On the Autoreducibility of Random Sequences

Todd Ebert[1], Wolfgang Merkle[2], and Heribert Vollmer[3]

[1] California State University, Long Beach,
Department of Computer Engineering and Computer Science,
Bellflower Blvd, Long Beach, CA 90840, USA,
e-mail address: `ebert@cecs.csulb.edu`

[2] Ruprecht-Karls-Universität Heidelberg, Mathematisches Institut,
Im Neuenheimer Feld 294, D–69120 Heidelberg, Germany,
e-mail address: `merkle@math.uni-heidelberg.de`

[3] Universität Hannover, Theoretische Informatik,
Appelstraße 4, D–30167 Hannover, Germany,
e-mail address: `vollmer@informatik.uni-hannover.de`

Abstract. A binary sequence $A = A(0)A(1)\ldots$ is called i.o. Turing-autoreducible if A is reducible to itself via an oracle Turing machine that never queries its oracle at the current input, outputs either $A(x)$ or a don't-know symbol on any given input x, and outputs $A(x)$ for infinitely many x. If in addition the oracle Turing machine terminates on all inputs and oracles, A is called i.o. truth-table-autoreducible.
We obtain the somewhat counterintuitive result that every Martin-Löf random sequence, in fact even every rec-random or p-random sequence, is i.o. truth-table-autoreducible. Furthermore, we investigate the question of how dense the set of guessed bits can be when i.o. autoreducing a random sequence. We show that rec-random sequences are never i.o. truth-table-autoreducible such that the set of guessed bits has positive constant density in the limit, and that a similar assertion holds for Martin-Löf random sequences and i.o. Turing-autoreducibility. On the other hand, our main result asserts that for any rational-valued computable function r that goes non-ascendingly to zero, any rec-random sequence is i.o. truth-table-autoreducible such that on any prefix of length m at least a fraction of $r(m)$ of the m bits in the prefix are guessed.
We include a self-contained account of the hat problem, a puzzle that has received some attention outside of theoretical computer science. The hat problem asks for guessing bits of a finite sequence, thus illustrating the notion of i.o autoreducibility in a finite setting. The solution to the hat problem is then used as a module in the proofs of the positive results on i.o. autoreducibility.

1 Introduction

In probability theory one fundamental idea is the concept of independence. A collection of random variables $X_1, X_2, \ldots$ is independent if the information obtained from observing the outcome of the variables X_j where $j \neq i$ leaves the distribution of X_i unaffected, in that the a posteriori distribution of X_i equals its a priori distribution. Moreover, viewed from a computational standpoint this idea can be translated as saying that an algorithm whose goal on input i is to guess or estimate the outcome of X_i should not benefit from querying about the outcome of the X_j where $j \neq i$. For example, consider the chance experiment where the bits of an infinite binary sequence $R(0)R(1)R(2)\ldots$ are obtained by successive tosses of a fair coin. If we want to come up with a procedure that on input i computes the bit $R(i)$ while having access only to the remaining bits of R but not to $R(i)$, the *a posteriori* probability of guessing $R(i)$ given knowledge of $R(j)$ for all $j \neq i$ equals the *a priori* probability of guessing $R(i)$, thus the chance of success when guessing $R(i)$ cannot be better than $1/2$ and the probability that all bits of R are guessed correctly by the given rule is 0. In particular, it seems natural to regard the information obtained from observing $R(j)$ with $j \neq i$ as unhelpful for guessing $R(i)$. However, somewhat counterintuitively, we demonstrate in this paper that their are algorithms that with probability 1 are, infinitely often and without error, capable of guessing the outcome of $R(i)$ by querying an oracle about the outcomes of $R(j)$, $i \neq j$.

For the moment, say an effective procedure with limited access to R as described above *autoreduces* R in case the procedure computes $R(i)$ for all i, and the procedure *i.o. autoreduces* R in case the procedure computes $R(i)$ for infinitely many i, while for all other inputs the procedure eventually signals ignorance about the correct value.Then it appears that for any effective procedure it is impossible to autoreduce or even to i.o. autoreduce R because at first glance it would seem certain that in the limit, half of the membership guesses must be wrong. Indeed, by Corollary 28 below, for almost all sequences R (i.e., with probability 1) the sequence R cannot be autoreduced. However, and this comes as a slight surprise, almost all sequences R can be i.o. autoreduced according to Theorem 19.

But how can it be that almost all sequences can be i.o. autoreduced when the bits of these sequences, hence in particular all the bits guessed, are chosen independently of all the other bits? Recall that by the strong law of large numbers, with probability 1 the frequency $(R(0) + \ldots + R(n-1))/n$ of 1's in R converges to $1/2$. Furthermore, given a sequence of subsets of the natural numbers where the kth set has cardinality k, the Borel-Cantelli Lemma tells us that with probability 1, at most finitely many of these sets have an empty intersection with R [34]. This shows that independent random events considered collectively may possess certain properties with probability one. Thus we may assume that certain properties are present in almost all sequences, and for appropriate properties this can be exploited in order to compute certain bits of a sequence. In summary, the crux of the following investigation rests on determining degrees to which properties that are present in almost all sequences may be used to occasionally compute the outcome of a random variable by observing the outcomes of other random variables where the random variables are mutually independent.

Being almost convinced that random sequences might indeed be i.o. autoreducible, we might still wonder how we can overcome the obstacle that when guessing $R(i)$ for given places i, necessarily we err half the time. The key observation is that a procedure that i.o. autoreduces R can decide on its own whether to make a guess on a certain input i. Then, by assuming an appropriate property that is present in almost all sequences, for all such sequences an effective procedure may compute infinitely many bits, despite its querying limitations. Observe in this connection that for the autoreductions to be constructed in the sequel, for almost all sequences the places that are guessed form a set that is not computable. Furthermore, the set of guessed bits cannot have constant positive density in the set of all words.

In what follows, we use the known concept of autoreducibility by oracle Turing machines for capturing the idea of using a sequence A as oracle in order to compute A, yet not being able to query A about the bit to be computed. The concept of i.o. autoreducibility, where just infinitely many bits of A have to be computed, is modeled by means of oracle Turing machines that, in addition to 0 and 1, may also output a special don't-know symbol. Moreover, we will consider restricted concepts of autoreducibility and i.o. autoreducibility that correspond to various reducibilities considered in recursion theory and complexity theory. For example, we introduce i.o. tt-autoreducibility, which is defined similarly to the usual truth-table-reducibility from recursion theory.

Furthermore, we do not just show that almost all sequences can be i.o. autoreduced by effective reductions of truth-table type but, and this is more, the latter assertion holds for all Martin-Löf-random, rec-random, and even p-random sequences. The positive results on general i.o. autoreducibility are complemented by negative results on i.o. autoreducibility where for example a positive constant fraction of all bits has to be guessed correctly. These negative results exhibit interesting interactions between the type of random sequence considered and the type of autoreduction employed, intuitively speaking.

The aim and scope of this paper can then be summarized as follows. We try to contribute to the investigation of the question of which types of random sequences are i.o. autoreducible, at which density, and with respect to which types of reducibilities.

We conclude this section with an outline of the paper and an overview on its technical contributions. In Section 2 we state a puzzle, the *hat problem*, also known as *colored hat problem* or *prisoners problem*. By means of the hat problem we illustrate how techniques from coding theory can be applied when trying to autoreduce random sequences. More precisely, we review perfect one-error-correcting codes and show that these codes can be used to derive optimal solutions for certain instances of the hat problem. In subsequent sections, the solutions are then used as basic modules when constructing autoreductions. The hat problem was introduced by Ebert [17] in order to illustrate the problem of autoreducing random sequences and has recently become well known outside of theoretical computer science [31, 33]. In an attempt to provide a reference for the hat problem that is also accessible to readers that are not interested in applications to autoreducibility, we have tried to make Section 2 self-contained.

In Section 3 we review effective random sequences and related issues in effective measure theory, and in Section 4 we give formal definitions for the concepts of autoreducibility that are subsequently used.

In Section 5, we consider i.o. autoreducibility of random sequences. We prove that *every* rec-random sequence is i.o. truth-table-autoreducible and, what is more, in fact any p-random sequence is i.o. truth-table-autoreducible via an oracle Turing machine that runs in polynomial time. As mentioned above, this result seems somewhat surprising and even paradoxical in that the machine that witnesses the autoreducibility has the task of infinitely often guessing a bit of a random sequence, and guessing correctly each time despite a high chance of error for each guess. We then show that these results require Turing machines where the number of queries made for a single input is unbounded. This is accomplished by proving that no rec-random sequence is i.o. bounded truth-table-autoreducible, i.e., i.o. autoreducible by an oracle Turing machine that is restricted to some fixed number of queries, and an analogous result is shown in a setting of polynomial time-bounds.

In Section 6, we introduce the notion of autoreducibility with density $r(m)$ as a gauge of how often an oracle machine can guess the bits of a random sequence, or, in other words, how dense the set of guessed bits can be with respect to the entire set of bits. A sequence is i.o. autoreducible with density $r(m)$ if it is i.o. autoreducible such that for all m, at least a fraction of $r(m)$ of the first m bits of the sequence is guessed. In Theorem 26 it is shown that rec-random sequences are never i.o. truth-table-autoreducible with positive constant density (i.e., with density $r(m) = \varepsilon m$ for some $\varepsilon > 0$), and that a similar assertion holds with respect to Martin-Löf random sequences and i.o. Turing-autoreducibility. On the other hand, Theorem 29, our main result, asserts that for any computable function r that goes non-ascendingly to 0, any rec-random sequence is i.o. truth-table-autoreducible with density $r(m)$. So we obtain essentially matching bounds on the density of guessed bits for i.o. truth-table-autoreductions of rec-random sequences.

Related Work

The current article is a joint full version of conference articles by Ebert and Vollmer [19] and Ebert and Merkle [18]. The hat problem was originally introduced in the literature by Ebert [17] in order to illustrate the problem of autoreducing random sequences. The hat problem has led to work in coding theory [24] since there are instances of the problem for which the optimal solution (code) is not known. Moreover, the hat problem has become well known outside of theoretical computer science [31, 33]. Independently and considerably earlier, towards the end of the 1980s, Beigel et al [8] considered voting problems that have a flavor similar to the hat problem. Moreover, Rudich [32] points out that the hat problem is essentially the same as a variant of the voting problems called “voting with abstention” and he reports unpublished earlier work on the latter. The concept of i.o. autoreducibility has been investigated by Stephan et al. [9], who construct a sequence in exponential time that is not i.o. truth-table-autoreducible in polynomial time.

Notation

We use standard notation, which is elaborated further in the references [5, 10, 11, 27].

We consider words over the binary alphabet $\{0, 1\}$, which are ordered by the usual length-lexicographical ordering; the $(i + 1)$st word in this ordering is denoted by s_i, hence for example s_0 is the empty word λ. Occasionally, we identify words with natural numbers via the mapping $i \mapsto \mathrm{s}_i$.

If not explicitly stated differently, a sequence is an infinite binary sequence and a class is a set of sequences. A subset A of the natural numbers $\mathbb{N}$ is identified with its characteristic sequence $A(0)A(1)\ldots$, where $A(x)$ is 1 if and only if $x \in A$; notation defined for such subsets is extended to the corresponding sequences; e.g., an oracle Turing machine may reduce one sequence to another. The term class refers to a set of sequences.

