Complexity of complexity and strings with maximal plain and prefix Kolmogorov complexity

Bruno Bauwens, Alexander Shen

The Journal of Symbolic Logic
Volume 00, Number 0, XXX 0000

COMPLEXITY OF COMPLEXITY AND STRINGS WITH MAXIMAL PLAIN AND PREFIX KOLMOGOROV COMPLEXITY

B. BAUWENS AND A. SHEN

Abstract. Péter Gács showed (Gács 1974) that for every n there exists a bit string x of length n whose plain complexity $C(x)$ has almost maximal conditional complexity relative to x, i.e., $C(C(x)|x) \geq \log n - \log^{(2)} n - O(1)$. (Here $\log^{(2)} i = \log\log i$.) Following Elena Kalinina (Kalinina 2011), we provide a simple game-based proof of this result; modifying her argument, we get a better (and tight) bound $\log n - O(1)$. We also show the same bound for prefix-free complexity.

Robert Solovay showed (Solovay 1975) that infinitely many strings x have maximal plain complexity but not maximal prefix complexity (among the strings of the same length): for some c there exist infinitely many x such that $|x| - C(x) \leq c$ and $|x| + K(|x|) - K(x) \geq \log^{(2)} |x| - c \log^{(3)} |x|$. In fact, the results of Solovay and Gács are closely related. Using the result above, we provide a short proof for Solovay's result. We also generalize it by showing that for some c and for all n there are strings x of length n with $n - C(x) \leq c$ and

$$n + K(n) - K(x) \geq K(K(n)|n) - 3\,K(K(K(n)|n)|n) - c.$$

We also prove a close upper bound $K(K(n)|n) + O(1)$.

Finally, we provide a direct game proof for Joseph Miller's generalization (Miller 2006) of the same Solovay's theorem: if a co-enumerable set (a set with c.e. complement) contains for every length a string of this length, then it contains infinitely many strings x such that

$$|x| + K(|x|) - K(x) \geq \log^{(2)} |x| - O(\log^{(3)} |x|).$$

Introduction. Plain Kolmogorov complexity $C(x)$ of a binary string x was defined in [6] as the minimal length of a program that computes x. (See, e.g., [4, 8, 12, 15] for more details.) It was clear from the beginning that this complexity function is not computable: no algorithm can compute $C(x)$ given x. In [3] (see also [4, 8]) a stronger non-uniform version of this result was proven: for every n there exists a string x of length n such that conditional complexity $C(C(x)|x)$,

Bruno Bauwens is supported by the Portuguese science foundation FCT (SFRH/BPD/75129/2010); and partially supported by CSI^2 (PTDC/EIA-CCO/099951/2008) and NAFIT ANR-08-EMER-008-01 projects. Alexander Shen is supported by NAFIT ANR-08-EMER-008-01 project. Part of this paper was published in the proceedings of the 39-th International Conference on Automata, Languages and Programming (ICALP2012). The authors are grateful to Elena Kalinina and Nikolay Vereshchagin for useful discussions.

i.e., the minimal length of a program that maps x to $C(x)$, is at least $\log n - \log^{(2)} n - O(1)$. (If the complexity function were computable, this conditional complexity would be bounded.)

In Section 1 we revisit this classical result and improve it a bit by removing the $\log^{(2)} n$ term.[1] No further improvement is possible because $C(x) \leq n + O(1)$ for every string x of length n, therefore $C(C(x)|x) \leq \log n + O(1)$ for all such x. We also prove that we can guarantee $C(x) \geq n/2$ (in addition to $C(C(x)|x) \geq \log n - O(1)$), which was (in weaker form) conjectured by Robert Solovay and Gregory Chaitin, and mentioned as Conjecture 3.14.6 on p. 145 in [2].

We use a game technique that was developed by Andrej Muchnik (see [11, 10, 14]) and turned out to be useful in many cases. Recently Elena Kalinina (in her master thesis [5]) used it to provide a proof of Gács' result. We use a more detailed analysis of essentially the same game to get a better bound.

In Section 2 we use this improved bound to provide a simple proof of an old result due to Solovay. The complexity $C(x)$ of an n-bit string x never exceeds $n+O(1)$, and for most n-bit strings x the value of $C(x)$ is close to n. Such strings may be called "C-random". There is another version of complexity, called prefix complexity, where the programs are assumed to be self-delimiting (see [4, 8, 12] for details). For an n-bit string x its prefix complexity $K(x)$ does not exceed $n+K(n)+O(1)$, and for most n-bit strings x the value of $K(x)$ is close to $n+K(n)$. Such strings may be called K-random[2].

