On infinite sequences (almost) as easy as π

José L. Balcázar* Ricard Gavaldà* Montserrat Hermo♯

June 21st, 1994

Abstract. It is known that infinite binary sequences of constant Kolmogorov complexity are exactly the recursive ones. Such a kind of statement no longer holds in the presence of resource bounds. Contrary to what intuition might suggest, there are sequences of constant, polynomial-time bounded Kolmogorov complexity that are not polynomial-time computable. This motivates the study of several resource-bounded variants in search for a characterization, similar in spirit, of the polynomial-time computable sequences. We propose some definitions, based on Kobayashi's notion of compressibility, and compare them to the standard resource-bounded Kolmogorov complexity of infinite strings. Some non-trivial coincidences and disagreements are proved and, in particular, they coincide for all usual bounds that are at least logarithmic. The resource-unbounded case is also considered.

---

* Department of Software (LSI), Universitat Politècnica de Catalunya, Pau Gargallo 5, E-08028 Barcelona, Spain. Partially supported by the ESPRIT Basic Research Actions Program of the EC under contract No. 7141 (project ALCOM II) and by DGICYT, project number PB92–0709.

♯ Departamento de Lenguajes y Sistemas Informáticos, Universidad del País Vasco, Apdo. 649, E-20080 San Sebastián, Spain.

E-mail addresses: `balqui@lsi.upc.es`, `gavalda@lsi.upc.es`, `montse@si.ehu.es`

1. Introduction

Most real numbers one can think of, such as π, have the property that there is a mechanical procedure to obtain the successive bits in its binary expansion. It is immediate that such a procedure exists if and only if the set of positions where the bit is 1 is recursive.

Moreover, frequently such procedures require a "feasible" amount of time to produce their output, in the sense that there is a not too large upper bound on the time elapsed until the n-th bit is produced. For instance, we will say that a sequence is polynomial-time constructible if there is a polynomial n^k that bounds the time needed to obtain the n-th bit.

The definition of Kolmogorov complexity of a finite string can be extended to infinite strings, where additionally classes of functions can be used as bounds. One says that the Kolmogorov complexity of the infinite binary sequence α is in $\mathcal{F}$ if there is a function $f \in \mathcal{F}$ such that each finite prefix of α, of length n, can be recovered from (n plus) a string of length $f(n)$. A beautiful result due to Meyer (described in [Lov69]) characterizes the recursive sequences: these are exactly those having $O(1)$ Kolmogorov complexity.

Kobayashi introduces an interesting variant on this theme [Kob81]: he says that α is f-compressible when there is another infinite sequence β such that, for all n, the first n bits of α can be obtained from n plus the first $f(n)$ bits of β. Thus, not only each prefix can be recovered from a string of length $f(n)$, but also these strings are highly correlated, in that the short ones are prefixes of the long ones, thus possibly extending to one single infinite word.

One purpose of this paper is to analyze how robust this notion is, by proposing five small, very natural variants and studying whether they coincide. The case in which resource bounds are imposed is also considered. We also compare them to the definition given by Kolmogorov complexity, and to recursive (or polynomial-time computable) sequences.

We focus essentially on "easy" sequences. Our results are as follows:

1. For constant bounds on the complexity and polynomial time, three of our definitions exactly coincide with Kobayashi's, and characterize the polynomial-time computable sequences. The simulations are nontrivial. The others are provably different, and in particular constant polynomial-time Kolmogorov complexity does *not* characterize polynomial-time computable sequences. This contrasts sharply with the recursive case. It actually follows from our proof that the sequences of constant polynomial-time Kolmogorov complexity are not included in DTIME(t) for any recursive time bound t.

2. For bounds whose growth rate is at least that of a logarithm, and under (small) constant factors, all the definitions coincide. The proof works both in the presence and in the absence of reasonable time bounds.

