Contents lists available at SciVerse ScienceDirect

Theoretical Computer Science

journal homepage: www.elsevier.com/locate/tcs

Finite state complexity

Cristian S. Calude[a], Kai Salomaa[b,*], Tania K. Roblot[a]

[a] Department of Computer Science, University of Auckland, Auckland, New Zealand

[b] School of Computing, Queen's University, ONT. K7L 3N6 Kingston, Ontario, Canada

ARTICLE INFO

Article history:
Received 21 January 2011
Received in revised form 16 May 2011
Accepted 17 June 2011
Communicated by G. Rozenberg

Keywords:
AIT
Finite state complexity
Incompressibility

ABSTRACT

In this paper we develop a version of Algorithmic Information Theory (AIT) based on finite transducers instead of Turing machines; the complexity induced is called finite-state complexity. In spite of the fact that the Universality Theorem (true for Turing machines) is false for finite transducers, the Invariance Theorem holds true for finite-state complexity. We construct a class of finite-state complexities based on various enumerations of the set of finite transducers. In contrast with descriptional complexities (plain, prefix-free) from AIT, finite-state complexity is computable and there is no *a priori* upper bound for the number of states used for minimal descriptions of arbitrary strings. Upper and lower bounds for the finite-state complexity of arbitrary strings, and for strings of particular types, are given and incompressible strings are studied.

Algorithmic Information Theory [9,7] uses various measures of descriptional complexity to define and study various classes of "algorithmically random" finite strings or infinite sequences. This theory, based on the existence of a universal Turing machine (of various types), is very elegant and has produced many important results, as one can see from the latest monographs on the subject [17,14].

The incomputability of all descriptional complexities was an obstacle towards more "down-to-earth" applications of AIT (e.g. for practical compression). One possibility to avoid incomputability is to restrict the resources available to the universal Turing machine and the result is resource-bounded descriptional complexity [6]. Various models which have been studied in this area did not produce significant understanding of deterministic randomness (i.e. chaoticity and software-generated randomness).

Another approach is to restrict the computational power of the machines used. For example, the size of the smallest context-free grammar, or straight-line program, generating the singleton language $\{x\}$ is a measure of the descriptional complexity of x. This model, investigated since the '70s, has recently received much attention [10,16,15,18] (also because of connections with Lempel-Ziv encodings [15,18]). Further restricting the computational power, from context-free grammars to finite automata (DFA), one obtains automatic complexity [22]. A similar descriptional complexity measure for languages was considered in [21].

The first connections between finite-state machine computations and randomness have been obtained for infinite sequences. In [1] it was proved that every subsequence selected from a (Borel) normal sequence by a regular language is also normal. Characterisations of normal infinite sequences have been obtained in terms of finite-state gamblers, information lossless finite-state compressors and finite-state dimension: (a) a sequence is normal if and only if there is no finite-state gambler that succeeds on it [19,5], and (b) a sequence is normal if and only if it is incompressible by any information lossless finite-state compressor [24].

* Corresponding author. Tel.: +1 613 533 6073; fax: +1 613 533 6513.
E-mail addresses: cristian@cs.auckland.ac.nz (C.S. Calude), ksalomaa@cs.queensu.ca (K. Salomaa), trob048@aucklanduni.ac.nz (T.K. Roblot).

0304-3975/$ – see front matter © 2011 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2011.06.021

Computations with finite transducers are used in [13] for the definition of finite-state dimension of infinite sequences. The NFA-complexity of a string [10] can be defined in terms of finite transducers that are called in [10] "NFAs with advice"; the main problem with this approach is that NFAs used for compression can always be assumed to have only one state.

In this paper we define the *finite-state complexity of a finite string* x in terms of an enumeration of finite transducers and the input strings used by transducers which output x.

The main obstacle in developing a version of AIT based on finite transducers is the non-existence of a universal finite transducer (Theorem 3.1). To overcome this negative result we show that the set of finite transducers can be enumerated by a computable (even a regular) set (Theorems 2.1 and 2.2), and, based on this, we prove the Invariance Theorem (Theorem 3.2) for finite-state complexity. The finite-state complexity is computable and examples of finite-state complexities of some strings are presented.