A natural question arises: how "C-randomness" and "K-randomness" are related? This question was studied by Solovay who proved that K-randomness implies C-randomness but not vice versa (see the unpublished notes [13] and its exposition in [2]). More precisely, consider the "randomness deficiencies" $d_C(x) = |x| - C(x)$ and $d_K(x) = |x| + K(|x|) - K(x)$. Solovay proved that:

1. **$d_C(x) \leq O(d_K(x))$;**
2. **the reverse statement can be proved with additional error term:**
$$d_K(x) \leq O(d_C(x)) + \log^{(2)} n$$
for every n-bit string x;
3. **the error term cannot be deleted: there exists a constant c and infinitely many strings x such that $d_C(x) \leq c$ but**
$$d_K(x) \geq \log^{(2)} |x| - O(\log^{(3)} |x|).$$

Using the result of Section 1, we provide a short proof for statement (3), the most difficult one, even in a stronger form where $O(\log^{(3)} |x|)$ is replaced by $O(1)$. Then we prove a stronger statement about strings of fixed length n, with close lower and upper bounds:

[1] Note added in proof: alternatively, this improvement can also be shown using [1, Theorem 3.1] (and Theorem 5.1 for prefix complexity).

[2] In [2], such strings are called "strongly K-random", in contrast to the "weakly K-random" strings x, which only satisfy $K(x) \geq |x| - O(1)$.

- $d_K(x) \le O(d_C(x)) + K(K(n)|n)$ for every n-bit string x;
- for some constant c and for every n there exist a string x of length n such that $d_C(x) \le c$ and $d_K(x) \ge K(K(n)|n) - 3\,K(K(K(n)|n)|n) - c$.

It is stronger because the result of Section 1 then allows us to choose n in such a way that $K(K(n)|n) = \log^{(2)} n + O(1)$; this choice also makes the other term $O(1)$.

Finally, in Section 3 we give another example of game technique by presenting a simple proof of a different generalization of Solovay's result. This generalization is due to Miller [9]: every co-enumerable set (a set with c.e. complement) that contains a string of every length, contains infinitely many x such that

$$d_K(x) \ge \log^{(2)} |x| - O(\log^{(3)} |x|).$$

§1. Complexity of complexity can be high.

THEOREM 1. *There exist some constant c such that for every n there exists a string x of length n such that $C(C(x)|x) \ge \log n - c$.*

To prove this theorem, we first define some game and show a winning strategy for the game. (The connection between the game and the statement that we want to prove will be explained later.)

1.1. The game. Game G_n has parameter n and is played on a rectangular board divided into cells. The board has 2^n columns and n rows numbered $0, 1, \dots, n-1$ (the bottom row has number 0, the next one has number 1 and so on, the top row has number $n-1$), see Fig. 1.

Initially the board is empty. Two players: White and Black, alternate their moves. At each move, a player can pass or place a token (of his color) on the board. The token can not be moved or removed afterwards. Also Black may blacken some cell instead. Let us agree that White starts the game (though it does not matter).

The position of the game should satisfy some restrictions; the player who violates these restrictions, loses the game immediately. Formally the game is infinite, but since the number of (non-trivial) moves is a priori bounded, it can be considered as finite, and the winner is determined by the last (limit) position on the board.

Restrictions: (1) each player may put at most 2^i tokens in row i (thus the total number of black and white tokens in a row can be at most $2^i + 2^i$); (2) in each column Black may blacken at most half of the cells.

We say that a white token is *dead* if either it is on a blackened cell or has a black token in the same column strictly below it.

Winning rule: Black wins if he killed all white tokens, i.e., if each white token is dead in the final position.

For example, if the game ends in the position shown at Fig. 1, the restrictions are not violated (there are $3 \le 2^2$ white tokens in row 2 and $1 \le 2^1$ white token in row 1, as well as $1 \le 2^2$ black token in row 2 and $1 \le 2^0$ black token in row 0). Black loses because the white token

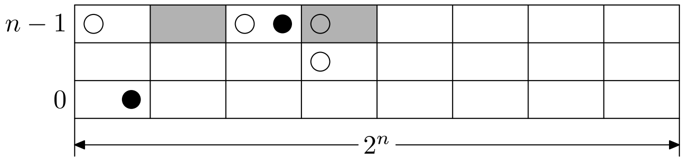

FIGURE 1. Game board

in the third column is not dead: it has no black token below and the cell is not blackened. (There is also one living token in the fourth column.)

1.2. How White can win. The strategy is quite simple. White starts by placing a white token in an upper row of some column and waits until Black kills it, i.e., blackens the cell or places a black token below. In the first case White puts a token directly below it, and waits again. Since Black has no right to make all cells in a column black (at most half may be blackened), at some point he will be forced to place a black token below the white token in this column. After that White switches to some other column. (The ordering of columns is not important; we may assume that White moves from left to right.)

Note that when White switches to a next column, it may happen that there is a black token in this column or some cells are already blackened. If there is already a black token, White switches again to the next column; if some cell is blackened, White puts her token in the topmost white (non-blackened) cell.

This strategy allows White to win. Indeed, Black cannot place his tokens in all the columns due to the restrictions (the total number of his tokens is $\sum_{i=0}^{n-1} 2^i = 2^n - 1$, which is less than the number of columns). White also cannot violate the restriction for the number of her tokens on some row i: all dead tokens have a black token strictly below them, so the number of them on row i is at most $\sum_{j=0}^{i-1} 2^j = 2^i - 1$, hence White can put an additional token.

