, according to Kolmogorov's idea a sequence is random if it is "almost" incompressible. However, complexity is defined for finite sequences. Therefore to define randomness for an infinite sequence we need to consider some finite strings related to it. The most natural choice is prefixes.

If an infinite sequence is denoted by x , let $x_{1:n}$ denote a finite string consisting of the first n bits of x . We could try the following definition: x is random if and only if

$$\exists C \forall n \ K(x_{1:n}) > n - C.$$

The constant term C is natural since the complexity K is defined up to an additive constant. Unfortunately, this definition does not work: there is no sequence x that satisfies this requirement. Ten years passed before this difficulty was resolved. The solution is sometimes considered as a "technical trick". However, what is considered as a technical trick by mathematicians corresponds to a reality well known to computer scientists: we should distinguish between a program that reads/writes a bit string (of a specified length) and a program that reads from the (potentially infinite) input stream or writes into the output stream. Storing a file or a string, we should reserve additional place to store its length or reserve some symbol as a terminator. Both solutions require additional space, at least $\log n$ bits for keeping the length of an n -bit string.

There are different technical solutions; one of them is that we require our descriptions to have the prefix property: if a string t is a description of some x , then any continuation of t (i. e., any string that extends t) is also a description of x . So we do not need to say when the description stops, since the trailing bits do not change anything. If we modify the definition of the Kolmogorov com-

plexity in this direction (which requires some precautions but is feasible), the formula suggested in the previous paragraph becomes a reasonable definition of randomness for infinite sequences. (Technically speaking, we may switch from the “plain” complexity to “prefix” complexity. This gives some other advantages; for example, for this version of complexity the complexity of a pair of binary strings (under any computable encoding) does not exceed the sum of their complexities. (This is not true for the original “plain” Kolmogorov complexity where an additive logarithmic term is needed.)

Another Kolmogorov’s idea was to define a random sequence as *a sequence which escapes from every effectively null set*. In order to define the notion of an “effectively null set” we take a usual definition of a null set (a set of measure zero) and interpret the existential quantifier in an effective way (instead of mere existence we demand that the required object be provided by some algorithm). This gives us the following definition:

A set A is an *effectively null set* if there exists a program p which, for any integer n given as input, produces an infinite series of strings

$$x_0^{(n)}, x_1^{(n)}, \dots$$

such that for all n

$$\sum_i 2^{-|x_i^{(n)}|} < 1/n$$

and for every $w \in A$ and for every n the sequence w has one of the strings $x_i^{(n)}$ as a prefix.

This idea was developed by Martin-Löf [15], a student of Kolmogorov. The effectively null sets correspond to “non-randomness” tests, and a sequence is random if it resists to all these tests. The existence of a universal algorithm allows us to construct one *universal test*: every sequence which resists to this test resists as well to all other tests and is therefore random.

One of the principal results of the algorithmic information theory is the connection between the incompressibility of prefixes of infinite sequence and its randomness seen as resistance to every algorithmic test. This equivalence is a theorem proved in the 1970ies by Levin and Schnorr [12, 16] in the contexts of slightly different definitions (they used some version of the so called “monotone” complexity; see [20] for the details). Thus a good definition of randomness (for an infinite sequence) was obtained; “good” means here that two different reasonable definitions turn out to be equivalent. Moreover, all basic theorems of probability theory that have the form “for almost all x the property P is true” can be now reformulated as follows: “for every random (in the sense described above) sequence the property P is true”. The latter result is not a formal statement; we mean that different authors studied different theorems of this form (for example, the ergodic theorem) and proved that these theorems remain true for every algorithmic random sequence. In certain cases (for example, for ergodic theorem), this is a rather delicate work and about ten years were required to complete it.

The relation between complexity and measure can be used also for finite sequences. For example, we may prove that any incompressible sequence has some property (by showing that sequences which do not have this property can be compressed). Then we conclude that almost all sequences have this property (being incompressible).

1.4.3 Sequences of low complexity

We have seen that the sequences that have prefixes of high complexity are random. It is natural to ask which sequences have prefixes of small complexity. There exists a nice theorem, proved independently by several authors long ago when the theory of Kolmogorov complexity appeared. According to the date of the first publication, this theorem must be attributed to Albert Meyer and it was published in a paper by Loveland [14]. Its proof may be found in [21]. Let us state this result using the notation of the previous section.