Our notation is standard [4,7]. If X is a finite set then X^* is the set of all strings (words) over X with ε denoting the empty string. The length of $x \in X^*$ is denoted by $|x|$.

1. Finite transducers

A generalised finite transducer [4] is a tuple $T = (X, Y, Q, q_0, Q_F, E)$, where X is the input alphabet, Y the output alphabet, Q is the finite set of states, $q_0 \in Q$ is the start state, $Q_F \subseteq Q$ is the set of accepting states and $E \subseteq Q \times X^* \times Y^* \times Q$ is the finite set of transitions. If $e = (q_1, u, v, q_2) \in E$, $q_1, q_2 \in Q$, $u \in X^*$, $v \in Y^*$ is a transition from q_1 to q_2, we say that the input (respectively, output) label of e is u (respectively, v). Also, when the states are understood from the context we say that the transition e is labeled by u/v. Here we consider transducers where both the input and output alphabet is always binary, that is, $X = Y = \{0, 1\}$.

A generalised transducer T is said to be a *(deterministic sequential) transducer* [4] if it has no transitions with input label ε and for any $q \in Q$ and $i \in \{0, 1\}$ there exists a unique $q' \in Q$ and $v \in \{0, 1\}^*$ such that (q, i, v, q') is a transition of T. The set of transitions of a deterministic sequential transducer is fully represented by a function

$$\Delta : Q \times \{0, 1\} \to Q \times \{0, 1\}^*. \tag{1}$$

For a transducer all states are considered to be final. Hence a transducer can be given by a triple (Q, q_0, Δ) where Δ is as in (1).

Let π_1 (respectively, π_2) denote the projection from $Q \times \{0, 1\}^*$ to Q (respectively, to $\{0, 1\}^*$). In details, the function $T : \{0, 1\}^* \to \{0, 1\}^*$ computed by the transducer (Q, q_0, Δ) is defined by $T(\varepsilon) = \varepsilon$, $\quad T(xa) = T(x) \cdot \mu(\hat{\delta}(q_0, x), a)$, where $\delta(q, a) = \pi_1(\Delta(q, a))$, $\mu(q, a) = \pi_2(\Delta(q, a))$, $q \in Q$, $x \in \{0, 1\}^*$, $a \in \{0, 1\}$. Here the extension of δ, $\hat{\delta} : Q \times \{0, 1\}^* \to Q$ is defined by setting $\hat{\delta}(q, \varepsilon) = q$, $\hat{\delta}(q, xa) = \delta(\hat{\delta}(q, x), a)$, $q \in Q$, $x \in \{0, 1\}^*$.

Sometimes we use $\cdot$ to denote the concatenation of strings; that is the concatenation of strings x and y can be denoted either as xy or $x \cdot y$.

2. Regular enumerations of transducers

We use binary strings to encode transducers and prove that the set of all legal encodings of transducers is a computable, in some cases even regular, language. We encode a transducer by listing for each state q and input symbol $a \in \{0, 1\}$ the output and target state corresponding to the pair (q, a), that is, $\Delta(q, a)$. Thus, the encoding of a transducer is a list of (encodings of) states and output strings.

By bin(i) we denote the binary representation of $i \geq 1$. Note that for all $i \geq 1$, bin(i) always begins with 1; bin$(1) = 1$, bin$(2) = 10$, bin$(3) = 11, \ldots$; by string(i) we denote the binary string obtained by removing the leading 1 from bin(i), i.e. bin$(i) = 1 \cdot$ string(i). If $\mathrm{Log}(i) = \lfloor \log_2(i) \rfloor$, then $|\mathrm{string}(i)| = \mathrm{Log}(i)$, $i \geq 1$.

For $v = v_1 \cdots v_m$, $v_i \in \{0, 1\}$, $i = 1, \ldots, m$, we use the following functions producing self-delimiting versions of their inputs (see [7]): $v^{\dagger} = v_1 0 v_2 0 \cdots v_{m-1} 0 v_m 1$ and $v^{\diamond} = \overline{(1v)^{\dagger}}$, where $^-$ is the negation morphism given by $\overline{0} = 1$, $\overline{1} = 0$. It is seen that $|v^{\dagger}| = 2|v|$, and $|v^{\diamond}| = 2|v| + 2$.