In fact we may even allow Black to blacken all the cells except one in each column, and White will still win, but this is not needed (and the $n/2$ restriction will be convenient later).

1.3. Proof of Gács' theorem. Let us show that for each n there exists a string x of length n such that $C(C(x|n)|x) \geq \log n - O(1)$. Note that here $C(x|n)$ is used instead of $C(x)$; the difference between these two numbers is $O(\log n)$ since n can be described by $\log n$ bits, so the difference between the complexities of these two numbers is $O(\log^{(2)} n)$.

Consider the following strategy for Black (assuming that the columns of the table are indexed by strings of length n):

- Black blackens the cell in column x and row i as soon as he discovers that $C(i|x) < \log n - 1$. (The constant 1 guarantees that less than half of the cells will be blackened.) Note that Kolmogorov complexity is an upper semicomputable function, and Black approximates it from above, so more and more cells are blackened.

- Black puts a black token in a cell (x, i) when he finds a program of length i that produces x with input n (this implies that $C(x|n) \leq i$). Note that there are at most 2^i programs of length i, so Black does not violate the restriction for the number of tokens on any row i.

Let White play against this strategy (using the strategy described above). Since the strategy is computable, the behavior of White is also computable. One can construct a decompressor V for the strings of length n as follows: each time White puts a token in a cell (x, i), a program of length i is assigned to x. By White's restriction, no more than 2^i programs need to be assigned. By universality, a white token on cell (x, i) implies that $C(x|n) \leq i + O(1)$. If White's token is alive in (x, i), there is no black token below, so $C(x|n) \geq i$, and therefore $C(x|n) = i + O(1)$. Moreover, for a living token, the cell (x, i) is not blackened, so $C(i|x) \geq \log n - 1$. Therefore, $C(C(x|n)|x) \geq \log n - O(1)$.

Remark: the construction also guarantees that $C(x|n) \geq n/2 - O(1)$ for that x. (Here the factor $1/2$ can be replaced by any $\alpha < 1$ if we change the rules of the game accordingly.) Indeed, according to White's strategy, he always plays in the highest non-black cell of some column, and at most half of the cells in a column can be blackened, therefore no white tokens appear in the lower half of the board.

1.4. Modified game and proof of Theorem 1. Now we need to get rid of the condition n and show that for every n there is some x such that $C(C(x)|x) \geq \log n - O(1)$. Imagine that White and Black play simultaneously all the games G_n. Black blackens the cell (x, i) in game $G_{|x|}$ when he discovers that $C(i|x) < \log |x| - 1$, as he did before, and puts a black token in a cell (x, i) when he discovers an *unconditional* program of length i for x. If Black uses this strategy, he satisfies the stronger restriction: the total number of tokens in row i *on all boards* is bounded by 2^i.

Assume that White uses the described strategy on each board. What can be said about the total number of white tokens in row i? The dead tokens have black tokens strictly below them and hence the total number of them does not exceed $2^i - 1$. On the other hand, there is at most one living white token on each board. We know also that in G_n white tokens never appear below row $n/2 - 1$, so the number of alive white tokens does not exceed $2i + O(1)$. Therefore we have $O(2^i)$ white tokens on the i-th row in total.

For each n there is a cell (x, i) in G_n where White wins in G_n. Hence, $C(x) < i + O(1)$ (because of the property just mentioned and the computability of White's behavior), $C(x) \geq i$ and $C(i|x) \geq \log n - 1$ (by construction of Black's strategies and the winning condition). Theorem 1 is proven.

1.5. Version for prefix complexity.

Theorem 2. *There exist some constant c such that for every n there exists a string x of length n such that $C(K(x)|x) \geq \log n - c$ and $K(x) \geq n/2 - c$. This also implies that $K(K(x)|x) = \log n + O(1)$.*

The proof of $C(K(x)|x) \geq \log n - c$ goes in the same way. Black places a token in cell (x, i) if some program of length i for a prefix-free (unconditional) machine computes x (and hence $K(x) \leq i$) and blackens the cell if $K(i|x) < n - 1$; White uses the same strategy as described above. The sum of 2^{-i} for all black tokens is less than 1 (Kraft inequality); some white tokens are dead, i.e., strictly above black ones, and for each column the sum of 2^{-j} over these tokens (x, j), does not exceed $\sum_{j>i}^{n} 2^{-j} < 2^{-i}$. Hence the corresponding sum for all dead white tokens is less than 1; for the rest the sum is bounded by $\sum_n 2^{-n/2+1}$, so the total sum is bounded by a constant, and we conclude that for the token in the winning column x the row number is $K(x) + O(1)$, and this cell is not blackened.