Theorem 1.4.1. *A sequence x is recursive (i. e., computable by an algorithm) if and only if $\exists C \forall n \ K(x_{1:n} | n) < C$.*

We use here a slightly more general—namely, *conditional*—form of Kolmogorov complexity. In order to simplify our presentation we did not mention it until now, but it is a useful and natural notion. We define $K(x | y)$ (complexity of x while knowing y) as the length of the shortest description of x , if descriptions have access to y as input. Formally,

$$K_f(x | y) = \begin{cases} \min |t| & \text{such that } f(t, y) = x, \\ \infty & \text{if such } t \text{ does not exist.} \end{cases}$$

The existence of optimal functions is proved in the same way as before. If y is fixed, the complexity $K(x | y)$ as a function of x coincides with $K(x)$ up to a constant so we get nothing really new (recall that the complexity is defined up to an additive constant anyway). But this new notion makes sense, for example, if we let y be the length of x (or the number of zeros in x , the substring formed by bits with even indices, etc.)

The theorem says that the “simplest” infinite sequences are exactly the computable ones. It is important to use $K(x_{1:n}|n)$ and not $K(x_{1:n})$ since even for a computable x the prefix $x_{1:n}$ contains a small amount of information, i.e., the length n . (Why didn't we add a similar term in the characterization of random sequences? In fact this is also possible but not necessary.)

In one direction this theorem is trivial: if a sequence is recursive then complexity is bounded (in fact, bounded by the complexity of a program that produces $x_{1:n}$ given n).

The converse implication is more subtle. It is one of the examples that appear from time to time in theoretical computer science, when it is possible to prove

that an algorithm exists but it is impossible to construct it. In this specific case we can prove that the sequence is recursive but there is no computable bound on the size of the program generating x that depends only on C .

We can explain informally why this happens (see [6, 7] for details) in the following way. Consider a sequence x that starts with a large number N of zeros that are followed by 1, then some string z and then zeros again. Any program that generates x gives us complete information about z (we have only to delete the leading zeros), and its complexity is high if z has high complexity. On the other hand the complexity $K(x_{1:n}|n)$ is low if $n \leq N$ (since only zeros appear in $x_{1:n}$ and can be low for $n > N$ since in this case we know some number n greater than N and this information may be useful for finding z).

1.4.4 Back to the definition of randomness

Our definition of “random sequence” (in this section we say “algo-random”) can be criticized from many different viewpoints. First, this definition uses the notion of an algorithm that was never used in probability theory. It leads to a natural question: is the notion of algorithm really necessary to give a reasonable definition of a random sequence?

Second, one could note that some easily definable sequences are algo-random. For example, there exists a sequence defined by G. Chaitin (called ω) that is algo-random. It is defined as follows. Consider an optimal algorithm in the sense described in Section 1.2, but in the self-delimiting version, and apply it to random bits obtained by coin tossing. (This algorithm will actually use only finitely many of these bits to produce the output.) The computation may terminate or not, depending on the choice of random bits. Then ω is defined as the probability of termination.⁽³⁾

This sequence is (as Chaitin noted) an algo-random one, and this raises a question. The proposition “ $x \neq \omega$ for almost all x ” is true (almost all sequences differ from ω). However, we cannot claim that “ $x \neq \omega$ for all algo-random x ”, since ω is one of them. Even if this example seems to be a little artificial, a true problem is raised.

The first possibility is to change the notion of algo-randomness, allowing a broader class of randomness tests. In this way we obtain a notion of “arithmo-random” sequences. Two formal definitions are possible: one considers the classical theory based on algorithms and then relativizes these algorithms using arithmetical oracles; the other one defines everything directly using arithmetic formulas. These two approaches lead to the same notion, which corresponds to a smaller class of random sequences. The problem is that this definition is not closed: there is no

³Similar experiment was performed at the early stages of Unix development. Some standard utilities were taken and sequences of random bits were fed into them. The probability of crash turned out to be embarrassingly large. (Authors' note)

universal test in the class considered. This is due to an important structural difference between the enumerable sets and the arithmetic sets: universal set exists for enumerable sets but not for the arithmetic ones.

Then we may make another, more radical, suggestion [5]: let us consider all the theorems of the form “for almost all x , $P(x)$ ”, where P is a formula in some language. There are countably many theorems of this form, and their set is recursively enumerable. Each of these theorems corresponds to a set of measure 1 (sequences for which P is true). Consider the intersection of all these sets. The σ -additivity (countable additivity) of the measure guarantees that this intersection also has measure 1. Let us take this intersection as the set of random sequences. Then by definition all the theorems of probability theory (provably true for almost all sequences) are true for the sequences from this set; however, we encounter then other difficulties (related to the basic problems in the foundations of set theory, like the absence of the set of all sets, etc.)