In Table 1 we present the encodings of the first binary strings.

Consider a transducer T with the set of states $Q = \{1, \ldots, n\}$. The transition function Δ of T (as in (1)) is encoded by a binary string

$$\sigma = \mathrm{bin}(i_1)^{\ddagger} \cdot \mathrm{string}(i'_1)^{\diamond} \cdot \mathrm{bin}(i_2)^{\ddagger} \cdot \mathrm{string}(i'_2)^{\diamond} \cdots \mathrm{bin}(i_{2n})^{\ddagger} \cdot \mathrm{string}(i'_{2n})^{\diamond}, \tag{2}$$

where $\Delta(j, k) = (i_{2j-1+k} \bmod n, \mathrm{string}(i'_{2j-1+k}))$, $i_t, i'_t \geq 1$, $t = 1, \ldots, 2n$, $j = 1, \ldots, n$, $k \in \{0, 1\}$. We denote by $m \bmod n$ the smallest positive integer congruent with m modulo n.[1] In (2), $\mathrm{bin}(i_t)^{\ddagger} = \varepsilon$ if the corresponding transition of Δ is a self-loop, i.e. $\pi_1(\Delta(j, k)) = j$; otherwise, $\mathrm{bin}(i_t)^{\ddagger} = \mathrm{bin}(i_t)^{\dagger}$.

The transducer T encoded by σ is called $T^{S_0}_{\sigma}$, where S_0 is the set of all strings (2) where $1 \leq \mathrm{bin}(i_j) \leq n$ for all $j = 1, \ldots, 2n$.

[1] In (2) we use i_t instead of $i_t \bmod n$ in order to guarantee that the set of legal encodings of all transducers is regular, cf. Theorem 2.1.

Table 1
S_0 encoding.

n	bin(n)	bin(n)†	string(n)	string(n)$^\diamond$	$\lvert$bin(n)$^\dagger\rvert = \lvert$string(n)$^\diamond\rvert$
1	1	11	ε	00	2
2	10	1001	0	0110	4
3	11	1011	1	0100	4
4	100	100001	00	011110	6
5	101	100011	01	011100	6
6	110	101001	10	010110	6
7	111	101011	11	010100	6
8	1000	10000001	000	01111110	8

Theorem 2.1. *The set of all transducers can be enumerated by a regular language. More precisely, we can construct a regular set S_0 such that:* (a) *for every $\sigma \in S_0$, $T_\sigma^{S_0}$ is a transducer,* (b) *for every transducer T one can compute a code $\sigma \in S_0$ such that $T = T_\sigma^{S_0}$.*

Proof. We consider the languages $X = \{\text{bin}(n)^\dagger : n \geq 1\} = \{11, 1001, 1011, \ldots\}$, $Y = \{\text{string}(n)^\diamond : n \geq 1\} = \{00, 0110, 0100, \ldots\}$ and we define the language

$$S_0 = (((X \cup \{\varepsilon\})Y)^2)^*. \tag{3}$$

The languages X and Y are regular, hence S_0 is regular by (3).

The claim (a) follows from (2). For (b) we note that in view of the construction it is clear that every string $\sigma \in S_0$ has a unique factorisation of the form $\sigma = x_1 \cdot y_1 \cdots x_{2n} \cdot y_{2n}$, for appropriate strings $x_1, \ldots, x_{2n} \in X \cup \{\varepsilon\}$ and $y_1, \ldots, y_{2n} \in Y$. So, from σ we uniquely get the length n and the codes $x_s \cdot y_s$, for $s = 1, 2, \ldots, 2n$. Every x_s can be uniquely written in the form $x_s = \text{bin}(t_s)^\dagger$ and every y_s can be uniquely written in the form $y_s = \text{string}(r_s)^\diamond$.