2. Preliminaries and definitions

Kolmogorov complexity, in many disguises, provides a means of defining the quantity of information contained in a finite string [LV93]. It is usually defined as the size of a minimal program able to instruct a universal Turing machine to produce the string. Of course, for a fixed, individual string, its complexity is a fixed number. Considering sets of strings, it is possible to see their complexity as some kind of function, and allows one to abstract from a few aspects of low relevance, e.g. by leaving unspecified precisely which universal machine is used, or not computing precisely some constants.

One natural case arises when an infinite binary sequence is considered. Then the set of finite strings can be taken as the set of all prefixes of that sequence, and the complexity of the sequence itself can be defined by the rate of growth of the function that associates, to each natural n, the complexity of the n-th prefix of the sequence. We study this case.

Of course we are interested in the infinite sequence itself, but it is well known that the finite prefixes contain additional information furnished by their length. To overcome this small technical problem, when working with finite prefixes we provide always the length of the desired output, n. This is done by starting the universal Turing machine in a configuration in which n is written in a separate tape. In this case, if the input tape contains a word u, we say that the machine works "on input u and n". We will accept also to start the machine in a configuration in which an infinite sequence appears in the input tape.

Definition 1. For classes of functions $\mathcal{F}$ and $\mathcal{G}$, denote by $K[\mathcal{F},\mathcal{G}]$ the set of infinite binary sequences α such that, for some $f \in \mathcal{F}$ and $g \in \mathcal{G}$, and for all n, there is a string u of length $f(n)$ such that the universal Turing machine on input u and n outputs $\alpha_{1:n}$ in time $g(n)$.

We will use the symbol ∞ in place of $\mathcal{G}$ to mean that the time bound is not being considered; and similarly for the forthcoming definitions.

Observe that u provides not only its own bit pattern as information, but also the information contained in its length $f(n)$. If we want to avoid this spurious source of information, we could request that the universal Turing machine has as input an infinite string from which it only reads the first $f(n)$ symbols. The ratio between both measures is small, for we can always include the value of $f(n)$ into the seed and make it "self-delimiting" [LV93], at the cost of at most doubling the length.

Still, the idea of the infinite string motivates the resource-bounded compressibility of Kobayashi: just swap quantifiers, and request that the infinite source string be always the same for all prefixes. What follows is not the exact formulation of Kobayashi, but, in the absence of resource bounds, it is easily seen to coincide with it. We denote it by C_1 since we will propose other variants.

Definition 2. For classes of functions $\mathcal{F}$ and $\mathcal{G}$, denote by $C_1[\mathcal{F},\mathcal{G}]$ the set of infinite binary sequences α such that there is an infinite sequence β, which, for some $f \in \mathcal{F}$ and $g \in \mathcal{G}$, and for all n, makes the universal Turing machine on input β and n output $\alpha_{1:n}$ in time $g(n)$; moreover, during the process it does not read past the $f(n)$-th bit of β.

An intuitive (but somewhat inexact) view of this definition is: as before, we have available $f(n)$ bits to produce the first n bits of α. But now we want the source bits for each prefix to be highly correlated, instead of completely independent; we demand that they are the first $f(n)$ bits of a single infinite sequence β.

The very fact that this view is not exact motivates the variants we propose. The inexactness comes from the ambiguity present in the intuitive explanation, as to whether the value of $f(n)$ itself is or is not available, and to how to formalize the independence of the values of β past $f(n)$. We propose the following variants:

Definition 3. For classes of functions $\mathcal{F}$ and $\mathcal{G}$, denote by $C_2[\mathcal{F},\mathcal{G}]$ the set of infinite binary sequences α such that there is an infinite sequence β, which, for some $f \in \mathcal{F}$ and $g \in \mathcal{G}$, and for all n, makes the universal Turing machine on input β and n output $\alpha_{1:n}$ in time $g(n)$; moreover, the same output is obtained if any other sequence γ is used, provided that $\gamma_{1:f(n)} = \beta_{1:f(n)}$.