More subtle versions of this approach can be considered, but they are based on rather delicate techniques of the set theory. For example, instead a provably minimal set we may consider a set which would be minimal in a consistent way: this means that it is impossible to prove that it is not minimal. The existence of such a set is not at all evident; the proof makes use of fine techniques of the set theory. To give an informal image of this approach we may compare it with the presumption of innocence. A sequence must always be presumed random; if it is suspected not to be such, it must be taken to court; but in the absence of any proofs whatsoever of its “guilt” the sequence must be exonerated (that is, considered as random) for the benefit of the doubt.

Acknowledgement. The authors are grateful to Alexander Shen for many helpful comments.

Bibliography

- [1] A. Blass and Y. Gurevich, *Algorithms: a quest for absolute definitions*, Bull. EATCS **81** (Oct. 2003), p. 195-225.
- [2] G. Chaitin, *On the length of programs for computing finite binary sequences by bounded-transfer Turing machines*, AMS Notices **13** (1966), p. 133.
- [3] G. Chaitin, *On the length of programs for computing finite binary sequences by bounded-transfer Turing machines II*, AMS Notices **13** (1966), p. 228-229.
- [4] G. Chaitin, *Computational complexity and Gödel's incompleteness theorem*, AMS Notices **17** (1970), p. 672.
- [5] B. Durand, V. Kanovei, V. Uspensky and N. Vereshchagin, *Do stronger definitions of randomness exist?*, Theoret. Comput. Sci. **290** (2003), no. 3, p. 1987-1996.
- [6] B. Durand and S. Porrot, *Comparison between the complexity of a function and the complexity of its graph*, Theoret. Comput. Sci. **271** (2002), no. 1-2, p. 37-46.
- [7] B. Durand, A. Shen and N. Vereshchagin, *Descriptive complexity of computable sequences*, Theoret. Comput. Sci. **271**, (2002), no. 1-2, p. 47-58.
- [8] Y. Gurevich, *On Kolmogorov machines and related issues*, Bull. EATCS **35** (June 1988), p. 71-82.
- [9] D. Hofstadter, *Gödel, Escher, Bach: An Eternal Golden Braid*, Basic Books, 1979.
- [10] A. N. Kolmogorov, *Three approaches to the definition of the concept of "quantity of information"* (in Russian), Problemy Peredachi Informatsii **1** (1965), no. 1, p. 3-11.
- [11] A. N. Kolmogorov and V. A. Uspensky, *On the definition of an algorithm*, Translations, series 2, Amer. Math. Soc. **29** (1963), p. 217-245. (Translated from Uspekhi Mat. Nauk **13** (1958), no. 4, p. 3-28.)

- [12] L. Levin, *The concept of a random sequence*, Soviet Math. Doklady **212** (1973), p. 1413-1416.
- [13] M. Li and P. Vitányi, *An Introduction to Kolmogorov Complexity and its Applications*, Graduate Texts in Computer Science, Springer-Verlag (New York), 1997 (2nd edition).
- [14] D. W. Loveland, *A variant of Kolmogorov concept of complexity*, Information and Control **15** (1969), p. 510-526.
- [15] P. Martin-Löf, *The definition of random sequences*, Information and Control **9** (1966), p. 602-619.
- [16] C. P. Schnorr, *Process complexity and effective random sets*, J. Comput. System Sci. **7** (1973), p. 376-388.
- [17] D. Shasha and C. Lazere, *Out of Their Minds: The Lives and Discoveries of 15 Great Computer Scientists*, Copernicus, 1995.
- [18] R. J. Solomonoff, *A formal theory of inductive inference, I*, Information and Control **7** (1964), p. 1-22.
- [19] V. Uspensky and A. Semenov, *Algorithms: Main Ideas and Applications* (transl. from Russian by A. Shen), Kluwer, 1993.
- [20] V. Uspensky, A. Semenov and A. Shen, *Can an (individual) sequence of zeros and ones be random?*, Russian Math. Surveys **45** (1990), no. 1, p. 121-189.
- [21] A. K. Zvonkin and L. Levin, *The complexity of finite objects and the development of the concepts of information and randomness by means of the theory of algorithms*, Russian Math. Surveys **25** (1970), no. 6, p. 83-124.