It remains to note that $K(K(x)|x)$ is greater than $C(K(x)|x) \geq \log n - O(1)$ for x of length n; on the other hand, $n/2 \leq K(x) \leq 2n + O(1)$, so the length of $K(x)$ (in binary) is $\log n + O(1)$, and the conditional prefix complexity of a string given its length is bounded by the length, hence $K(K(x)|x) \leq K(K(x)|\log n) + O(1) \leq \log n + O(1)$.

Remark: In fact, $K(K(x)|x) \leq \log|x| + O(1)$ for all x (this will be useful in the next section). In general, if $z \leq O(n)$, then $K(z|\log n) \leq \log n + O(1)$, because we may add leading zeros to the binary representation of z up to length $\log n + O(1)$, and the prefix complexity of a string given its length does not exceed the length. (Note that for $z = K(x)$ and $n = |x|$ we have $z \leq O(n)$, and $K(K(x)|x) \leq K(K(x)|n) \leq K(K(x)|\log n)$.)

§2. Strings with maximal plain and prefix complexity. In this section we provide a new proof and a generalization for Solovay's result mentioned in the introduction. For completeness we first reproduce a proof of the simple upper bound for $d_C(x)$ in terms of $d_K(x)$.

Theorem 3 (Solovay [13]).

$$d_C(x) \leq O(d_K(x)).$$

Proof. Assume that $d_C(x)$ is large for some x of length n; we need to show that $d_K(x)$ is almost as large. Let $d_C(x)$ be equal to some c, and p be a plain program for x. Let $\hat{c}$ and $\hat{n}$ be the self-delimiting programs for c and n of length $O(\log c)$ and $K(n)$. Then $\hat{c}\hat{n}p$ is a self-delimiting program for x that gives prefix deficiency $c - O(\log c)$. ⊣

As we have mentioned, the reverse statement is not true: $d_K(x)$ can be big even if $d_C(x)$ is small. However, there exists an upper bound for d_K in terms of d_C and other complexities:

Theorem 4. *For any x of length n*

$$d_K(x) \leq O(d_C(x)) + K(K(n)|n).$$

Note that $K(K(n)|n) \le \log^{(2)} n + O(1)$ (see the remark that ends the previous section), so this bound implies the bound from [13] mentioned in the introduction).

PROOF. Let us denote $d_C(x)$ by c. As Levin noted, $C(x) = K(x|\,C(x)) + O(1)$ [7] (see also [8, p. 203]). So with $O(c)$-precision we have

$$n = C(x) = K(x|\,C(x)) = K(x|n).$$

Now we apply

$$K(u,v) = K(u) + K(v|u, K(u)) + O(1).$$

(additivity for prefix complexity, see, e.g., [3, 4, 8]) for $u = n$, $v = x$:

$$K(x) = K(n,x) = K(n) + K(x|n, K(n)),$$

all with $O(1)$-precision. Combining these two observations, we get

$$\begin{aligned} d_K(x) &= n + K(n) - K(x) = \\ &= (K(x|n) + O(c)) + K(n) - (K(n) + K(x|n, K(n))) = \\ &= K(x|n) - K(x|n, K(n)) + O(c). \end{aligned} \tag{2.1}$$

It is easy to see that $K(x|n) \le K(x|n, K(n)) + K(K(n)|n) + O(1)$, so $d_K(x)$ is bounded by $K(K(n)|n) + O(c)$. ⊣

Remark: With essentially the same proofs, we can replace terms $O(d_K(x))$ and $O(d_C(x))$ by $d_K(x) + O(\log d_K(x))$ and $d_C(x) + O(\log d_C(x))$ in Theorems 3 and 4.

The following theorem shows that *for all* n the second term in the bound of Theorem 4 is unavoidable (up to $O(\log K(K(n)|n))$ precision).

THEOREM 5. *For some c and all n there exists a string x of length n such that $d_C(x) \le c$, and*

$$d_K(x) \ge K(K(n)|n) - 3\,K(K(K(n)|n)|n) - c.$$

As we have said in the introduction, we can combine this result with Theorem 2 to obtain Solovay's result as corollary, even without $\log^{(3)}$-term:

COROLLARY 6. *There exists a constant c and infinitely many x such that $d_C(x) \le c$ and $d_K(x) \ge \log^{(2)} |x| - c$.*

Before proving Theorem 5, we prove the corollary directly.

PROOF. First we choose n, the length of string x, in such a way that

$$K(K(n)|n) = \log^{(2)} n + O(1)$$

and $K(n) \ge (\log n)/2 - O(1)$ (Theorem 2). We know already from equation (2.1) that for a string x with C-deficiency c the value of K-deficiency is $O(c)$-close to $K(x|n) - K(x|n, K(n))$. In other words, adding $K(n)$ to the condition n in $K(x|n)$ should decrease the complexity, so let us include $K(n)$ in x somehow. We also have to guarantee maximal C-complexity of x. This motivates the following choice:

- choose r of length $n - \log^{(2)} n$ such that $K(r|n, K(n)) \geq |r|$. Note that this implies $K(r|n, K(n)) = |r| + O(1)$, since the length of r is determined by the condition.
- Let $x = \langle K(n)\rangle r$, the concatenation of $K(n)$ (in binary) with r. Note that $\langle K(n)\rangle$ has at most $\log^{(2)} n + O(1)$ bits for every n, and by choice of n has at least $\log^{(2)} n - O(1)$ bits, hence $|x| = n + O(1)$.