Next we compute the unique transition encoded by $x_s \cdot y_s = \text{bin}(t_s)^\dagger \cdot \text{string}(r_s)^\diamond$ according to (2). First assume that $x_s \neq \varepsilon$. There are two possibilities depending on s being odd or even. If $s = 2i+1$, for $0 \leq i \leq n$, then $\Delta(s, 0) = (t_s \bmod n, \text{string}(r_s))$; if $s = 2i$, for $1 \leq i \leq n$, then $\Delta(s, 1) = (t_s \bmod n, \text{string}(r_s))$. The decoding process is unique and shows that the transducer obtained from σ is $T_\sigma^{S_0} = T$. Secondly, if $x_s = \varepsilon$, then $\Delta(s, 0) = (s, \text{string}(r_s))$ for an odd s, and $\Delta(s, 1) = (s, \text{string}(r_s))$ for an even s. □

Given a string $\sigma \in S_0$, an explicit encoding of the transition function of $T_\sigma^{S_0}$ can be computed in quadratic time.

Example 2.1. Some simple examples are listed in Table 2. The first example is the smallest transducer; the last example is the identity transducer.

The encoding used in Theorem 2.1 is regular but not too compact as the pair $(i, \text{string}(j))$ is coded by $\text{bin}(i)^\dagger \cdot \text{string}(j)^\diamond$, a string of length $2(\text{Log(i)} + \text{Log(j)}) + 4$.

By using the encoding

$$x^\S = 0^{|\text{string}(|x|+1)|} \cdot 1 \cdot \text{string}(|x| + 1) \cdot x \tag{4}$$

we obtain a more compact one. Indeed, instead of Table 1 use the encoding in Table 3, where

$$\text{string}^\S(n) = 0^{|\text{string}(|\text{string}(n)|+1)|} \cdot 1 \cdot \text{string}(|\text{string}(n)| + 1) \cdot \text{string}(n),$$
$$\text{bin}^\#(n) = 1^{|\text{string}(|\text{string}(n)|+1)|} \cdot 0 \cdot \overline{\text{string}(|\text{string}(n)| + 1)} \cdot \text{string}(n),$$

and the pair $(i, \text{string}(j))$ is coded by $\text{bin}^\#(i+1) \cdot \text{string}^\S(j+1)$, a string of length $2 \cdot \text{Log}(\text{Log}(i+1)+1) + \text{Log}(i+1) + 2 \cdot \text{Log}(\text{Log}(j+1)+1) + \text{Log}(j+1) + 2 < 2(\text{Log(i)} + \text{Log(j)}) + 4$ almost everywhere.

By iterating the formula (4) we can indefinitely improve almost everywhere the encoding of the pairs $(i, \text{string}(j))$ obtaining more and more efficient variants of Theorem 2.1.

Theorem 2.2. *We can construct a sequence of computable sets $(S_n)_{n\geq 1}$ such that:* (a) *for every $\sigma \in S_n$, $T_\sigma^{S_n}$ is a transducer,* (b) *for every transducer T one can compute a code $\sigma \in S_n$ such that $T = T_\sigma^{S_n}$,* (c) *the difference in length between the encodings of the pair $(i, \text{string}(j))$ according to S_n and S_{n+1} tends to ∞ with n.*

3. Finite-state complexity

Transducers are used to "define" or "represent" strings in the following way. First we fix a computable set S as in Theorem 2.1 or Theorem 2.2. Then, we say that a pair (T_σ^S, p), $\sigma \in S$, $p \in \{0, 1\}^*$, defines the string x provided $T_\sigma^S(p) = x$; the pair (T_σ^S, p) is called a *description* of x. We define the size of the description (T_σ^S, p) of x by

$$||(T_\sigma^S, p)|| = |\sigma| + |p|.$$

Table 2
Transducers S_0 encodings.