An *assignment* is a (total) function from some subset of the natural numbers to $\{0, 1\}$. An assignment is *finite* iff its domain is finite. An assignment with domain $\{0, \ldots, n - 1\}$ is identified in the natural way with a word of length n. For an assignment σ with domain $\{z_0 < \ldots < z_{n-1}\}$, the *word associated with* σ is the (unique) word w of length n that satisfies $w(i) = \sigma(z_i)$ for $i = 0, \ldots, n-1$.

The restriction of an assignment σ to a set I is denoted by $\sigma|I$. In particular, for any sequence X, the assignment $X|I$ has domain I and agrees there with X. For a sequence X and an assignment σ, we write $\langle X, \sigma\rangle$ for the sequence that agrees with σ for all arguments in the domain of σ and agrees with X, otherwise.

The class of all sequences is referred to as *Cantor space* and is denoted by $\{0, 1\}^\infty$. The class of all sequences that have a word x as common prefix is called the *cylinder generated by* x and is denoted by $x\{0, 1\}^\infty$. For a set W, let $W\{0, 1\}^\infty$ be the union of all the cylinders $x\{0, 1\}^\infty$ where the word x is in W.

We write Prob[.] for probability measures and $\mathbf{E}$[.] for expected values. Unless stated otherwise, all probabilities refer to the *uniform measure* (or *Lebesgue measure*) on Cantor space, which is the probability distribution obtained by choosing the individual bits of a sequence by independent tosses of a fair coin. Usually we write Prob[A satisfies ...] instead of Prob[$\{A \in \{0, 1\}^\infty : A$ satisfies ...$\}$] in case it is understood from the context that the measure is uniform measure with respect to A.

Logarithms are to base 2. The function $\langle ., .\rangle$ from $\mathbb{N} \times \mathbb{N}$ to $\mathbb{N}$ is the usual effective and effectively invertible pairing function [37].

2 The hat problem and error-correcting codes

This section features the hat problem, which asks for guessing a single bit of a randomly chosen finite sequence from the remaining bits, a problem that resembles the task of i.o. autoreducing random infinite sequences. Subsequently, in constructions that solve the latter task, the hat problem and its solution are used as a basic module. The hat problem is formulated as a puzzle and as such has received some attention in the public [31,33].

> **The hat problem**
>
> In the hat problem for a team of n players, a binary sequence of n bits is chosen by independent tosses of a fair coin. Player i is assigned the ith bit (or, equivalently, is assigned one of two possible hat colors according to this bit).
> Afterwards, each player may cast a guess on its own bit (or may abstain from guessing) under the following conditions. The players know the bits of all other player but not their own bit. Once the game begins, the players are neither allowed to communicate nor do they know whether the other players have already guessed. However, the players can meet for a strategy session before the game begins.
> The team wins if and only if there is at least one correct and no incorrect guess.

At first sight, since each player may only observe events that are independent of his own hat color, one might expect that the team should have no more than a 50% chance of winning. However, since they converse before the game, we demonstrate how collaboration can increase their chances.

Example 1 *Consider the hat problem with $n = 3$ players and suppose the team agrees on the following guessing strategy. Upon start of the game, each player observes the hats of his two teammates. If both hats have the same color, then the player guesses his hat is colored differently. However, if the hats have different colors, then he passes.*

To compute the chances for a win under this strategy, we distinguish two cases. The first case occurs when all three hats have the same color, i.e., for two out of the 8 equiprobable assignments. In this case, each player will incorrectly guess. The second case is exactly two of the three hats are colored the same. Here exactly one player will venture a guess, this guess is correct and the team wins. Hence the probability of winning using the above strategy is $1 - 2/8 = 3/4$.

Next we want to extend the solution to the hat problem with three players given in Example 1 to other team sizes. For a given team size n, identify assignments of colors with words of length n in the natural way; i.e., the players are numbered from 1 to n and the jth bit of the word represents player j's hat color. The word that represents the actual assignment of colors is called the *true word*. With the true word understood, there are exactly two words of length n that agree with player j's view, the true word and the word that differs from the true word exactly at the jth position. Call these two words the *consistent words* of player j.

Specifying a strategy amounts to determining for any player and for any possible pair of consistent words for this player, whether the player should cast a vote and if yes, in favor of which of the two consistent words. This means that a strategy can be pictured as a directed graph G in the following way. The nodes of the graph are just the possible assignments, i.e., the words in $\{0,1\}^n$. The graph contains an edge from u to v if and only if these two nodes may occur as the consistent nodes of some player and in this situation the player votes in favor of v. Formally, we have $G = (V, E)$ where

$$V = \{0,1\}^n, \qquad E \subseteq \{(u,v) \mid u \text{ and } v \text{ differ exactly in one position } \} \quad ,$$

and for any pair u and v of nodes in V, at most one of the edges (u,v) and (v,u) is in E.

Consider any strategy and its associated graph, and assume that the strategy is applied in a situation where the true word is u. The team wins if according to the given strategy some player casts a vote in favor of u but no player guesses in favor of a word different from u. In terms of the associated graph, this means that the team wins on the assignment u if and only if

$$\text{some edge is pointing to } u \text{ and no edge is pointing away from } u. \tag{1}$$

From this characterization of winning assignments we obtain an equivalent formulation of the hat problem as a network problem, which is stated in Remark 2. Afterwards, we construct solutions to this network problem and translate them back to the hat problem. The point in considering the network problem is that the way its solutions work are more easily understood than for the hat problem.

Remark 2 *The hat problem can be reformulated as a problem on communication networks with a hypercube topology, where the nodes of the network correspond to the possible assignments of hat colors to the players.*

In order to obtain an equivalent version of the hat problem with n players, we consider a network of 2^n processors or nodes. The nodes are connected according to a hypercube topology [23]. That is, each node is labelled by a unique word of length n and between any two distinct nodes there is a link if and only if their labels differ in exactly one bit. The links are capable of transmitting information in either direction but only in one direction at a time. The task is to give a pattern of communication such that there is a maximum number of nodes u such that

$$\textit{some node is sending to } u \textit{ and } u \textit{ is not sending to any node.} \tag{2}$$

A pattern of communication specifies for each link either the status idle or an orientation, i.e., one of the two possible directions of sending.

Any pattern of communication translates naturally into a strategy for the hat problem with n players and vice versa. Furthermore, under this translation the fraction of nodes that satisfy (2) *coincides with the success probability of the* *strategy. For a proof of the two latter assertions it suffices to observe that the representation of a strategy as directed graph is essentially identical to a pattern of communication, and to compare the conditions* (1) *and* (2).

Example 3. The solution of the hat problem with $n = 3$ players from Example 1 translates as follows into a solution of the network problem. The network has a hypercube topology with 2^3 nodes and the pattern of communication specifies that

> exactly the nodes 000 and 111 are sending, and each of them sends to all its neighbors in the hypercube.

This way every processor u with label different from 000 and 111 is receiving but does not send and thus satisfies (2).

In order to extend the solution to the network problem given in Example 3 to larger networks, we review some notation and facts from coding theory. The *(Hamming) distance* $\mathrm{d}(u,v)$ of two words u and v of identical length is the number of positions at which u and v differ. Furthermore, the *unit sphere* with *center* u is the set of all words of the same length that differ from u at most at one position, i.e., the set

$$\{v : \mathrm{d}(u,v) \leq 1\}\,.$$

The *surface* of a unit sphere is just the sphere with its center removed.

In the network problem, the set of neighbors of a node w coincides with the surface of the unit sphere centered at w. Accordingly, the solution to the network problem from Example 3 can be reformulated as follows.

> Exactly the nodes 000 and 111 are sending, and each of them sends to all nodes on the surface of the unit sphere centered at this node.

The easy idea underlying this solution works also for networks where the parameter n is larger than 3. Just select a subset C of all words of length n and let each node (labelled by a word) in C send to all nodes that are on the surface of the unit sphere centered at this node but are not in C themselves. If, for example, we choose the set C such that the unit spheres around the words in C in are disjoint, then exactly the nodes on the surfaces of these unit spheres satisfy (2); i.e., these are the nodes that receive but do not send. The fraction of such nodes becomes maximum among all corresponding choices of C in case the unit spheres around the words in C partition the set of all words of the given length. Such partitions are studied in coding theory under the name of perfect one-error-correcting codes.

Definition 4. *Any subset of* $\{0,1\}^n$ *is called a* code *with* (codeword) length n. *A code C with length n is called* perfect one-error-correcting *if the unit spheres around the codewords in C form a partition of* $\{0,1\}^n$ *(i.e., if for any word w of length n there is exactly one word $c \in C$ such that* $\mathrm{d}(w,c) \leq 1$*).*

For any perfect one-error-correcting code of length n, the number of all words 2^n must be divisible by the unit sphere volume $n+1$, hence $n+1$ must be a power of 2. It is well known that this necessary condition on the codeword length is also sufficient [20, 40].

Fact 5 *Let n be of the form $2^k - 1$ where $k > 0$ is a natural number. Then there is a perfect one-error-correcting code C_n of codeword length n. Furthermore, the codes C_n can be chosen such that their union $\bigcup_{\{n:n=2^k-1\}} C_n$ is decidable in polynomial time. For example, such codes are given by the well-known family of binary Hamming codes [40].*

Proof. We identify words and binary vectors in the obvious way, hence the words of any given length form a vector space under addition modulo 2 and over the field with two elements. Let M be the $k \times n$ binary matrix where for $i = 1, \ldots, n$, column i of M is the binary representation of the number i. The matrix M defines a linear mapping $w \mapsto Mw$ from words of length n to words of length k. Let C_n denote the kernel space of this mapping, i.e., the set of all words that are mapped to 0^k. Then C_n is a perfect one-error-correcting code of length n.

For a proof, first observe that trivially every word in the kernel has length n. Second, any word w of length n is contained in a unit sphere centered at a word in C_n. In case Mw is zero, this is obvious. Otherwise, Mw appears as a column of M, say, as column j, hence flipping the jth bit of w results in a word in C_n. Third, the unit spheres centered at the words in C_n are mutually disjoint. Fix any two distinct words u and v in C_n. Then M maps both of u and v to 0^k and, by linearity of matrix multiplication, the same holds for $u - v$. On the other hand, $M(u-v)$ is just the sum over all column vectors j of M such that u and v differ at the jth position. Now the sum over one or over two distinct column vectors of M cannot be equal to 0^k, hence $\mathrm{d}(u, v) \geq 3$ and in particular the unit spheres centered at u and v must be disjoint. (The third item is a special form of a well-known result in coding theory that the minimum distance of a code which is the kernel space of some matrix M is equal to d, where d is the least number for which there are d linearly dependent column vectors of M. In our case it is easy to check that M has 3 linearly dependent vectors, but not 2. Thus C_n has a minimum distance of 3.)

Finally, the problem of deciding if a word belongs to $\bigcup_{\{n:n=2^k-1\}} C_n$ can be solved in polynomial-time, as essentially it only involves multiplying the input with a matrix that has moderate size and is easy to compute. □

Remark 6 summarizes the application of error-correcting codes to the hat problem and the related network problem.

Remark 6 *Let n be of the form $2^k - 1$ for some natural number $k > 0$ and consider the hat problem and the network problem with parameter n. Fix a perfect one-error-correcting code of code word length n and call the words in this code, as well as the nodes labelled by these words, designated.*

A solution to the network problem is given by the following pattern of communication. By Remark 2, under this pattern of communication exactly the nodes that are not *designated satisfy* (2).

Every designated node sends to all its neighbors in the hypercube; the other nodes do not send.
(That is, every designated node w sends to all nodes on the surface of the unit sphere centered at w.)