As we have seen (looking at equation (2.1)), it is enough to show that

$$K(x|\, K(n), n) \leq n - \log^{(2)} n$$

and $K(x|n) = n$ (the latter equality implies $C(x) = n$, as Levin has noted[3]); all the equalities here and below are up to $O(1)$ additive term.

- Knowing n, we can split x in two parts $\langle K(n)\rangle$ and r. Hence, $K(x|\, K(n), n) = K(K(n), r|n, K(n))$, and this equals $K(r|n, K(n))$, i.e., $n - \log^{(2)} n$ by choice of r.
- To compute $K(x|n)$, we use additivity:

  $$K(x|n) = K(K(n), r|n) = K(K(n)|n) + K(r|\, K(n), K(K(n)|n), n).$$

  By choice of n, we have $K(K(n)|n) = \log^{(2)} n$, and the last term simplifies to $K(r|\, K(n), \log^{(2)} n, n)$, and this equals $K(r|\, K(n), n) = n - \log^{(2)} n$ by choice of r. Hence $K(x|n) = \log^{(2)} n + (n - \log^{(2)} n) = n$.

$\dashv$

Remark: One can ask how many strings are suitable for Corollary 6. By Theorem 4, the length n of such a string must satisfy $K(K(n)|n) \geq \log^{(2)} n - O(1)$. By Theorem 2, there is at least one such n for every $|n|$ (length of n as a binary string). Hence such n can be found within exponential intervals.

Then one can ask (for some n with this property) how many strings x of length n are suitable for Corollary 6. By a theorem of Chaitin [8], there are at most $O(2^{n-k})$ strings of length n with K-deficiency k, hence we can have at most $O(2^{n-\log^{(2)} n})$ such strings. It turns out that at least a constant fraction of them is suitable for Corollary 6. To show this, note that in the proof every different r of length $|n| - \log^{(2)} n + O(1)$ leads to a different x. For r we need $K(r|n, K(n)) \geq |r| - O(1)$, and there are $O(2^{n-\log^{(2)} n})$ such r.

The corollary is proved, and we proceed to the

Proof of Theorem 5. In the proof above, in order to obtain a large value $K(x|n) - K(x|n, K(n))$, we incorporated $K(n)$ directly in x (as $\langle K(n)\rangle$). To show that $C(x) = K(x|n) + O(1)$ is large, we used that the length of $\langle K(n)\rangle$ equals $K(K(n)|n) + O(1)$. For arbitrary n this trick does not work, but we can use a shortest program for $K(n)$ given n (on a plain machine) instead of $\langle K(n)\rangle$. For every n, we construct x as follows:

[3] We already mentioned the equation $C(x) = K(x|\, C(x)) + O(1)$, so $C(x)$ is a fixed point of the function $i \mapsto K(x|i)$ up to $O(1)$-precision. Since this function changes logarithmically slow compared to i, the reverse statement is also true: if $K(x|i) = i$ with some precision d, then $i = C(x)$ with $O(d)$-precision.

- let q be a shortest program that computes $K(n)$ from n on a *plain* machine (if there are several shortest programs, we choose the one that appears first, so it can be reconstructed from n and $K(n)$). Note that $|q| = C(K(n)|n) = C(q|n)+O(1)$ (remember that a shortest program is always incompressible). By Levin's result (conditional version: $C(u|v) = K(u|v, C(u|v))$), the last term also equals $K(q|n, |q|) + O(1)$;
- let r be a string of length $n - |q|$ such that $K(r|n, K(n), q) \geq |r|$. This implies $K(r|n, K(n), q) = |r| + O(1)$, since the length of r is determined by the condition.
- now we define x as the concatenation qr.

Now the proof goes as follows. We have to prove two things (together they obviously imply the statement of Theorem 5):

- that $C(x) = n + O(1)$, and $d_K(x) \geq |q| - K(|q|\,|n) + O(1)$.
- that $K(K(n)|n) - 3\,K(K(K(n)|n)|n) \leq C(K(n)|n) - K(C(K(n)|n)|n) + O(1)$.

The second part is a special case of the following

LEMMA 7. $K(a|b) - 3\,K(K(a|b)|b) \leq C(a|b) - K(C(a|b)|b) + O(1)$

for $a = K(n)$ and $b = n$. The proof of this lemma will be given after we finish the rest of the proof.