Definition 4. For classes of functions $\mathcal{F}$ and $\mathcal{G}$, denote by $C_3[\mathcal{F},\mathcal{G}]$ the set of infinite binary sequences α such that there is an infinite sequence β, which, for some $f \in \mathcal{F}$ and $g \in \mathcal{G}$, for all n, and for all $m \geq f(n)$, makes the universal Turing machine on input $\beta_{1:m}$ and n output $\alpha_{1:n}$ in time $g(n)$.

Definition 5. For classes of functions $\mathcal{F}$ and $\mathcal{G}$, denote by $C_4[\mathcal{F},\mathcal{G}]$ the set of infinite binary sequences α such that there is an infinite sequence β, which, for some $f \in \mathcal{F}$ and $g \in \mathcal{G}$, and for all n, makes the universal Turing machine on input $\beta_{1:f(n)}$ and n output $\alpha_{1:n}$ in time $g(n)$.

Furthermore, later we will impose some conditions on the functions in $\mathcal{F}$. It will be helpful to have a fifth definition of compressibility available, although it coincides with the fourth one for most natural bound families $\mathcal{F}$.

Definition 6. For classes of functions $\mathcal{F}$ and $\mathcal{G}$, denote by $C_5[\mathcal{F},\mathcal{G}]$ the set of infinite binary sequences α such that there is an infinite sequence β, which, for some $f \in \mathcal{F}$ and $g \in \mathcal{G}$, for all n, and for some $k \leq f(n)$, makes the universal Turing machine on input $\beta_{1:k}$ and n output $\alpha_{1:n}$ in time $g(n)$.

The following facts can be checked routinely:

Proposition 7. For all $\mathcal{F}$ and $\mathcal{G}$,

$$C_1[\mathcal{F},\mathcal{G}] \subseteq C_2[\mathcal{F},\mathcal{G}] \subseteq C_3[\mathcal{F},\mathcal{G}] \subseteq C_4[\mathcal{F},\mathcal{G}] \subseteq C_5[\mathcal{F},\mathcal{G}] \subseteq K[\mathcal{F},\mathcal{G}].$$

Proposition 8. Whenever $f(n) \leq f'(n)$, we have $C_1[f,\mathcal{G}] \subseteq C_1[f',\mathcal{G}]$, and similarly with C_2, C_3, and C_5.

It is easy to see that, for this last one, similar proofs do not work for C_4. Actually, it follows from a later result that for C_4 this proposition does not hold.

Intuitively, one does not expect strange functions to be selected as bounds for the complexity; however, no explicit conditions have been stated. One condition that is worth studying is monotonicity: one may expect that imposing this condition could prevent the somewhat anomalous behavior shown in the next section. We will see that this is not the case either.

Definition 9. MON is the class of all functions f that are monotonic, that is, $f(n) \leq f(n+1)$ for every n. For all $\mathcal{F}$ and $\mathcal{G}$ and every $i = 1 \ldots 5$, $mC_i[\mathcal{F},\mathcal{G}]$ is defined as $C_i[\mathrm{MON} \cap \mathcal{F},\mathcal{G}]$, and $mK[\mathcal{F},\mathcal{G}]$ is defined as $K[\mathrm{MON} \cap \mathcal{F},\mathcal{G}]$.

It is also straightforward to verify the following inclusions.

Proposition 10. For all $\mathcal{F}$ and $\mathcal{G}$,

$$mC_1[\mathcal{F},\mathcal{G}] \subseteq mC_2[\mathcal{F},\mathcal{G}] \subseteq mC_3[\mathcal{F},\mathcal{G}] \subseteq mC_4[\mathcal{F},\mathcal{G}] \subseteq mC_5[\mathcal{F},\mathcal{G}] \subseteq mK[\mathcal{F},\mathcal{G}].$$

3. Constant complexity

It was shown by Meyer that $K[O(1),\infty]$ coincides with the recursive sequences [Lov69]. It is easy to see that also $C_1[O(1),\infty]$ does, as was observed in [Kob81]. Notice that one can extend this fact to $C_2[O(1),\infty]$ and $C_3[O(1),\infty]$:

Proposition 11. The following classes coincide:

1/ the recursive sequences;

2/ $C_1[O(1),\infty]$;

3/ $C_2[O(1),\infty]$;

4/ $C_3[O(1),\infty]$.