A solution to the hat problem with n players is given by the strategy where each player behaves according to the following rule. By Remark 2, under this strategy the team wins exactly for the assignments that are not *designated.*

In case one of the consistent words is a designated word, venture a guess according to the assumption that the true word is the other consistent word; otherwise, pass.
(That is, in case the consistent words are a designated word w and a word on the surface of the sphere centered at w, guess in favor of the consistent word on the surface, i.e., the one different from w.)

The fraction of designated words is $1/(n+1)$ because the spheres centered at the designated words partition $\{0,1\}^n$ and each such sphere consists of one designated and n other words. Hence if this strategy is applied in the hat problem with n players, the team wins with probability $1 - 1/(n+1)$.

Example 7. Consider the hat problem with $n = 7$ players. For the strategy described in Remark 6, the codewords comprise a fraction of $1/8$ of the 128 words in $\{0,1\}^7$, hence there are 16 codewords and 112 error words and the team wins with probability $112/128 = 0.875$.

In the network problem with parameter n, a node can send at most to n other nodes, hence among all nodes that send or receive, at least a fraction of $1/(n+1)$ nodes must send. The pattern of communication described in Remark 6 achieves this bound and thus is an optimum solution to the network problem, hence by the discussion in Remark 2 also the corresponding strategy for the hat problem is optimum. Remark 8 contains an alternate, somewhat more formal proof for the optimality of this strategy, which features the idea that for any strategy the expected number of correct and incorrect guesses is the same.

Remark 8 *Let n be of the form $n = 2^k - 1$. For the hat problem with n players, the probability of success of $1-1/(n+1)$ that is achieved by the strategy described in Remark 6 is optimum.*

For a proof, fix any strategy. Recall that the colors of the hats are assigned according to independent tosses of a fair coin, hence whenever the strategy tells a player to guess, the probability of a correct guess is exactly $1/2$. As a consequence, the expected number of correct and incorrect guesses per player is the same, and by linearity of expectation, the same holds for the entire team; i.e., if we define the random variables g_c and g_i as equal to the number of correct and incorrect guesses respectively for the entire team, their expected values $\mathbf{E}[g_c]$ and $\mathbf{E}[g_i]$ coincide. Furthermore, if the strategy considered has probability of success of p, then we have

$$p \leq \mathbf{E}[g_c] = \mathbf{E}[g_i] \leq (1-p)n. \tag{3}$$

In (3), the middle equation holds by the preceding discussion, the left-hand inequality follows because for each assignment that leads to a win there must be at least one correct guess, and the right-hand inequality holds because for any winning assignment there is no wrong guess, while for any other assignments there are at most n wrong guesses. So we have $p \leq (1-p)n$, and by rearranging we obtain $p \leq 1 - 1/(n+1)$.

Lenstra and Seroussi [24] discuss applications of coding theory to the hat problem. They investigate into good strategies for numbers of players that are not of the form $2^k - 1$ and for more general versions of the hat problem with more than two colors. For the hat problem with two colors, they show that strategies are equivalent to covering codes. Their observation is reviewed in Remark 9, where we give an equivalent formulation in terms of communication patterns for the network problem.

Remark 9 *A code of length n is called a covering code (more precisely, a 1-covering code) if any word of length n differs at most at one position from some word in the code. For example, perfect one-error-correcting codes are covering codes, however, in general the unit spheres centered at the words in a covering code are not mutually disjoint.*

Given a pattern of communication for the network problem with parameter n, let C be the set of all nodes that do not satisfy (2); i.e., C contains the nodes that are sending or a neither sending nor receiving, and the complement of C contains the nodes that receive but do not send. Then C is a covering code because the nodes not in C are receiving, hence each such node must be at distance at most 1 from a sending node, which then must be a node in C. Conversely, given any covering code of length n, there is a pattern of communication where every node in C sends to all its neighbors that are not in C themselves. With this pattern, exactly the nodes that are not in C satisfy (2).

3 Random sequences

This section gives a brief introduction to the theory of effective measure. We focus on effective random sequences and related concepts that are used in the following. For more comprehensive accounts of effective measure we refer to the references [4, 5, 27].

Imagine a casino that offers roulette and consider the sequence of outcomes red and black that occur in the course of the game. We would certainly not call this sequence random if there were a method to determine any next bit before the corresponding drawing has actually taken place. But also if we just knew a strategy that guarantees winning an unbounded amount of money when starting with finite initial capital, this would indicate that the sequence is non-random. So we might be tempted to call a sequence non-random if there is such a strategy. The problem with this definition is that for any sequence there is a strategy that wins against this sequence, e.g., the one that works by always predicting correctly the next bit of the sequence. However, the latter is not a problem for real casinos because for them a sequence is "random enough" if it does not permit a winning strategy that a gambler can actually play. In general, this suggests to define randomness relative to a certain class of admissible betting strategies instead of striving for an absolute concept. In what follows, the admissible betting strategies are just the ones that are computable in a specific model of computation. A sequence is called random with respect to such a model of computation if none of the admissible betting strategies leads to an unbounded gain when playing against this sequence.

In order to formalize the ideas of the preceding paragraph, consider the following gamble. Imagine a player that successively places bets on the individual bits of an unknown sequence A. The betting proceeds in rounds $i = 1, 2, \ldots$. During round i, the player receives as input the length $i-1$ prefix of A and then, first, decides whether to bet on the i th bit being 0 or 1 and, second, determines the stake that shall be bet. The stake might be any fraction between 0 and 1 of the capital accumulated so far; i.e., in particular, the player is not allowed to incur debts. Formally, a player can be identified with a *betting strategy*

$$b \colon \{0,1\}^* \to [-1, 1]$$

where on input w the absolute value of $b(w)$ is the fraction of the current capital that shall be at stake and the the bet is placed on the next bit being 0 or 1 depending on whether $b(w)$ is negative or non-negative.

The player starts with positive, finite capital. At the end of each round, in case of a correct guess, the capital is increased by that round's stake and, otherwise, is decreased by the same amount. So given a betting strategy b, we can inductively compute the corresponding *payoff function* d_b by applying the equations

$$d_b(w0) = d_b(w) - b(w) \cdot d_b(w), \qquad\qquad d_b(w1) = d_b(w) + b(w) \cdot d_b(w).$$

Intuitively speaking, the payoff $d_b(w)$ is the capital the player accumulates till the end of round $|w|$ by betting on a sequence that has the word w as a prefix. The payoff function d_b satisfies the fairness condition

$$d_b(w) \;=\; \frac{d_b(w0) + d_b(w1)}{2}\;. \tag{4}$$

We call a function d from words to nonnegative reals a *martingale* iff $d(\lambda) > 0$ and d satisfies the fairness condition (4), with d_b replaced by d, for all words w. By the discussion above, for a betting strategy b the function d_b is always a martingale and, conversely, it can be shown that every martingale has the form d_b for some betting strategy b. Hence betting strategies and martingales are essentially equivalent. Accordingly, we extend occasionally notation defined for betting strategies to martingales and vice versa.

Definition 10. *A betting strategy b* succeeds *on a sequence A if the corresponding martingale d_b is unbounded on the prefixes of A; i.e., if*

$$\limsup_{n \in \omega} \; d_b(A|\{0, \ldots, n\}) \;=\; \infty.$$

In what follows, we will consider computable betting strategies. Any computable betting strategy b is confined to rational values, and there is a Turing machine that on input w outputs some appropriate finite representation of $b(w)$.

Computable betting strategies are not only of interest in connection with the definition of random sequences, but are also the basis of the theory of effective measure. A betting strategy is said to *succeed on* or to *cover* a class iff it succeeds on every sequence in the class. Ville demonstrated that a class has uniform measure 0 iff the class can be covered by some, not necessarily effective, betting strategy [5, 42]. This result justifies the following notation. A class has *measure* 0 with respect to a given class of betting strategies iff it is covered by some betting strategy in the class. By appropriately restricting the class of admissible betting strategies, one obtains restricted concepts of measure 0 classes, which come in handy when investigating classes occurring in recursion theory or complexity theory. Most of these classes are countable and hence have uniform measure 0, i.e., from the point of view of uniform measure all these classes have the same size. However, given a specific class $\boldsymbol{C}$, we might try to restrict the class of admissible betting strategies such that the resulting concept of measure 0 class is interesting in the sense that we can still cover relevant subclasses of $\boldsymbol{C}$, but not the class $\boldsymbol{C}$ itself. In the context of recursion theory, this led to the consideration of computable betting strategies [4, 35, 36, 38, 43]. In connection with complexity classes one imposes additional resource-bounds [5, 26, 27, 29], e.g., in the case of the class $\mathbf{E}$ of sequences that can be computed in deterministic line time 2^{lin}, Lutz proposed to use betting strategies that are computable in polynomial time.

Remark 11 *The resources needed to compute a betting strategy are measured with respect to the length of the input w, for example, a betting strategy b is computable in polynomial time if $b(w)$ can be computed in time $|w|^c$ for some constant c.*

A prefix w of a sequence A encodes $A(\mathrm{s}_0)$ through $A(\mathrm{s}_{|w|-1})$ and accordingly on input w, a betting strategy determines a bet on whether $x = \mathrm{s}_{|w|}$ is in the unknown sequence or not. Observe that the length of x is approximately $\log |w|$, thus for example a time bound $|w|^c$ translates to a time bound of the form $2^{\mathrm{O}(|x|)}$.

After this short digression to effective measure theory we return to the endeavor of defining concepts of random sequences via restricted classes of betting strategies.

Definition 12. *A sequence is* rec-random *if no computable betting strategy succeeds on it. A sequence is* p-random *if no betting strategy that is computable in polynomial time succeeds on this sequence.*

Besides p-random and rec-random sequences, we will consider Martin-Löf-random sequences [28]. Let $W_0, W_1, \ldots$ be the standard enumeration of computationally enumerable sequences [37].

Definition 13. *A class $\mathcal{N}$ is called a* Martin-Löf null class *iff there exists a computable function $g \colon \mathbb{N} \to \mathbb{N}$ such that for all i*

$$\mathcal{N} \subseteq W_{g(i)}\{0,1\}^\infty \quad \textit{and} \quad \mathrm{Prob}[W_{g(i)}\{0,1\}^\infty] < \frac{1}{2^i}\,.$$

For such a function g, the sequence $W_{g(0)}, W_{g(1)}, \ldots$ is called a Martin-Löf null cover *for $\mathcal{N}$. A sequence is* Martin-Löf-random *if it is not contained in any Martin-Löf null class.*

Martin-Löf-random sequences have been characterized in terms of martingales by Schnorr [36]. A sequence is *Martin-Löf-random* if and only if it cannot be covered by a subcomputable martingale. A martingale d is *subcomputable* iff there is a computable function $\widetilde{d}$ in two arguments such that for all words w, the sequence $\widetilde{d}(w,0), \widetilde{d}(w,1), \ldots$ is nondecreasing and converges to $d(w)$.

Remark 14 *For any sequence X we have*

$$X \textit{ Martin-Löf-random } \Rightarrow X \textit{ rec-random } \Rightarrow X \textit{ p-random} \tag{5}$$

and both implications are strict.