For the first part we follow the same structure as above. Using equation (2.1), we see that it is enough to show that with $O(1)$-precision (we omit $O(1)$-terms in the sequel) we have $K(x|\,K(n), n) \leq n - |q| + K(|q|\,|n)$ and $K(x|n) = n$ (the latter equality implies $C(x) = n$). Let us prove these two statements:

- Knowing $|q|$, we can split x in two parts q and r. Hence, $K(x|\,K(n), n, |q|) = K(q, r|n, K(n), |q|)$. Given $n, K(n), |q|$ we can search for a program of length $|q|$ that on input n outputs $K(n)$; the first one is q. Hence,

$$K(q, r|n, K(n), |q|) = K(r|n, K(n), |q|) = n - |q|$$

(the last equality is due to the choice of r), and therefore

$$K(x|\,K(n), n) \leq n - |q| + K(|q|\,|n).$$

- For $K(x|n)$ we use additivity:

$$K(x|n) \geq K(x|n, |q|) = K(q, r|n, |q|) = K(q|n, |q|) + K(r|q, K(q|n, |q|), n).$$

By choice of q we have $K(q|n, |q|) = |q|$. The last term is $K(r|q, |q|, n)$ and is equal to $K(r|q, n) = n-|q|$ by choice of r. Hence, $K(x|n) \geq |q|+(n-|q|) = n$. Since x is an n-bit string, we have also $K(x|n) \leq n$.

⊣

Theorem 5 is proved except for the proof of Lemma 7, which we give now.

PROOF. The condition b is used everywhere, so the statement is a conditional version of the inequality

$$K(a) - 3\,K(K(a)) \leq C(a) - K(C(a)) + O(1).$$

As usually, the proof of the conditional version follows the unconditional one, so we consider the unconditional version for simplicity. Note that $K(a) - C(a) \leq K(C(a))$. Indeed, every program p for plain machine can be converted to a self-delimiting one by adding a self-delimiting description of $|p|$ before p. Hence it remains to show that $2\,K(C(a)) \leq 3\,K(K(a)) + O(1)$. This follows from another Solovay's result from [13] (see also [2]) which says that

$$K(a) - C(a) = K(K(a)) + O(K(K(K(a)))).$$

From this result we conclude that

$$|\,K(K(a)) - K(C(a))| \leq O(\log K(K(a))),$$

and this is enough for our purpose. $\dashv$

§3. Game-theoretic proof of Miller's theorem. In this section we provide a simple game-based proof of a result due to Miller [9]; as we have seen in the introduction, this result implies that C-randomness differs from K-randomness. (The original proof in [9] uses a different scheme that involves the Kleene fixed-point theorem.)

Theorem 8 (J. Miller). *For any co-enumerable set Q of strings that contains a string of every length, there exist infinitely many x in Q such that $d_K(x) \geq \log^{(2)} |x| - O(\log^{(3)} |x|)$.*

Solovay's result follows by choosing Q to be the set of strings x such that $d_C(x) \leq c$ for large enough c (then Q contains strings of all lengths); this set is co-enumerable. One can also conclude that the set of strings x with $d_K(x) < c$ is not co-enumerable for large enough c (when this set contains strings of all lengths). One may also observe that because of Theorem 4, this result also implies a weak form of Gács' theorem: there exist infinitely many x such that $K(K(x)|x) \geq \log|x| - O(\log^{(2)} |x|)$.

Proof. Let us consider the following game specified by a natural number C and a finite family of disjoint finite sets $S_1, \ldots, S_N$. During the game each element $s \in S = \cup_{j=1}^{N} S_j$ is labeled by two non-negative rational numbers $A(s)$ and $B(s)$ called "Alice's weight" and "Bob's weight". Initially all weights are zeros. Alice and Bob make alternate moves. On each move each player may increase her/his weight of several elements $s \in S$.

Both players must obey the following restrictions for the total weight:

$$\sum_{s\in S} A(s) \leq 1 \quad \text{and} \quad \sum_{s\in S} B(s) \leq 1.$$

In addition, Bob must be "fair": for every j Bob's weights of all $s \in S_j$ must be equal. That means that basically Bob assigns weights to $j \in \{1, \ldots, N\}$ and Bob's weight $B(j)$ of j is then evenly distributed among all $s \in S_j$ so that

$$B(s) = B(j)/\#S_j$$

for all $s \in S_j$. Alice does not need to be fair.

This extra requirement is somehow compensated by allowing Bob to "disable" certain $s \in S$ (this does not decrease the size of S). Once an s is disabled it cannot be "enabled" any more. Alice cannot disable or enable anything. For every j, Bob is not allowed to disable *all* $s \in S_j$: every set S_j should contain at least one element that is enabled (=not disabled).

The game is infinite. Alice wins if at the end of the game (or, better to say, in the limit) there exists an enabled $s \in S$ such that

$$\frac{A(s)}{B(s)} \geq C.$$

Now we have to explain two things: why Alice has a (computable) winning strategy in the game (with some assumptions on the parameters of the game) and why this implies Miller's theorem.