This fact carries over also to polynomial time, and actually to other time bounds:

Proposition 12. The following classes coincide:

1/ the polynomial-time computable sequences;

2/ $C_1[O(1),\text{poly}]$;

3/ $C_2[O(1),\text{poly}]$;

4/ $C_3[O(1),\text{poly}]$.

The remaining three definitions coincide for constant bounds but, as we shall see, differ from the previous ones.

Proposition 13. $C_4[O(1),\text{poly}] = C_5[O(1),\text{poly}] = K[O(1),\text{poly}]$.

Proof. By Proposition 7, it suffices to show $K[O(1),\text{poly}] \subseteq C_4[O(1),\text{poly}]$. Fix an infinite sequence α in $K[O(1),\text{poly}]$ and note that there is a finite number of seeds $s_0, s_1, \ldots, s_k$ such that each prefix of α. is generated by some s_i.

Let $P \in \{0,1\}^\star$ be a program for the universal machine that contains $s_0, \ldots, s_k$ as constants and executes as follows: scan the program tape and verify that it contains a string of the form $x\#0^i$, with $x \in \{0,1\}^\star$; if so, execute s_i, otherwise halt without valid output.

Thus, each prefix of α is generated from a prefix of appropriate length of the sequence $P\#0^\infty$, and only a finite number of prefixes of $P\#0^\infty$ are needed in total because there is a finite number of strings s_i. Furthermore, the time needed is dominated by the computation time of s_i. This proves that α is in $C_4[O(1),\text{poly}]$. $\square$

The main result of this section is the following:

Theorem 14. For every total recursive function f there is an infinite sequence that is in $C_4[O(1),\text{poly}]$ but not DTIME(f)-computable. (The sequence is automatically in $C_5[O(1),\text{poly}]$ and in $K[O(1),\text{poly}]$.)

Proof. Using diagonalization, build a tally set $T \subseteq 0^\star$ with the following properties:

1. χ_T is not DTIME(f)-computable, where χ_T is the characteristic sequence of T over $0^\star$;
2. $T \in$ DTIME(g), for some increasing time-constructible function $g > f$;
3. strings in T are very far one from another; more precisely, T contains only strings of the form $0^{s(m)}$, where s is defined inductively by:

$$s(1) = 1; \qquad s(m+1) = g(s(m)).$$

(This poses no problem: when diagonalizing, it is always possible to choose the new diagonalization point as far as we need.)

We show that $\chi_T \in C_4[O(1), \text{poly}]$. The crucial observation is the following: Given some n, it is possible to decide membership in T in time linear in n for all strings of length $\leq n$, except possibly for one. Indeed: consider the maximum m such that $s(m+1) \leq n$. Strings $0^{s(i)}$, with $i \leq m$, are decided in time $g(s(i)) \leq g(s(m)) \leq s(m+1) \leq n$. And listing all 0^j such that $j = s(i)$ for some $i \leq m+1$ takes time at most $g(s(0)) + g(s(1)) + \ldots + g(s(m)) \leq O(n^2)$ (the time-constructibility of g is used here). So the only "hard bit" in building $\chi_{T1:n}$ might be deciding "$0^{s(m+1)} \in T$".

Now define two programs P_N and P_Y. Given n, P_N builds a sequence of n bits as follows: it decides in time $O(n^2)$ all bits but one of $\chi_{T1:n}$. For the "hard bit" that might take superlinear time to decide, it says "NO". P_Y is identical to P_N except that when in doubt about that hard bit it says "YES".