The first implication in (5) is immediate by the characterization of Martin-Löf-random sequences in terms of subcomputable martingales and the observation that for a computable betting strategy the corresponding martingale is computable, too. Likewise, the second implication follows from the definitions of rec-random and p-random sequences in terms of computable and polynomial-time computable betting strategies. Furthermore, the second implication is strict because one can construct a computable p-random sequence by diagonalizing against an appropriate weighted sum of all betting strategies that are computable in polynomial time. The strictness of the first implication was implicitly shown by Schnorr [36]. For a proof, it suffices to recall that the prefixes of a Martin-Löf-random sequence cannot be compressed by more than a constant while a corresponding statement for rec-random sequences is false [25, Theorem 3.6.1 and Exercise 2.5.13].

By definition, a class $\mathcal{N}$ has uniform measure 0 if there is a, not necessarily computable, function g as in Definition 13. Thus the concept of a Martin-Löf null class is indeed an effective variant of the classical concept of a class that has uniform measure 0. In particular, any Martin-Löf null class has uniform measure 0. By σ-additivity and since there are only countably many computable functions, the union of all Martin-Löf null classes has uniform measure 0. Accordingly, the class of Martin-Löf random sequences, and hence by Remark 14 also the classes of rec-random and of p-random sequences, have uniform measure 1. We note in passing that it can be shown that the union of all Martin-Löf null classes is again a Martin-Löf null class [15, Section 6.2].

We conclude this section by describing a standard technique for the construction of betting strategies.

Remark 15 *Let I be a finite set and let Θ be a subset of all partial characteristic functions with domain I. Then there is a betting strategy that, by betting on places in I, increases its capital by a factor of $2^{|I|}/|\Theta|$ for all sequences B where the restriction of B to I is in Θ.*

The betting strategy is best described in terms of the corresponding martingale. The martingale takes the capital available when betting on the least element of I and distributes it evenly among the elements of Θ, then computing values upwards according to the fairness condition for martingales.

4 Autoreducibility

For the moment, call a sequence X *autoreducible* if there is an effective procedure that on input x computes $X(x)$ while having access to the values $X(y)$ for $y \neq x$. Intuitively speaking, for an autoreducible sequence the information on $X(x)$ is not only stored at x but can also be effectively recovered from the remainder of the sequence. For example, any computable sequence is autoreducible and for an arbitrary sequence Y, the sequence $Y(0)Y(0)Y(1)Y(1)\dots$ is autoreducible. In a recursion theoretic setting, the concept of autoreducibility was introduced by Trakhtenbrot [39]; further investigations showed, among other results, that autoreducibility is tightly connected to the concepts of mitoticity and introreducibility [7, 16, 21, 22]. The concept of autoreducibility was transferred to complexity theory by Ambos-Spies [1] and subsequently resource-bounded versions of autoreducibility have been studied by several authors [9, 13, 14, 41]. Selfreducibility is a special form of autoreducibility where only queries less than the current input may be asked; it can be shown that certain forms of selfreducibility characterize certain types of generic sets [2, 3, 12].

Now consider the question of whether a random sequence can be autoreducible. By definition, the bits of an autoreducible sequence depend on each other in an effective way. This suggests that by exploiting the dependencies, we might come up with an effective betting strategy that succeeds on this sequence. Indeed Martin-Löf-random sequences are never autoreducible and, similarly, rec-random sequences are not autoreducible by reductions that are confined to non-adaptive queries, see Corollary 28 below.

Pushing the issue further, we might ask whether for a random sequence R it is at least possible to recover some of the values $R(x)$ from the values $R(y)$ with $y \neq x$. Trivially, this is possible for finitely many places x, so let's consider the case of infinitely many x. For the moment, call a sequence X *i.o. autoreducible* if there is an effective procedure that for infinitely many inputs x computes $X(x)$ while having access to the values $X(y)$ for $y \neq x$, whereas for all other inputs the procedures eventually signals that it cannot compute the correct value. For example, any sequence that, if viewed as a set, has an infinite computable subset is i.o. autoreducible, hence in particular any sequence that corresponds to an infinite computably enumerable set is i.o. autoreducible. Observe that by a standard diagonalization argument of finite extension type, one can easily construct a sequence that is computable in the halting problem and is not i.o. autoreducible.

For an i.o. autoreducible sequence R there are infinitely many places x where the value of $R(x)$ depends in an effective way on the remainder of the sequence R. On first sight, the situation looks rather similar to the case of an autoreducible sequence and indeed it is tempting to assume that random sequences cannot be i.o. autoreducible. So the following result is somewhat surprising. *Every* p-random sequence is i.o. autoreducible by a reduction procedure that runs in polynomial time. This and related results are demonstrated in Section 5. In the remainder of this section, we give formal definitions for various concepts of autoreducibility.

Recall the concept of an *oracle Turing machine* [10], which is a Turing machine that during its computation has access to a sequence X, the oracle. In case an oracle Turing machine M eventually terminates on input x and with oracle X, let $M(X,x)$ denote the computed value and, otherwise, i.e., if M does not terminate, say that $M(X,x)$ is undefined. But rather than use standard oracle machines M whose defined outputs $M(X,x)$ belong to $\{0,1\}$, we also allow the machines to output a special "don't-know-symbol" $\perp$, which has the intended meaning of signaling that the correct value is not known.

Definition 16. *Let M be an oracle Turing machine (with output in $\{0,1,\perp\}$) and let A, B, and E be sequences. Then M* reduces A to B on E *if and only if*

$$(i)\quad M(B,x) = A(x) \text{ for all } x \text{ where } M(B,x) \neq \perp, \text{ and}$$
$$(ii)\quad M(B,x) \neq \perp \text{ for all } x \in E.$$

If an oracle Turing machine M reduces a sequence A to a sequence B on an infinite set, we say that M *infinitely often reduces* or, for short, *i.o. reduces* A to B. If M reduces A to B on the set of all words, we say that M *reduces* A to B. Obviously, the latter notion coincides with the usual concept of reduction by a $\{0,1\}$-valued oracle Turing machine, where it is required that $M(B,x)$ agrees with $A(x)$ for all x.

Definition 17. *Let M be an oracle Turing machine and let A and E be sequences. The set of query words occurring during the computation of M on input x and with oracle A is denoted by $\mathrm{Q}(M,A,x)$.*

The sequence A is autoreduced on set E by M *if M reduces the sequence A to itself on E and $x \notin \mathrm{Q}(M,A,x)$ for all x. The sequence A is* i.o. autoreduced by M *if A is autoreduced by M on an infinite set. The sequence A is* autoreduced by M *if A is autoreduced by M on the set of all words.*

Next we define concepts of autoreducibility that correspond to the standard effective reducibilities considered in recursion theory [37] and to the standard reducibilities computable in polynomial time considered in complexity theory [10]. More precisely, for

$$r \in \{\mathrm{T}, \mathrm{tt}, \mathrm{btt}, \mathrm{btt}(k), \text{p-T}, \text{p-tt}, \text{p-btt}, \text{p-btt}(k)\},$$

we define the concepts of r-autoreducibility on a set E, of i.o. r-autoreducibility, and of r-autoreducibility.

For a start, we consider Turing- or, for short, T-autoreducibility. A sequence is *T-autoreducible on E* if it can be autoreduced on E by some oracle Turing machine M. A sequence is *i.o. T-autoreducible* if it is T-autoreducible on an infinite set and a sequence is *T-autoreducible* if it is T-autoreducible on the set of all words.

The definitions for the remaining cases are basically the same, however the oracle Turing machine M that performs the autoreduction has in addition to satisfy certain requirements. In particular, in the following M must always be total; i.e., on all inputs and for all oracles, M must eventually finish its computation. In the case of *truth-table-autoreducibility* (tt), M has to ask its queries non-adaptively; i.e., M computes a list of queries that are asked simultaneously and, after receiving the answers, M is not allowed to access the oracle again. In the case of *bounded truth-table-autoreducibility* (btt), the queries have to be asked non-adaptively while the number of queries that might be asked on a single input is bounded by a constant. Even more restrictive, in the case of *btt(k)-autoreducibility* the number of non-adaptive queries is bounded by the fixed constant k. The concepts of *polynomial time-bounded autoreducibility* like p-T- or p-tt-autoreducibility are defined accordingly where it is required in addition that M runs in polynomial time.

We conclude this section by some technical remarks on the representation of oracle Turing machines. By definition, tt-autoreductions are performed by total oracle Turing machines that query the oracle non-adaptively. Such an oracle Turing machine can be conveniently represented by a pair of computable functions g and h where $g(x)$ gives the set of words queried on input x and $h(x)$ specifies how the answers to the queries in the set $g(x)$ are evaluated; i.e., $h(x)$ tabulates a $\{0, 1, \perp\}$-valued function over k variables. In this situation, we refer to $h(x)$ as truth-table. Likewise, oracle Turing machines that witness a p-btt-autoreduction can be represented by pairs of functions g and h that are computable in polynomial time.

Remark 18. Alternative to the $\{0, 1, \perp\}$-valued oracle Turing machine model, one could use the standard model, in that rather than output “don't-know”, the Turing machine would simply query the oracle about the value of x, and output that value. This formulation was originally used by Ebert [17]; a similar model is used by Arslanov [7], with different notation and in the special case of a weak truth-table-reduction that infinitely often queries the oracle only at places strictly smaller than the current input.

We emphasize that the results of this paper hold for both, the $\{0, 1, \perp\}$-valued and the standard model; the main motivation for adopting the former is that it complies better with the usual classification of reducibilities according to whether they query the oracle adaptively or non-adaptively, i.e., the definitions of the various concepts of i.o. autoreducibility of truth-table type preserve the idea of accessing the oracle only once.

5 Autoreductions of Random Sequences

Next we apply the solution of the hat problem as discussed in Section 2 to the construction of i.o. autoreductions of rec-random sequences.

Theorem 19. *Every rec-random sequence is i.o. tt-autoreducible.*

Proof. Fix any rec-random sequence R. Partition the natural numbers into consecutive intervals $I_1, I_2, \ldots$ where I_k has size $l_k = 2^k - 1$. Write R in the form

$$R = w_1 w_2 \ldots \qquad \text{where for all } k, \quad |w_k| = l_k \ ,$$

i.e., w_k is the word associated with the restriction of R to I_k. Furthermore, for every $k > 0$ fix a perfect one-error-correcting code C_k of codeword length l_k such that given x, we can decide in polynomial time whether x is in one of the codes C_k.

In a nutshell, the proof of Theorem 19 works as follows. The code words in C_k comprise such a small fraction of all words of length l_k that in case infinitely many words w_k were in C_k there would be a computable betting strategy that succeeds on R. But R is assumed to be rec-random, hence w_k is not in C_k for almost all k. Then in order to construct an oracle Turing machine that witnesses that R is i.o. tt-autoreducible, we handle the intervals I_k individually and when working on I_k, we simulate the solution of the hat problem with l_k players. This way we can compute $R(x)$ for a single place x in I_k whenever w_k is not in C_k. But the latter is the case for almost all k, thus we are able to autoreduce R as required. Details follow.

Claim 1 *For almost all k, w_k is not in C_k.*

Consider the following betting strategy. For every $k > 0$, a portion $a_k = 1/2^k$ of the initial capital 1 is exclusively used for bets on words in the interval I_k. On each interval I_k, the betting strategy follows the strategy described in Remark 15 where C_k plays the role of Θ; i.e., the capital a_k is bet on the event that (the word associated to) the restriction of the unknown sequence to I_k is in C_k. By construction, just a fraction of $a_k = 1/(l_k + 1)$ of all words of length l_k belongs to C_k, hence the capital a_k increases to 1 for all k such that the restriction of the unknown sequence to I_k is in C_k. As a consequence, the betting strategy succeeds on any sequence such that for infinitely many k, the restriction of the sequence to the interval I_k is an element of C_k. But no computable betting strategy can succeed on the rec-random sequence R, hence Claim 1 follows. □

Observe that the proof of Claim 1 depends on the choice of the l_k only in so far that as it is required that the sum over the a_k, where $a_k = 1/(l_k + 1)$, converges.