LEMMA 9. *Assume that $N \geq 2^{8C}$ and $\#S_j \geq 8C$ for all $j \leq N$. Then Alice has a computable winning strategy.*

Let us show first why this statement implies the theorem. First we show how for a given c one can find some $x \in Q$ with $d_K(x) \geq c$. (Then we look more closely on the length of this x and check that indeed the statement of Theorem 8 is true.) Consider the following values of the game parameters:

$$C = 2^c \quad \text{and} \quad N = 2^{8C} = 2^{2^{c+3}}$$

Let us take the sets of all strings of length

$$\log 8C + 1, \dots, \log 8C + N$$

as $S_1, \dots, S_N$.

Consider the following strategy for Bob in this game. He enumerates the complement of Q and disables all its elements. In parallel, he approximates the prefix complexity from above; once he finds out that $K(n)$ does not exceed some l, he increases the weights of all 2^n strings of length n up to 2^{-l-n}. Thus at the end of the game $B(x) = 2^{-K(n)-n}$ for all $s \in S$ that have length n (i.e., for $s \in S_j$ where $j = n - \log 8C$). Note that Bob's total weight never exceeds its limit, since $\sum_n 2^{-K(n)} \leq 1$.

Alice's limit weight function $x \mapsto A(x)$ is lower semi-computable given c, as both Alice's and Bob's strategies are computable given c. Therefore, since prefix complexity is equal to the logarithm of a priori probability (coding lemma),

$$K(s|c) \leq -\log A(s) + O(1)$$

for all $s \in S$. As Alice wins, there exists a string $s \in Q$ of some length $n \leq N + \log 8C$ such that $A(s)/B(s) \geq C$, i.e.,

$$-\log A(s) \leq -\log B(s) - c = K(n) + n - c.$$

This implies that

$$K(s|c) \leq K(n) + n - c + O(1),$$

and

$$K(s) \leq K(n) + n - c + O(\log c).$$

Now let us look at the length of a string s constructed for a given c. The maximal possible length is $\log(8C) + N$, which is $O(N)$ since $N = 2^{8C}$ is much bigger than $\log(8C)$. So the length is at most

$$O(2^{2^{c+3}}).$$

In other terms, $c + 3 \geq \log^{(2)} |s| - O(1)$ and the deficiency of s is at least $c - O(\log c)$, which is at least $\log^{(2)} |s| - O(\log^{(3)} |s|)$. $\dashv$

It remains to prove the Lemma by showing a winning strategy for Alice.

Proof of Lemma 9. The strategy is rather straightforward. The main idea is that playing with one S_i, Alice can force Bob to spend twice more weight than she does. Then she switches to the next S_i, and so on until Bob's weight is exhausted while she has solid reserves. To achieve her goal on one set of M elements, Alice assigns sequentially weights $1/2^M, 1/2^{M-1}, \dots, 1/2^1$ and after each move waits until Bob increases his weight enough to satisfy the game requirements, or disables the corresponding element. Since he cannot disable all elements and is forced to use the same weights for all elements while Alice puts more than half of the weight on the last element, Bob has factor $M/2$ as a handicap, and we may assume that $M/2$ beats C-factor that Bob has in his favor.

Now the formal details. Assume first that $\#S_j = M = 4C$ for all j and $N = 2^M$. (We will show later how to adjust the proof to the case when $|S_j| \geq 8C$ and $N \geq 2^{8C}$.)

Alice picks an element $x_1 \in S_1$ and assigns the weight $1/2^M$ to x_1. Bob (to avoid losing the entire game) has either to assign a weight of more than $1/C2^M$ to all elements in S_1, or to disable x_1. In the second case Alice picks another element $x_2 \in S_1$ and assigns a (twice bigger) weight of $2/2^M$ to it. Again Bob has a dilemma: either to increase the weight for all elements of S_1 up to $2/C2^M$, or to disable x_2. In the second case Alice picks x_3, assigns a weight of $4/2^M$ to it, and so on. (If this process continues long enough, the last weight would be $2^{M-1}/2^M = 1/2$.)

As Bob cannot disable all the elements of S_1, at some step i the first case occurs, and Bob assigns a weight greater than $2^{i-1}/C2^M$ to all the elements of S_1. Then Alice stops playing on S_1. Note that the total Alice's weight of S_1 (let us call it β) is the sum of the geometric sequence:

$$\beta = 1/2^M + 2/2^M + \ldots + 2^{i-1}/2^M < 2^i/2^M \leq 1.$$

Thus Alice obeys the rules. Note that total Bob's weight of S_1 is more than $M2^{i-1}/C2^M = 2^{i+1}/2^M$, so it exceeds at least two times the total Alice's weight spent on S_1. This implies, in particular, that Bob cannot beat Alice's weight for the last element if the game comes to this stage (and Alice wins the game in this case.)

Then Alice proceeds to the second set S_2 and repeats the procedure. However this time she uses weights $\alpha/2^M, 2\alpha/2^M, \dots,$ where $\alpha = 1 - \beta$ is the weight still available for Alice. Again she forces Bob to use twice more weight than she does.