For any n, one of P_Y and P_N outputs correctly $\chi_{T1:n}$ in quadratic time, so $\chi_T \in K[O(1), \text{poly}] = C_4[O(1), \text{poly}]$.

Note that, of course, both P_Y and P_N will correctly decide any particular "hard bit" when they are given larger and larger n's. □

Corollary 15. $K[O(1), \text{poly}]$ is not the set of polynomial-time computable sequences.

The complexity bounds that we use most often are all monotonic, and in this case the situation changes slightly. Indeed, observe that the proof of Proposition 13 uses the value of $f(n)$ to select the appropriate program. Hence, f goes back and forth and is not monotonic. This fact is crucial for that result.

Proposition 16. $mC_4[O(1), \text{poly}]$ is exactly the set of polynomial-time computable sequences.

Hence, monotonicity makes a provable difference for $C_4[O(1), \text{poly}]$. For constant bounds, it is easily shown that it makes no difference for all the other definitions.

Proposition 17. For constant bounds, C_1, C_2, C_3, C_5, and K coincide with their respective monotonic variants.

So, with the monotonicity conditions, all the definitions from C_1 to C_4 characterize the polynomial time computable sequences, while C_5 and K provably do not, as follows from Corollary 15.

4. Logarithmic complexity

In this section we prove that all our definitions coincide for many interesting bounds above logarithmic, modulo (small) constant factors and polynomial time overheads. Our result applies to all bounds with a mild assumption on their growth rate.

We first present the result for the logarithmic bounds, which is an interesting case in itself and provides most intuitions for the general proof. In short, infinite words having Kolmogorov complexity bounded by a function in $O(\log n)$, are also $O(\log n)$-compressible, under all the definitions given, and consuming essentially the same resources in both cases. The technique is based in a method presented in [BHM92], called there "double exponential skip". We will prove that, whenever a sequence has logarithmically long seeds s_n for each length n, then it is possible to construct a single infinite β, whose prefixes can also be used, within a reasonable cost, as seeds for each length.

Theorem 18.

$$C_1[\log, \text{poly}] = C_2[\log, \text{poly}] = C_3[\log, \text{poly}] = C_4[\log, \text{poly}] = C_5[\log, \text{poly}] = K[\log, \text{poly}].$$

Proof. By Proposition 7, we only have to show that $K[\log, \text{poly}] \subseteq C_1[\log, \text{poly}]$. Suppose that α has seeds $\{s_n\}$, with $|s_n| < k \log n$, for every length n, that is, on input s_n and n, the universal machine produces $\alpha_{1:n}$ in time polynomial in n. Here k is some constant. Padding up each s_n, and possibly using a larger constant, we can assume without loss of generality that $|s_n| = k \log n$.

The infinite sequence β is constructed by appending the seeds corresponding to double powers of 2: those of the form $s_{2^{2^n}}$. We prepend $k-1$ place-holder 0's that simplify our later calculations. Therefore

$$\beta = 0^{k-1} s_2 s_{2^{2^1}} s_{2^{2^2}} \ldots s_{2^{2^n}} \ldots$$

It is not difficult to check that, for all n, $s_{2^{2^m}} = \beta_{k2^m:k2^{m+1}-1}$, where k is the same constant as before. It suffices to add up the lengths of the previous formatted seeds. With this β, we can prove that α is $h(n)$-compressible, where $h(n)$ is $O(\log n)$. The algorithm that prints out the sequence is the following:

> on input β and n,
> find m such that $2^{2^{m-1}} < n \leq 2^{2^m}$;
> extract $s = \beta_{k2^m:k2^{m+1}-1}$;
> simulate the universal machine on s and 2^{2^m};
> from the result $\alpha_{1:2^{2^m}}$ output $\alpha_{1:n}$;

As indicated above, the s used is exactly $s_{2^{2^m}}$, so that the output is precisely $\alpha_{1:2^{2^m}}$ as needed. We read from β less than the first $4k2^{m-1} \leq 4k \log n$ bits, and take time polynomial in $2^{2^m} \leq n^2$. Thus, α is in $C_1[\log, \text{poly}]$. □