Next we define an oracle Turing machine M that witnesses that R is i.o. tt-autoreducible. By Claim 1, fix k_0 such that w_k is not in C_k for all $k > k_0$. On inputs in the intervals I_1 through I_{k_0}, M simply outputs $\perp$. On any input x in an interval I_k with $k > k_0$, M queries the oracle non-adaptively on all words in I_k that are different from x. Thereby M determines two words u' and u'', and knows that w_k is equal to one of the words

$$v_0 = u'0u'' \quad \text{and} \quad v_1 = u'1u'' ,$$

where the uncertainty is with respect to the value of $R(x)$. Then M outputs i in case $v_{1-i} \in C_k$ and $v_i \notin C_k$ and, otherwise, i.e., if neither v_0 nor v_1 is in C_k, M outputs $\perp$.

By construction, M is computable, queries its oracle non-adaptively, and does never query its oracle at the input. On intervals I_k with $k \leq k_0$, M always outputs $\perp$. On any interval I_k with $k > k_0$, M simulates the strategy for the hat problem with l_k players from Remark 6 where C_k is the set of designated words.

For any such interval, w_k is not in C_k by choice of k_0, hence the discussion in Section 2 shows that M computes the correct value $R(x)$ at the single input x in the interval at which w_k differs from the closest code word in C_k, while M outputs $\perp$ for all other inputs in the interval. In summary, M i.o. tt-autoreduces R. □

Subsequently, the statement of Theorem 19 will be strengthened in various ways. As an immediate improvement, we note that the proof of Theorem 19 can be adjusted in order to obtain the following stronger but also more technical version of the theorem. Given a computable function t, we call a sequence $t(m)$-*random* if no betting strategy that is computable in time $\mathrm{O}(t(m))$ succeeds on this sequence, where m denotes the length of the prefix of the unknown set that a betting strategy receives as input.

Theorem 20. *Let b be an unbounded and non-decreasing computable function. Every m^2-random sequence is i.o. tt-autoreducible by an oracle Turing machine M that on inputs of length n runs in time $\mathrm{O}(n)$ and asks at most $b(n)$ queries.*

Proof. The proof of Theorem 20 is rather similar to the proof of Theorem 19. We just indicate the necessary adjustments. Recall that the i.o. tt-autoreductions, as well as the betting strategy in the proof of Theorem 19 are built up from modules that work essentially independently on the intervals I_j. The key trick in the proof of Theorem 20 is to shift the intervals such that the length of the words contained in them are considerably larger than the length of the corresponding interval. More precisely, the interval I_k contains the first l_k words of length n_k, where the sequence $n_0, n_1, \dots$ is chosen such that we have for all k,

$$(i)\ \ 2^{n_k} < n_{k+1}\ , \qquad (ii)\ \ 2^{l_k} < n_k\ , \qquad (iii)\ \ l_k < b(n_k).$$

In addition, we assume that there is a Turing machine that on input 1^n uses at most n steps to decide whether n appears in the sequence $n_0, n_1, \dots$ and if so, to compute the index k with $n = n_k$. Such a sequence can be obtained by standard methods as described in the chapter on uniform diagonalization and gap languages in Balcázar et al. [10]. For example, we can first define a sufficiently fast growing time-constructible function $r : \omega \to \omega$ and then let n_i be the value of the i-fold iteration of r applied to 0.

Similar to Claim 1 in the proof of Theorem 19, we can argue that for any m-random sequence R and for almost all k, the restriction w_k of R to the interval I_k is not in the code C_k. Otherwise, there were a betting strategy similar to the one used in the proof of the mentioned claim that wins on R; i.e., on any interval I_k the strategy bets a fraction of $1/2^k$ of its initial capital on the event that w_k is in C_n. By choice of the n_k, this strategy can be chosen to run in time m^2. Recall in connection with this time bound that the individual bets are specified as a fraction of the current capital, hence placing the bets related to interval I_k requires to compute the outcomes of the bets on the previous intervals.

The sequence R is then i.o. tt-autoreducible by a reduction that simulates the solution to the hat problem on every interval I_k where w_k is not in C_k. On an input of length n, this reduction runs in time n and asks at most $b(n)$ queries because of (ii) and (iii), respectively. □

For a proof of the following corollary it suffices to observe that p-random sequences are in particular m^2-random, while every m^2-random sequence in turn is i.o. tt-autoreducible in polynomial time according to Theorem 20.

Corollary 21. *Every p-random sequence is i.o. p-tt-autoreducible.*

The next theorem shows that neither Theorems 19 and 20 nor Corollary 21 extend to i.o. btt-autoreducibility; i.e., the proofs of these results require oracle Turing machines that ask an unbounded number of queries.

Theorem 22. *(a) No rec-random sequence is i.o. btt-autoreducible.*
(b) No p-random sequence is i.o. p-btt-autoreducible.

Proof. (a) Fix any sequence A that is i.o. btt-autoreducible. It suffices to show that A is not rec-random, i.e., that there is a computable betting strategy b that succeeds on A.

Among all oracle Turing machines $M = (g, h)$ that i.o. btt-autoreduce A, we will consider only the ones that are *normalized* in the sense that the set of queries $g(x)$ is empty whenever the truth-table $h(x)$ is constant (i.e., whenever $h(x)$ evaluates to the same value for all assignments on $g(x)$). Furthermore, among all normalized oracle Turing machines that i.o. btt-autoreduce A, let $M = (g, h)$ be one such that its norm

$$l = \sup_{x \in \{0,1\}^*} |g(x)|$$

is minimum. In case $l = 0$, M is independent of the oracle and we can compute $A(x)$ for the infinitely many places x where the reduction does not yield $\perp$, hence A is not rec-random and we are done. So assume $l > 0$.

For any word x, there is a word $r(x) > x$ such that $g(r(x))$ has size l and contains only words strictly larger than x. Assuming otherwise, by hard-wiring $A(z)$ into M for all $z \leq x$ and for all z that are contained in one of the sets $g(y)$ with $y \leq x$, we would obtain an oracle Turing machine that again btt-autoreduces A and has norm strictly smaller than M, thus contradicting the choice of M. Let x_1 be the least word x such that $g(x)$ has size l and for $s > 1$, let

$$x_{s+1} = r(\max J_s) \quad \text{where} \quad J_s = \{x_s\} \cup g(x_s) \ .$$

By choice of r, this inductive definition yields an infinite sequence $x_1, x_2, \ldots$ such that the function $s \mapsto x_s$ is computable, the sets J_s all have size $l + 1$, and any element of J_s is less than any element in J_{s+1}.

Consider an arbitrary index s. Then $h(x_s)$ is not constant because $g(x_s)$ has size $l > 0$ and M is normalized. Thus there is an assignment on $g(x_s)$ such that $h(x_s)$ evaluates to a value different from $\perp$. Let α be the least such assignment and let i_s be the value obtained by applying the truth-table $h(x_s)$

to α. Extend α to an assignment α_s on the set J_s where $\alpha_s(x_s) = 1 - i_s$. Observe that J_s and α_s can be computed from s.

We construct now a computable betting strategy b that succeeds on A. The construction is based on the observation that for all s, the restriction of A to J_s differs from α_s. Otherwise, $M(A, x_s) = i_s$ would differ from $A(j_s) = 1 - i_s$ and M would not autoreduce A.

The betting strategy b can be viewed as working in stages $s = 1, 2, \ldots$. The bets of stage s are all on places in J_s and use the capital accumulated till the beginning of stage s for betting against the event that the restriction of the unknown sequence to J_s is equal to α_s. Formally, the bets at stage s are performed according to the strategy described in Remark 15 where $\Theta = \Theta_s$ contains all assignments on J_s that differ from α_s. The size of J_s is $l + 1$, hence there are 2^{l+1} assignments to J_s and Θ_s contains a fraction of

$$\delta = \frac{2^{l+1} - 1}{2^{l+1}}$$

of all assignments on J_s. As a consequence, if the unknown sequence is indeed A, then the capital increases by the constant factor $1/\delta > 1$ during each stage s, hence b succeeds on A.

(b) Fix any sequence A that is i.o. p-btt-autoreducible. Again it suffices to show that A is not p-random, i.e., that there is a betting strategy b that is computable in polynomial time and succeeds on A. The ideas underlying the construction of b are similar to the ones used in the proof of assertion (a). The remaining differences relate to the fact that b now has to be computable in polynomial time and hence cannot perform an essentially unbounded search for places where the reduction does not yield $\perp$.

Fix an oracle Turing machine $M = (g, h)$ that p-btt-autoreduces A. For the scope of this proof, given a word w, we write x_w for $\mathrm{s}_{|w|}$; i.e., if w is viewed as prefix of a sequence X, then x_w is the first word y such that $X(y)$ is not encoded into w. Furthermore, we write m_w and n_w for the length of w and of x_w. For any word w, let M_w be defined by

$$M_w(X, x) = M(\langle X, w \rangle, x),$$

i.e., in order to obtain M_w, the word w is hard-wired into M by overwriting the length m_w prefix of the oracle by w. For all words w, let

$$J(w) = \{x_w\} \cup \{z : z \in g(x_w) \text{ and } x_w < z\}.$$

For the scope of this proof, call a word w *promising* if $M_w(X, x_w) \neq \perp$ for some sequence X. For any promising word w, among all such sequences X let $X(w)$ be the one such that the restriction of X to $J(w) \setminus \{x_w\}$ is minimum and let γ_w be the corresponding restriction. Let $i(w) = M_w(X(w), x_w)$ and extend γ_w to an assignment α_w on $J(w)$ by letting $\alpha_w(x_w) = 1 - i(w)$. Similar to the proof of the first assertion we can argue that for any promising prefix w of A, the restriction of A to $J(w)$ differs from α_w because otherwise M did not i.o. btt-autoreduce A.

We construct now a betting strategy b that is computable in polynomial time and succeeds on A. Define a partition $I_0, I_1, \ldots$ of the set of all words by

$$I_s := \{x : d(s) \leq |x| < d(s+1)\} \quad \text{where } d(0) = 1, \;\; d(s+1) = 2^{d(s)}.$$

There are infinitely many promising prefixes of A because in particular any prefix w of A where $M(A, x_w) \neq \perp$ is promising. In the sequel we assume that there are infinitely many promising prefixes w of A such that x_w is contained in one of the even intervals $I_0, I_2, \ldots$, and we omit the virtually identical considerations for the symmetrical case where there are infinitely many promising prefixes w where x_w is contained in an odd interval.

The betting strategy b works similar to the one used in the proof of assertion (a) and we leave the details of its construction and verification to the reader. By definition of the intervals I_i, we can pick an index s_0 such that for all $s > s_0$ and for all w with x_w in I_{2s}, the set $J(w)$ is contained in the double interval $I_{2s} \cup I_{2s+1}$. During stage s, the betting strategy b bets on words in this double interval as follows. If $s \leq s_0$ or if there is no promising prefix v of the current input w such that x_v is in I_{2s}, abstain from betting. Otherwise, let v_s be the shortest promising prefix of w such that x_v is in I_{2s} and bet against the event that the restriction of the unknown sequence to $J(v_s)$ is equal to α_{v_s}. In case the unknown sequence is indeed A, there are infinitely many stages where the otherwise case applies and during each such stage, the capital increases at least by some constant positive factor. □

The proof of Theorem 22 actually shows that any i.o. p-btt-autoreducible sequence can be covered by an m^2-martingale. By standard techniques [5], one can construct an m^3-martingale that covers all sequences that are covered by an m^2-martingale, hence the class of i.o. p-btt-autoreducible sequences has measure 0 with respect to betting strategies that are computable in polynomial time.