Then Alice repeats the procedure for the third set S_3 with the remaining weight etc.

Let β_j be the total weight Alice spent on the sets $S_1, \ldots, S_j$, and $\alpha_j = 1 - \beta_j$ the weight remaining after the first j iterations. By construction, Bob's total weight spent on sets $S_1, \ldots, S_j$ is greater than $2\beta_j$, so we have $2\beta_j < 1$ and hence $\alpha_j > 1/2$. Consequently, Alice's total weight of each S_j is more than $1/2^{M+1}$. Hence after at most $N = 2^M$ iterations Alice wins.

If the size of S_j are large but different, we need to make some modifications. (We cannot use the same approach starting with $1/2^M$ where M is the size of the set: if Bob beats the first element with factor C, he spends twice more weight than Alice but still a small amount, so we do not have enough sets for a contradiction.)

However, the modification is easy. If the number of elements in S_j is a multiple of $4C$ (which is the case we use), we can split elements of S_j into $4C$ groups of equal size, and treat all members of each group G as one element. This means that if the above algorithm asks to assign to an "element" (group) G a weight w, Alice distributes the weight w uniformly among members of G and waits until either Bob disables all elements of the group or assigns $4C$-bigger weight to all elements of S_j.

If S_j is not a multiple of $4C$, the groups are not equal (the worst case is when some groups have one element while other have two elements), so to compensate for this we need to use $8C$ instead of $4C$.

Note that excess in the number of groups (when N is bigger than required $8C$) does not matter at all, we just ignore some groups. ⊣

REFERENCES

[1] **R. Beigel, H.M. Buhrman, P. Feijer, L. Fortnow, P. Grabowski, L. Longpre, A. Muchnik, F. Stephan,** and **L. Torenvliet,** ***Enumerations of the Kolmogorov function***, this JOURNAL, **vol. 7 (2006), no. 501, pp. 501 – 528.**

[2] **R.G. Downey** and **D.R. Hirschfeldt,** ***Algorithmic randomness and complexity***, **Theory and Applications of Computability, Springer, 2010.**

[3] **P. Gács,** ***On the symmetry of algorithmic information***, ***Soviet Math. Dokl.***, **vol. 15 (1974), pp. 1477–1480.**

[4] ———, ***Lecture notes on descriptional complexity and randomness***, http://www.cs.bu.edu/faculty/gacs/papers/ait-notes.pdf, **1988–2011.**

[5] **E. Kalinina,** ***Some applications of the method of games in Kolmogorov complexity***, ***Master's thesis***, **Moscow State University, 2011.**

[6] **A.N. Kolmogorov,** ***Three approaches to the quantitative definition of information***, ***Problemy Peredachi Informatsii***, **vol. 1 (1965), no. 1, pp. 3–11.**

[7] **L.A. Levin,** ***The various measures of the complexity of finite objects (an axiomatic description)***, ***Soviet Mathematics Doklady***, **vol. 17 (1976), no. 2, pp. 522–526.**

[8] **M. Li** and **P.M.B. Vitányi,** ***An introduction to Kolmogorov complexity and its applications***, **Springer-Verlag, New York, 2008.**

[9] **J.S. Miller,** ***Contrasting plain and prefix-free Kolmogorov complexity***, **unpublished, 2006.**

[10] **A. Muchnik,** ***On the basic structures of the descriptive theory of algorithms***, ***Soviet Mathematics Doklady***, **vol. 32 (1985), pp. 671–674.**

[11] A. A. Muchnik, I. Mezhirov, A. Shen, and N. Vereshchagin, *Game interpretation of Kolmogorov complexity*, unpublished, mar 2010.

[12] A. Shen, *Algorithmic information theory and Kolmogorov complexity*, *Technical Report 2000-034*, Department of Information Technology, Uppsala University, and Independent University of Moscow, Russia, December 2000.

[13] R.M. Solovay, *Draft of a paper (or series of papers) on Chaitin's work.*, 215 pp., unpublished, May 1975.

[14] N. Vereshchagin, *Kolmogorov complexity and games*, *Bulletin of the European Association for Theoretical Computer Science*, vol. 94 (2008), pp. 51–83.

[15] A.K. Zvonkin and L.A. Levin, *The complexity of finite objects and the development of the concepts of information and randomness by means of the theory of algorithms*, *Russian Mathematical Surveys*, vol. 25 (1970), no. 6:156, pp. 83–124.

LORIA, UNIVERSITÉ DE LORRAINE, 615-B248 RUE DU JARDIN BOTANIQUE, 54506 VANDŒVRE-LÈS-NANCY, FRANCE *URL*, WWW.BCOMP.BE: .

LIRMM CNRS & UNIVERSITY OF MONTPELLIER, 2. UMR 5506 - CC477, 161 RUE ADA, 34095 MONTPELLIER CEDEX 5, FRANCE. ON LEAVE FROM IITP RAS, MOSCOW.