As an interesting aside, let us mention that it turns out to be crucial to keep exactly the seeds corresponding to the lengths 2^{2^m}. It may seem that a single exponential separation should suffice; but this fails because there are logarithmically many smaller seeds of logarithmic length to be encoded, i.e. a total of $\log^2 n$ bits, that is too long. Surprisingly, a double exponential works. However, if we try a triply exponential gap, corresponding to lengths $2^{2^{2^m}}$, then these seeds are too large although there are less of them: the first appropriate m would be such that $2^{2^{2^{m-1}}} < n \leq 2^{2^{2^m}}$, and a straightforward computation shows that s becomes $n^{\log n}$ long.

This result can be generalized in a number of ways. An easy observation is that the time bounds can be different from the polynomials, provided that the class $\mathcal{G}$ of time bounds used has a closure property with respect to composition with polynomials: for all $g(n) \in \mathcal{G}$, and for every polynomial $p(n)$, $g(p(n)) \in \mathcal{G}$. Furthermore, with some additional work, it is possible to generalize the result to other families of complexity bounds. The condition that we require is that the bounding function does not have very long stretches where it hardly grows.

Theorem 19. Let f be nondecreasing, and such that $f(n^2) \geq 2f(n)$. Let $\mathcal{G}$ be a family of running times closed under composition with polynomials and $O(.)$. Then $K[O(f), \mathcal{G}] = C_i[O(f), \mathcal{G}]$, for i from 1 to 5.

Note that any function f satisfying $f(n^2) \geq 2f(n)$ is $\Omega(\log n)$, and that natural bounds such as $\log^k n$, $n^{1/k}$, and n/k for constant k satisfy this condition. However, not all functions in $\Omega(\log n)$ satisfy it: it is easy to build a function that grows linearly for a while, then remains constant for an exponentially long stretch until it meets with the logarithmic function, then starts to grow linearly again, and so on.

Proof. It is enough to prove that $K[O(f), \mathcal{G}] \subseteq C_1[O(f), \mathcal{G}]$. Let $\alpha \in K[O(f), \mathcal{G}]$ via the sequence of seeds w_n, so that $|w_n| \leq f(n)$ and the universal machine produces $\alpha_{1:n}$ from w_n and n in time bounded by a function in $\mathcal{G}$. Define the following function:

$$h(n) = \max\{m \mid \exists i \ : \ 2^{i-1} < f(n) \leq f(m) \leq 2^i\}.$$

That is, we divide the range of f in intervals given by powers of 2, and let $h(n)$ be the last point before f jumps to a larger interval. Since f is nondecreasing and at least doubles for n^2, we have $h(n) \leq n^2$.

For the sequence β, we select all seeds w_m for m in the range of h, together with the length that they are good for. More precisely, let $\{ m_1, m_2, m_3, \dots \}$ be the range of h, where $2^{i-1} < f(m_i) \leq 2^i$. Translate each m_i into binary in a standard fashion, and define β as

$$\beta = w_{m_1}\#m_1\#w_{m_2}\#m_2\#w_{m_3}\#m_3\#\dots$$

The program to output $\alpha_{1:n}$ from β and n searches in β for an i such that $m_{i-1} < n \leq m_i$. When found, feed w_{m_i} and m_i into the universal machine and output only the first n bits of the result.

As $m_i = h(n)$, the running time of this procedure is $O(g(h(n))) \leq O(g(n^2))$ for some $g \in \mathcal{G}$, which remains in $\mathcal{G}$ since $\mathcal{G}$ is closed under composition with polynomials and $O(.)$.

The prefix of β that is actually read consists of all pairs $w_{m_j}\#m_j\#$, with $m_j \leq m_i = h(n)$. The number of such pairs is at most $\lceil \log f(n) \rceil$, as there is at most one value of m_j for each interval $(2^j, 2^{j+1}]$ and $\lceil \log f(n) \rceil = \lceil \log f(m_i) \rceil$.