Remark 23. Theorem 22 extends by essentially the same proof to truth-table-reducibilities that may ask an arbitrary number of queries that are strictly smaller than the current input plus a constant number of queries that are larger.

6 A sharp bound on the density of guessed bits

From Theorem 19 we know that every random sequence is i.o. autoreducible. An interesting problem involves finding lower and upper bounds to the frequency at which computable autoreductions may yield bits of a random sequence.

Definition 24. *For all $m > 0$, the* density $\rho(E, m)$ *of a set E up to m is defined by*

$$\rho(E, m) \;=\; \frac{|E \cap \{s_0 \ldots s_{m-1}\}|}{m}\,. \tag{6}$$

An oracle Turing machine M i.o. T-autoreduces a sequence X with density $r(m)$ *if M i.o. T-autoreduces X on a set E such that $\rho(E, m) \geq r(m)$ for all $m > 0$*

(i.e., the density of the set of words x such that M guesses $X(x)$ is always at least $r(m)$).

A sequence A is called i.o. T-autoreducible with density $r(m)$ *if there is some oracle Turing machine that i.o. T-autoreduces A with density $r(m)$. Concepts like* i.o. tt-autoreducible with density $r(m)$ *are defined in the same manner.*

Thus, autoreducibility with density $r(m)$ measures the density of the guessed bits of an autoreduced sequence A in the sense that the ratio of guessed bits to bits considered is at least $r(m)$. It should be noted that the concept of density is sort of inverse to the previous concept of *rate* [17] where an autoreduction has rate $r(m)$ if the mth place guessed is not larger than $s_{r(m)}$.

We now study the question of what is the highest density a reducibility may achieve with random sequences of a certain type. A lower bound on the achievable density is easily obtained from the proof of Theorem 19, by noting that the autoreductions employed there obtain densities that depend on the length of the codewords in the employed error-correcting codes. In Example 25, we state corresponding bounds that are obtained by choosing the lengths of the codewords according to specific converging sums. Afterwards, in Theorem 29, we improve on this bound by elaborating the proof of Theorem 19.

Example 25 *Fix any rec-random sequence R. Let $j_1,\ j_2\ \ldots$ be any non-decreasing computable sequence such that*

$$\sum_{k=0}^{\infty} \frac{1}{l_k} < \infty, \qquad \text{where } l_k = 2^{j_k} - 1 > 0.$$

Then also the sum over the $\frac{1}{l_k+1}$ converges, hence as in the proof of Theorem 19 we can find perfect one-error-correcting codes C_k of codeword length l_k such that the sequence R can be written as $w_1 w_2 \ldots$ where each word w_k has length l_k and almost all w_k are in C_k. Moreover, by hard-wiring finitely many bits of R plus applying the solution of the hat problem to the words w_k, we obtain an autoreduction of R which guesses exactly one bit of each of the words w_k. In this situation, let E be the set of the bits that are guessed correctly and let $\rho(E,m)$ be the density of E defined in (6).

We aim at deriving upper and lower bounds for $\rho(E,m)$. If we let $i(m)$ be the maximum index i such that $s_i = l_1 + \ldots + l_i \leq m$, then by construction for all m we have

$$\frac{i(m)}{m} \leq \rho(E,m) \leq \frac{i(m)+1}{m}\,, \tag{7}$$

i.e., in order to bound $\rho(E,m)$ it suffices to bound $i(m)$.

If, as in the proof of Theorem 29, we let $l_k = 2^k - 1$, then s_i is $2^{i+1} - 1$, and accordingly $i(m)$ and $\rho(E,m)$ are approximately $\log m$ and $\log m/m$, respectively. In order to improve on this density, we might try to use values for l_k that grow slower. If we fix $\delta > 1$, the sum $\sum 1/(k \log^\delta k)$ converges, hence we might choose l_k as the least number of the form $2^j - 1$ that is greater or equal to $k \log^\delta k$. Some easy calculations show that for almost all i, we have $i^2 \leq s_i \leq i^{2.001}$, thus, *by* $s_{i(m)} \leq m \leq s_{i(m)+1}$,

$$i(m)^2 \leq m \leq (i(m)+1)^{2.001} \leq i(m)^{2.002}, \qquad \textit{hence} \qquad m^{\frac{1}{2.002}} \leq i(m) \leq m^{\frac{1}{2}}.$$

By (7), for almost all m the density $\rho(E,m)$ is between $m^{\frac{1}{2.002}}/m$ and $m^{\frac{1}{2}}/m$, i.e., the achieved density is approximately equal to $1/\sqrt{m}$.

In what follows, we prove something much stronger than the result from Example 25; namely that for any computable function r that goes non-ascendingly to 0, any rec-random sequence is i.o. truth-table-autoreducible with density $r(m)$. This result is consummately complemented by our next theorem, which shows that rec-random sequences are never i.o. truth-table-autoreducible with positive constant density (i.e., with density $r(m) = \varepsilon m$ for some $\varepsilon > 0$), and that a similar assertion holds with respect to Martin-Löf random sequences and i.o. Turing-autoreducibility.

Theorem 26. (a) *No rec-random sequence is i.o. tt-autoreducible with positive constant density.*
(b) *No Martin-Löf-random sequence is i.o. T-autoreducible with positive constant density.*

Proof. Assertions (a) and (b) are proved by showing that if a sequence is i.o. autoreducible with constant density, then the sequence cannot be random. In both cases, the argument relies on Claims 1 and 2 below.

Fix an oracle Turing machine M and a rational $\varepsilon_0 > 0$. We want to show that if M i.o. autoreduces a sequence with density $\varepsilon_0 m$, then this sequence cannot be random; i.e., an appropriate betting strategy succeeds on this sequence. Recall that the cylinder generated by a word w is the the class $w\{0,1\}^\infty$ of all sequences that extend w. We argue that for any w, the fraction of sequences in this cylinder that are i.o. autoreduced by M with density ε_0 is bounded by a constant $\delta < 1$, which does not depend on w. Let $\varepsilon = \varepsilon_0/2$ and for any $m > 0$ let

$$i(m) = \left\lceil \frac{m}{\varepsilon} \right\rceil, \qquad I(m) = \{s_m, \ldots, s_{m+i(m)-1}\}, \tag{8}$$

that is, assuming $|w| = m$, the interval $I(m)$ contains the first $i(m)$ words that are not in the domain of w. For any sequence X and any finite set I, let

$$\begin{aligned} \mathrm{correct}(X,I) &= |\{x \in I : M(X,x) = X(x)\}|, \\ \mathrm{incorrect}(X,I) &= |\{x \in I : M(X,x) = 1 - X(x)\}|. \end{aligned}$$

That is, for all inputs x in I such that $M(X,x)$ is defined and differs from $\perp$, we count for how many of these inputs the guess $M(X,x)$ is correct and for how many it is incorrect. In the remainder of this proof and with M and ε understood from the context, we say a sequence X is consistent with a word w of length m if

(i) X is an extension of w,

(ii) for all x in $I(m)$, the value $M(X, x)$ is defined and is computed without querying the oracle at place x,

(iii) $\text{incorrect}(X, I(m)) = 0$,

(iv) $\text{correct}(X, I(m)) \geq \varepsilon |I(m)|$.

Claim 1 *If M i.o. T-autoreduces a sequence with density ε_0, then this sequence is consistent with any prefix of itself.*

Proof. Fix any sequence A that satisfies the assumption of the claim and consider any prefix w of A. Then the conditions (i), (ii), and (iii) are satisfied by definition (recall that $M(A, x)$ is always defined in case M i.o. T-autoreduces A). If condition (iv) was false, then contrary to our assumption on A the oracle Turing machine M with oracle A would guess in the interval $I(m)$ and among the m smaller words in total strictly less than

$$m + \varepsilon |I(m)| \;\leq\; 2\varepsilon |I(m)| \;=\; \varepsilon_0 |I(m)| \tag{9}$$

bits, where the relations in (9) hold by (8) and by choice of ε. □

For any word w, let the sequence X_w be an extension of w that is obtained by the chance experiment where the bits of X_w that are not already determined by w are obtained by independent tosses of a fair coin. Let $\delta = 1/(1 + \varepsilon)$ and observe that $\delta < 1$ due to $\varepsilon > 0$.

Claim 2 *For any word w, the probability that X_w is consistent with w is at most δ.*

Proof. Fix any word w of length m. The key observation in the proof of Claim 2 is the following. If we examine for all inputs x in $I(m)$ such that the value $M(X_w, x)$ is in $\{0, 1\}$ at all, whether this value is a correct guess in the sense that it agrees with $X_w(x)$, then the expected number of correct and incorrect guesses is the same; i.e.,

$$\mathbf{E}[\text{correct}(X_w, I(m))] \;=\; \mathbf{E}[\text{incorrect}(X_w, I(m))]\,. \tag{10}$$

For a proof, first consider a single input x in $I(m)$. The assignment to X_w at x and at the places different from x are stochastically independent, hence by (ii) the same holds for $X_w(x)$ and $M(X_w, x)$. Furthermore, since $X_w(x)$ is chosen uniformly from $\{0, 1\}$, it follows that the probability for a correct and for an incorrect answer at x are both exactly half of the probability that $M(X_w, x)$ is in $\{0, 1\}$. Hence also the expected number of correct and incorrect answers at x coincide and (10) follows by linearity of expectation.

We conclude the proof of Claim 2 by distinguishing two cases.

Case I: $\mathbf{E}[\text{correct}(X_w, I(m))] \leq \delta\varepsilon |I(m)|$.

The random variable $\text{correct}(X_w, I(m))$ is non-negative, hence the case assumption implies that the probability that $\text{correct}(X_w, I(m))$ is at least $\varepsilon |I(m)|$ is at most δ. So also the probability that X_w is consistent with w is at most δ by condition (iv) in the definition of consistency.

Case II: $\mathbf{E}[\text{correct}(X_w, I(m))] > \delta\varepsilon|I(m)|$.

By (10), we also have $\mathbf{E}[\text{incorrect}(X_w, I(m))] > \delta\varepsilon|I(m)|$. The latter implies that the probability that $\text{incorrect}(X_w, I(m))$ is strictly larger than 0 is at least $\delta\varepsilon$ because by definition, the random variable $\text{incorrect}(X_w, I(m))$ is bounded by $|I(m)|$. Due to condition (iii), the probability that X_w is consistent is then at most

$$1 - \delta\varepsilon = 1 - \frac{\varepsilon}{1+\varepsilon} = \frac{1}{1+\varepsilon} = \delta \ . \qquad \square$$

Proof of (a). We can assume that M is in fact a tt-reduction, say, $M = (g, h)$. Fix any sequence A such that M i.o. tt-autoreduces A with density $r(m) = \varepsilon_0 m$. It suffices to show that A is not rec-random, and this is done by constructing a computable betting strategy that succeeds on A.