Furthermore, by the condition $f(n^2) \geq 2f(n)$, there must be a constant c such that $f(n) \geq c \log n$. Ignoring a small term introduced by the binary translation of the separators, the number of bits read from β is at most

$$\begin{aligned}
\sum_{j=1}^{\lceil \log f(n) \rceil} |w_{m_j}\#m_j\#| &\leq \sum_{j=1}^{\lceil \log f(n) \rceil} (|w_{m_j}| + \log m_j) \\
&\leq \sum_{j=1}^{\lceil \log f(n) \rceil} f(m_j) + c\, f(m_j) \\
&\leq (c+1) \sum_{j=1}^{\lceil \log f(n) \rceil} 2^j \\
&\leq (c+1)\, 2^{\lceil \log f(n) \rceil + 1} \leq 4\,(c+1)\, f(n).
\end{aligned}$$

This proves that $\alpha \in C_1[O(f), \mathcal{G}]$ □

If the bounding function f is itself computable with the resources in $\mathcal{G}$, we can give a slightly better result: we can replace the condition "$f(n^2) \geq 2f(n)$" by the weaker "$f(n^k) \geq 2f(n)$ for a fixed k". This latter condition holds, for example, by all bounds of the form $\log^{\epsilon} n$.

Theorem 20. Let f be nondecreasing, and such that $f(n^k) \geq 2f(n)$ for a fixed k. Let $\mathcal{G}$ be a family of time-constructible functions closed under composition with polynomials and $O(.)$. Assume furthermore that $\lceil \log f(n) \rceil$ can be computed from n in time bounded by a function in $\mathcal{G}$. Then $K[O(f), \mathcal{G}] = C_i[O(f), \mathcal{G}]$, for i from 1 to 5.

The proof is very similar to that of Theorem 19, forming β by concatenating all strings $w_{m_i}\#$ instead of $w_{m_i}\#m_i\#$. If $\lceil \log f(n) \rceil$ is computable in $\mathcal{G}$, so is $h(n)$, so the correct m_i can be computed from n alone before searching for the appropriate seed in β.

Let us consider the resource unbounded case. In the proofs before for every n we had to include a seed w_m with m not much larger than n. Otherwise, as we compute $\alpha_{1:m}$ in order to get $\alpha_{1:m}$, the computation time will be too large. The condition on the growth of f ensured that such an m not far from n can be always found. If time is not an issue, we can drop the strict requirement on the growth rate for all bounds above the logarithm.

Theorem 21. Let f be nondecreasing and $\Omega(\log n)$. Then $K[O(f), \infty] = C_i[O(f), \infty]$, for i from 1 to 5.

Finally, if we do not consider time and take only recursive bounds, we can combine the ideas in the last two theorems and obtain the equalities for sublogarithmic bounds. It is remarkable that our proof technique fails for functions f that are bounded, precisely the case covered by Meyer's result.

Theorem 22. For every recursive and nondecreasing f, $K[O(f), \infty] = C_i[O(f), \infty]$, for i from 1 to 5.

5. References

[BHM92] J.L. Balcázar, M. Hermo, and E. Mayordomo: "Characterizations of logarithmic advice complexity classes", In *Algorithms, Software, Architecture: Information Processing 92,* Vol. I, 315–321. Elsevier, 1992.

[Kob81] K. Kobayashi: "On compressibility of infinite sequences". Research Reports on Information Sciences, C–34. Tokyo Institute of Technology, 1981.

[LV93] M. Li and P.M.B. Vitányi: *An Introduction to Kolmogorov Complexity and Its Applications.* Texts and Monographs in Computer Science. Springer-Verlag, 1993.

[Lov69] D.W. Loveland: "A variant of the Kolmogorov concept of complexity". *Information and Control* **15**, 510–526, 1969.