The set of all words is partitioned into consecutive intervals $J_0, J_1, \ldots$ where the cardinality of J_i is denoted by l_i (i.e., J_0 contains the first l_0 words, J_1 the next l_1 words, and so on). The J_i are defined via specifying the l_i inductively. For all i, let $s_i = l_0 + l_1 + \ldots + l_i$. Let $l_0 = 1$ and for all $i \geq 1$ choose l_i so large that it contains $I(s_{i-1})$ as well as the query sets $g(x)$ for all x in $I(s_{i-1})$. This way, for any word w of length s_i, the consistency with w of any sequence X that extends w depends only on the restriction of X to the interval J_{i+1}. Call an assignment on J_{i+1} consistent with such a word w if the assignment is the restriction of a sequence that is consistent with w. By Claims 1 and 2, for any given word w of length s_i the following assertions hold with respect to the assignments on J_{i+1}.

- If w is a prefix of A, then the assignment obtained by restricting A to J_{i+1} is consistent.
- The consistent assignments comprise a fraction of at most δ of all assignments.

Now consider the betting strategy that for each interval J_i, uses the capital accumulated up to the first element of the interval in order to bet according to Remark 15 against all assignments on this interval that are not consistent with the already seen length s_i prefix of the unknown sequence. By the preceding discussion, in case the unknown sequence is indeed A then on each interval, b increases its capital at least by the constant factor $1/\delta > 1$; i.e., in this case b succeeds on A. Furthermore, the betting strategy b is computable since consistency of assignments can be decided effectively.

Proof of (b). For a given word w, let $\boldsymbol{E}_w$ be the class of all sequences that are consistent with w. For a word w of length $m \geq 1$, call a word u a consistent extension of w if for some sequence U,

- w is a prefix of u and u is a prefix of U, while U is consistent with w,
- the domain of u contains $I(m)$ and all queries that are made while computing $M(U, x)$ for any x in $I(m)$.
- for all x in $I(m)$, the computation of $M(U, x)$ terminates in at most $|u|$ steps,

Let $E(w)$ be the set of minimum consistent extensions of w (i.e., $E(w)$ contains any word if the word itself but none of its prefixes is a consistent extension of w). Then for any non-empty word w,

(i) $\boldsymbol{E}_w$ is the disjoint union of the cylinders generated by words in $E(w)$,
(ii) $\boldsymbol{E}_w$ comprises at most a fraction of δ of the cylinder generated by w,
(iii) $E(w)$ is computably enumerable in w.

Assertions (ii) and (iii) hold, respectively, due to Claim 2 and because for given w and u, it can be effectively checked whether u is a consistent extension of w. Concerning assertion (i), first observe that for any sequence in $\boldsymbol{E}_w$ all prefixes of sufficient length are consistent extensions of w. On the other hand, for any consistent extension u of w, the cylinder generated by u is a subclass of $\boldsymbol{E}_w$ because any sequence that extends u is consistent with w. Furthermore, by the minimality condition in the definition of $E(w)$, the words in $E(w)$ are mutually incomparable, hence the cylinders generated by these words are mutually disjoint.

Let $\boldsymbol{C}$ be the class of all sequences that are i.o. tt-autoreduced by M with density ε_0. We conclude the proof of assertion (b) by constructing a Martin-Löf null cover for $\boldsymbol{C}$. Let $V_0 = \{0,1\}$, and for all $i \geq 0$, let

$$V_{i+1} \quad = \quad \bigcup_{w \in V_i} E(w)\ .$$

Then the sets V_i are uniformly computably enumerable; i.e., $V_i = \mathrm{W}_{h(i)}$ for some computable function h. Using (i) and the fact that any sequence in this class is consistent with all of its prefixes, an induction argument shows that for all i the class $\boldsymbol{C}$ is contained in the union of the cylinders generated by the words in $\mathrm{W}_{h(i)}$. Furthermore, another induction argument, which uses (ii) in the induction step, shows that the union of the cylinders generated by $\mathrm{W}_{h(i)}$ has measure of at most δ^i. So if we fix a constant c such that δ^c is at most $1/2$, then $\mathrm{W}_{h(ci)}$ has measure at most $1/2^i$. In summary, $\mathrm{W}_{h(c0)}, \mathrm{W}_{h(c1)}, \ldots$ is a Martin-Löf null cover for $\boldsymbol{C}$. □

The concept of density can be relativized to an infinite subset Z of all words, i.e., we might say an oracle Turing machine M i.o. T-autoreduces a given sequence with *density $r(m)$ relative to Z* if the fraction of guessed bits among the first m words in Z is always at least $r(m)$. By a straightforward adaptation of the proof of Theorem 26, it is possible to demonstrate Corollary 27, from which Corollary 28 can be obtained as a special case.

Corollary 27. (a) *No rec-random sequence is i.o. tt-autoreducible with positive constant density relative to an infinite computable set.*
(b) *No Martin-Löf-random sequence is i.o. T-autoreducible with positive constant density relative to an infinite computable set.*

Corollary 28. *(a) No rec-random sequence is i.o. tt-autoreducible on a computable set.*
(b) No Martin-Löf-random sequence is i.o. T-autoreducible on a computable set.

Trakhtenbrot [39] observed that no Kolmogorov-Loveland stochastic sequence [25] is T-autoreducible, and his argument easily extends to i.o. T-autoreducibility on a computable set; from the latter, assertion (b) in Corollary 28 is immediate, because the Martin-Löf-random sequences form a proper subclass of the Kolmogorov-Loveland stochastic sequences.

The "negative" assertions in Theorem 26 and Corollaries 27 and 28 are complemented by the two following "positive" assertions due to Merkle and Mihailović [30]. There are rec-random sequences that are weak truth-table-autoreducible. There are Martin-Löf random sequences that are selfreducible with respect to the reducibility *being recursively enumerable in*, where selfreducible means autoreducible while asking only queries less than the current input.

Apparently, the arguments used in this section do not carry over to show that p-random sequences cannot be p-T- or p-tt-autoreducible; in fact, the latter assertion relates to major open problems in complexity theory. These relations are implicit in work of Buhrman et al. [14], from which, among others, the two following implications are immediate. First, if there are p-random sequences that are p-tt-autoreducible, then the complexity classes **MA** and **EXP** are the same. Second, if no p-random sequence is p-tt-autoreducible, then the complexity classes **BPP** and **EXP** differ. The first implication is immediate by the result of Buhrman et al. that $\mathbf{MA} \neq \mathbf{EXP}$ implies that the p-tt-autoreducible sequences have measure 0 with respect to polynomial-time computable betting strategies. For a proof of the second implication assume that $\mathbf{BPP} = \mathbf{EXP}$. By a result of Allender and Strauss, any p-random sequence is p-tt-complete for **BPP**. But **EXP** contains p-random sequences, hence by our assumption some p-random sequence is p-tt-complete for **EXP**. The assertion now follows by the result of Buhrman et al. that $\mathbf{BPP} = \mathbf{EXP}$ implies that all p-tt-complete sequences for **EXP** are p-tt-autoreducible.

Recall the negative result on constant bounds in Theorem 26, i.e., a rec-random sequence cannot be i.o. tt-autoreduced with constant positive density. This result is essentially matched by Theorem 29, which states that for any computable, rational-valued function r that goes non-ascendingly to 0, every rec-random sequence is i.o. tt-autoreducible with density $r(m)$.

Theorem 29. *Let $g\colon \mathbb{N} \to \mathbb{N}$ be any computable function that is unbounded and non-decreasing. Then every rec-random sequence is i.o. tt-autoreducible with density $r(m) = 1/g(m)$.*

Proof. Fix any rec-random sequence R. It suffices to show that R is i.o. tt-autoreducible with density $r(m)$ by some oracle Turing machine M. For every $k > 0$, let

$$l_k = 2^k - 1 \ .$$

For further use, fix perfect one-error-correcting codes C_k of codeword length l_k such that given x, we can decide effectively whether x is in one of the codes C_k. The computable function g is unbounded, thus we can define a computable sequence $t_0 < t_1 < \ldots$ where $g(t_k) \geq l_{k+1}$ and hence

$$r(t_k) \ \leq \ \frac{1}{l_{k+1}} \ .$$

For every $k > 0$, partition the set of all words into consecutive intervals $J_k^1, J_k^2, \ldots$ of length l_k; i.e., J_k^1 contains the first l_k words, J_k^2 contains the next l_k words, and so on. Let c_k be minimum such that the $(t_k{+}1)$th word s_{t_k} is in $J_k^{c_k}$ and let

$$H_k = J_k^2 \cup \ldots \cup J_k^{c_k} .$$

We construct M from oracle Turing machines $M_1, M_2, \ldots$, to which we refer as modules. Intuitively, module k is meant to ensure density $1/l_k$ in the interval between s_0 and s_{t_k}. Before defining the modules, we describe their properties and how they are combined to form M.

Module k never queries any place outside the set $J_k^1 \cup H_k$ and outputs $\perp$ on all inputs that are not in H_k. Furthermore, on oracle R, module k never makes a wrong guess and guesses exactly one bit in each of the intervals J_k^2 through $J_k^{c_k}$. In addition, we will ensure that the M_k are uniformly effective in the sense that there is an oracle Turing machine M_0 such that the values $M_k(x, X)$ and $M_0(\langle x, k\rangle, X)$ always either are both undefined or are both defined and identical.

Then M is obtained by running the modules in parallel as follows. For input x, if x is equal to s_0 or s_1, then M just outputs $R(x)$. Otherwise, M determines the finitely many k such that $x \in H_k$ and simulates the corresponding modules with input x and the current oracle. If any of these modules outputs a value different from $\perp$, then M outputs the value output by the least such module; otherwise, M outputs $\perp$. From the properties of the modules stated so far, we can already prove that M works as required.

Claim 1 *The oracle Turing machine M i.o. tt-autoreduces the sequence R with density $r(m)$.*

Proof. From the module assumptions, it is immediate that M is computable and that M queries its oracle non-adaptively and never queries the oracle at the current input. Moreover, on oracle R, all guesses of the modules and hence all guesses of M are correct.

Let E be the set of all inputs x such that $M(R, x)$ differs from $\perp$. Fix k and assume that m is in J_k^1 through $J_k^{c_k}$. Then we have

$$\rho(E, m) \geq 1/l_k \tag{11}$$

This follows because M guesses the first two bits, while module k and hence M guess at least one bit in every interval except the first one, where the intervals have length l_k. So if m is in interval J_k^j, up to m there are at most jl_k words and at least j guesses, hence the density up to m is at least $1/l_k$ and (11) holds.

In order to prove $\rho(E, m) \geq r(m)$, fix any m and choose k such that $t_{k-1} \leq m < t_k$. Then we have

$$\rho(E, m) \geq \frac{1}{l_k} \geq r(t_{k-1}) \geq r(m) , \tag{12}$$

where the inequalities follow because m is in J_k^1 through $J_k^{c_k}$ and hence (11) applies, by choice of t_{k-1}, and since r is non-ascending. $\square$

In order to construct modules that have the required properties, let w_k^j be the restriction of R to J_k^j; i.e.,

$$R = w_k^1 w_k^2 \dots \qquad \text{where } |w_k^j| = l_k, \text{ and let} \qquad w_k = w_k^1 \oplus \dots \oplus w_k^{c_k} \ ,$$

where $\oplus$ is bit-wise exclusive-or. The idea underlying the construction of the modules is as follows. The code words in C_k comprise such a small fraction of all words of length l_k that in case infinitely many words w_k were in C_k, there would be a computable betting strategy that succeeds on R. But R is assumed to be rec-random, hence w_k is not in C_k for almost all k. So in order to guess bits of w_k and then also of w_k^1 trough $w_k^{c_k}$, we can apply the solution to the hat problem described in Remark 6.

Claim 2 *For almost all k, w_k is not in C_k.*

Proof. Suppose we bet on the bits of an unknown sequence X. Similar to the definition of w_k, let v_k^j be the restriction of X to the interval J_k^j and let v_k be equal to $v_k^1 \oplus \dots \oplus v_k^{c_k}$. Recall that C_k contains a fraction of $a_k = 1/2^k$ of all words of length l_k and observe that the mapping

$$o_k : u \mapsto v_k^1 \oplus \dots \oplus v_k^{c_k - 1} \oplus u$$

is a bijection of $\{0,1\}^{l_k}$. Thus o_k maps just a fraction of a_k of all length l_k words to C_k. When betting on the places in $J_k^{c_k}$, we have already seen v_k^1 through $v_k^{c_k-1}$. Under the assumption that v_k is in C_k, we can exclude all but a fraction of $1/a_k$ of the possible assignments to $J_k^{c_k}$ and hence, by betting in favor of these assignments according to Remark 15, we can increase our capital by a factor of $1/a_k$ in case the assumption is true.

Now consider the following betting strategy. For every k, a portion of a_k of the initial capital 1 is exclusively used for bets on the the interval $J_k^{c_k}$. On each such interval, the bets are in favor of the a_k-fraction of assignments that o_k maps to C_k. Then the capital a_k increases to 1 for all k such that v_k is in C_k and consequently the betting strategy succeeds on any sequence such that the latter occurs for infinitely many k. But no computable betting strategy can succeed on the rec-random sequence R, hence Claim 2 follows. We leave it to the reader to show that this strategy is indeed computable. Observe in this connection that each word is contained in at most finitely many intervals of the form $J_k^{c_k}$ and consequently at most finitely many of the strategies related to these intervals might act in parallel. □

It remains to construct the modules. By Claim 1, fix k_0 such that w_k is not in C_k for all $k > k_0$. First assume $k \leq k_0$. Consider the least elements of the intervals J_k^2 through $J_k^{c_k}$ and for all these x, hard-wire $R(x)$ into module k. On all these x, module k outputs $R(x)$, while the module outputs $\perp$ on all other inputs.

Next assume $k > k_0$. On inputs that are not in H_k, module k simply outputs $\perp$. Now consider an input x in H_k and suppose that

$$x \text{ is element } i_0 \text{ of interval } J_k^{j_0}, \quad 0 \leq i_0 < l_k, \ \ 2 \leq j_0 \leq c_k \ .$$

For the given oracle X, define v_k and the v_k^j as in the proof of Claim 2; i.e., v_k^j is the restriction of X to J_k^j and v_k is the bit-wise exclusive-or of the v_k^j. Then on input x, module k queries its oracle at the remaining words in $J_k^1 \cup H_k$; i.e., the module obtains all bits of the words

$$v_k^j \quad = \quad v_k^j(0)\, v_k^j(1)\, \ldots\, v_k^j(l_k - 1)\;, \quad j = 1, \ldots, c_k\;,$$

except for the bit $X(x) = v_k^{j_0}(i_0)$. In order to guess this bit, the module tries to guess the bit $v_k(i_0)$. From the latter and from the already known bits $v_k^j(i_0)$ for $j \neq j_0$, the bit $v_k^{j_0}(i_0)$ can then be computed easily since $v_k(i_0)$ is the exclusive-or of the $v_k^j(i_0)$.

In order to guess $v_k(i_0)$, module k mimics the solution of the hat problem with l_k players that is given in Remark 6. More precisely, the module computes the remaining bits of v_k from the $v_k^j(i)$ and obtains two consistent words u_0 and u_1. In case for some r, the word u_r is in C_k, the module guesses $u_{1-r}(i_0)$ while the module abstains from guessing, otherwise. By assumption on k, on oracle R the word v_k is not in C_k, hence the discussion in Remark 6 shows that module k never guesses incorrectly and guesses exactly one bit in each of the intervals J_k^2 through $J_k^{c_k}$ (in fact, for some i, in each interval bit i is guessed). $\square$

Acknowledgments. We especially thank Ken Rose (Santa Barbara) for his enlightening discussions on error-correcting codes. We also acknowledge helpful hints from Klaus Ambos-Spies (Heidelberg), Charles Akemann (Santa Barbara), Dieter van Melkebeek (Madison), Klaus W. Wagner (Würzburg), and anonymous referees.

References

1. K. Ambos-Spies, P-mitotic sets. In E. Börger et al., editors, *Decision Problems and Complexity.* Lecture Notes in Computer Science 171:1–23. Springer-Verlag, 1984.
2. K. Ambos-Spies, H. Fleischhack, and H. Huwig, Diagonalizations over polynomial time computable sets. *Theoretical Computer Science* 51:177–204, 1987.
3. K. Ambos-Spies, Resource bounded genericity. In S. B. Copper et al., editors, *Computability, Enumerability, Unsolvability. Directions in Recursion Theory.* London Mathematical Society Lecture Note Series 224:1–60. Cambridge University Press, 1996.
4. K. Ambos-Spies and A. Kucera, Randomness in computability theory. In P. A. Cholak et al., editors, *Computability Theory and its Applications. Current Trends and Open Problems.* Proceedings of a 1999 MS-IMS-SIAM joint summer research conference, Boulder, USA, American Mathematical Society (AMS). Contemporary Mathematics 257:1–14, 2000.
5. K. Ambos-Spies and E. Mayordomo, Resource-bounded measure and randomness. In A. Sorbi, editor, *Complexity, Logic, and Recursion Theory,* p. 1-47. Dekker, New York, 1997.
6. K. Ambos-Spies, S. A. Terwijn, and X. Zheng, Resource bounded randomness and weakly complete problems. *Theoretical Computer Science* 172:195–207, 1997.

7. A. Arslanov, On the phenomenon of autocomputability. *Computing: the Australasian Theory Symposium 2000*, Electronic Notes in Theoretical Computer Science 31, Elsevier, Amsterdam, 2000.
8. J. Aspnes, R. Beigel, M. L. Furst, and S. Rudich, The expressive power of voting polynomials. *Combinatorica* 14:135–148, 1994.
9. R. Beigel, L. Fortnow, and F. Stephan. Infinitely often autoreducible sets. Technical Report 2002-029, NEC Research Institute, 2002.
10. J. L. Balcázar, J. Díaz, and J. Gabarró, *Structural Complexity*, volume I. Springer-Verlag, 1995.
11. J. L. Balcázar, J. Díaz, and J. Gabarró, *Structural Complexity*, volume II. Springer-Verlag, 1990.
12. J. L. Balcázar and E. Mayordomo, A note on genericity and bi-immunity. In *Proceedings of the Tenth Annual Structure in Complexity Theory Conference 1995*, p. 193–196. IEEE Computer Society Press, 1995.
13. H. Buhrman, L. Fortnow, D. van Melkebeek, and L. Torenvliet, Using autoreducibility to separate complexity classes. *SIAM Journal on Computing* 29:1497–1520, 2000.
14. H. Buhrman, D. van Melkebeek, K. W. Regan, D. Sivakumar, and M. Strauss, A generalization of resource-bounded measure, with an application to the BPP vs. EXP problem. *SIAM Journal on Computing* 30:576–601, 2000.
15. C. Calude, *Information and Randomness*, Springer-Verlag, 1994.
16. R. Daley, On the simplicity of busy beaver sets. *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik* 24:207–224, 1978.
17. T. Ebert, *Applications of Recursive Operators to Randomness and Complexity.* Ph.D. Thesis, University of California at Santa Barbara, Santa Barbara, U.S.A., 1998.
18. T. Ebert and W. Merkle, Autoreducibility of random sequences: a sharp bound on the density of guessed bits. In K. Diks and W. Rytter, editors, *Mathematical Foundations of Computer Science 2002*, Lecture Notes in Computer Science 2420:221–233, Springer-Verlag, 2002.
19. T. Ebert and H. Vollmer, On the autoreducibility of random sequences. In M. Nielsen and B. Rovan, editors, *Mathematical Foundations of Computer Science 2000*, Lecture Notes in Computer Science 1893:333–342, Springer-Verlag, 2000.
20. K. H. Kim and F. W. Roush, *Applied Abstract Algebra.* John Wiley and Sons, New York, 1983.
21. C. Jockusch and M. Paterson, Completely autoreducible degrees. *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik* 22:571–575, 1976.
22. R. Ladner, Mitotic recursively enumerable sets. *Journal of Symbolic Logic* 38:199–211, 1973.
23. F. Thomson Leighton, *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes.* Morgan Kaufmann Publishers, San Mateo, California, 1992.
24. H. W. Lenstra and G. Seroussi, On hats and other covers. IEEE International Symposium on Information Theory, Lausanne, Switzerland, 2002.
25. M. Li and P. Vitányi, *An Introduction to Kolmogorov Complexity and Its Applications.* Second edition. Springer-Verlag, 1997.
26. J. H. Lutz, Almost everywhere high nonuniform complexity. *Journal of Computer and System Sciences* 44:220–258, .
27. J. H. Lutz, The quantitative structure of exponential time. In L. A. Hemaspaandra and A. L. Selman, editors, *Complexity Theory Retrospective* II, p. 225–260, Springer-Verlag, 1997.

28. P. Martin-Löf, The definition of random sequences. *Information and Control* 9:602–619, 1966.
29. E. Mayordomo. *Contributions to the Study of Resource-Bounded Measure.* Doctoral dissertation, Universitat Politècnica de Catalunya, Barcelona, Spain, 1994.
30. W. Merkle and N. Mihailović, On the construction of effective random sets. In K. Diks and W. Rytter, editors, *Mathematical Foundations of Computer Science 2002*, Lecture Notes in Computer Science 2420:568–580, Springer-Verlag, 2002.
31. S. Robinson, Why mathematicians now care about their hat color. *New York Times*, April 10, 2001.
32. S. Rudich, private communication, 2001.
33. D. E. Shasha, Crowns of the Minotaur, *Scientific American*, October 2001.
34. A. N. Shiryaev, *Probability.* Springer-Verlag, New York, 1995.
35. C.-P. Schnorr, A unified approach to the definition of random sequences. *Mathematical Systems Theory* 5:246–258, 1971.
36. C.-P. Schnorr, *Zufälligkeit und Wahrscheinlichkeit.* Lecture Notes in Mathematics 218, Springer-Verlag, 1971.
37. R. I. Soare, *Recursively Enumerable Sets and Degrees.* Perspectives in Mathematical Logic. Springer-Verlag, 1987.
38. S. A. Terwijn, *Computability and Measure.* Doctoral dissertation, Universiteit van Amsterdam, Amsterdam, Netherlands, 1998.
39. B. A. Trakhtenbrot, On autoreducibility. *Soviet Mathematics Doklady* 11:814-817, 1970.
40. J.H. van Lint, *Introduction to Coding Theory*, 3rd edition. Springer-Verlag, 1999.
41. D. van Melkebeek, *Randomness and Completeness in Computational Complexity*, Lecture Notes in Computer Science 1950, Springer-Verlag, 2000.
42. J. Ville, *Étude Critique de la Notion de Collectif.* Gauthiers-Villars, Paris, 1939.
43. Y. Wang, *Randomness and Complexity.* Doctoral dissertation, Universität Heidelberg, Mathematische Fakultät, INF 288, Heidelberg, Germany, 1996.