Transducer	Code	Code length
$\Delta(1,0) = \Delta(1,1) = (1,\varepsilon)$	$\sigma = 0000$	4
$\Delta(1,0) = (1,\varepsilon), \Delta(1,1) = (1,0)$	$\sigma = 000110$	6
$\Delta(1,0) = (1,0), \Delta(1,1) = (1,\varepsilon)$	$\sigma = 011000$	6
$\Delta(1,0) = \Delta(1,1) = (1,0)$	$\sigma = 01100110$	8
$\Delta(1,0) = \Delta(1,1) = (1,1)$	$\sigma = 01000100$	8
$\Delta(1,0) = (1,0), \Delta(1,1) = (1,1)$	$\sigma = 01100100$	8

Table 3
S_1 encoding

n	bin(n)	bin$^{\#}(n)$	string(n)	string$^{\S}(n)$	length
1	1	0	ε	$0^0 1\varepsilon\varepsilon = 1$	1
2	10	1010	0	0100	4
3	11	1011	1	0101	4
4	100	10000	00	01100	5
5	101	10001	01	01101	5
6	110	10010	10	01110	5
7	111	10011	11	01111	5
8	1000	11011000	000	00100000	8

Based on the above, we define the *finite-state complexity* (with respect to the enumeration S) of a string $x \in \{0,1\}^*$ by the formula:

$$C_S(x) = \inf_{\sigma \in S,\ p \in \{0,1\}^*} \left\{ \mid \sigma \mid + \mid p \mid : T_\sigma^S(p) = x \right\}.$$

Comment 3.1. In the encoding S_0 a string v occurring as the output of a transition in T^{S_0} 'contributes' roughly $2 \cdot |v|$ to the size of an encoding $||(T^{S_0}, p)||$. With the encoding S_1 the contribution is $|v| + \mathrm{Log}(\mathrm{Log}(|v|) + 2$.

Comment 3.2. In analogy with AIT one could define the conditional finite-state complexity of a string x relative to a string p as the smallest encoding length of a transducer that on input p outputs x. We leave the study of conditional finite-state complexity to a future paper.

How "objective" is the above definition? First, finite-state complexity depends on the enumeration S; if S and S' are encodings then $C_{S'} = f(C_S)$, for some computable function f.

Secondly, finite-state complexity is defined as an analogue of the complexity used in AIT, whose objectivity is given by the Invariance Theorem, which in turn relies essentially on the Universality Theorem [7]. Using the existence of a universal (prefix-free) Turing machine one can obtain a complexity which is optimal up to an additive constant (the constant "encapsulates" the size of this universal machine). For this reason the complexity does not need to explicitly include the size of the universal machine. In sharp contrast, the finite-state complexity has to count the size of the transducer as part of the encoding length,[2] but can be more lax in working with the pair (σ, p). The reason for this is that there is no "universal" transducer; still, the Invariance Theorem holds true.

Thirdly, our proposal does not define just one finite-state complexity, but a class of "finite-state complexities" (depending on the underlying enumeration of transducers). At this stage we do not have a reasonable "invariance" result relating every pair of complexities in this class. In the theory of left-computable ε–randomness [8], the difference between two prefix complexities induced by different ε–universal prefix-free Turing machines can be arbitrarily large. In the same way here it is possible to construct two enumerations S', S'' satisfying Theorem 2.2 such that the difference between $C_{S'}$ and $C_{S''}$ is arbitrarily large.

Below we establish in a slightly more general way that no finite generalised transducer can simulate a transducer on a given input—not an unexpected result. For this we note the following two lemmas, and also that the pair (σ, w) can be uniquely encoded into the string $\sigma^\dagger w$.

Lemma 3.1 (*[4], Corollary 6.2*)**.** *Any rational relation can be realised by a transducer where the transitions are a subset of* $Q \times (X \cup \{\varepsilon\}) \times (Y \cup \{\varepsilon\}) \times Q$.

Lemma 3.2. *For any functional generalised transducer T there exists a constant M_T such that every prefix of an accepting computation of T that consumes input $x \in \{0,1\}^+$ produces an output of length at most $M_T \cdot |x|$.*

[2] One can use this approach also in AIT [20].

Proof. The statement follows from the observation that no functional generalised transducer can have a cycle where all transitions have input label ε. $\square$

Theorem 3.1. *Let S be an enumeration satisfying Theorem 2.1.*[3] *There is no functional generalised transducer U such that for all* $\sigma \in S$ *and* $w \in \{0,1\}^*$, $U(\sigma^\dagger w) = T^S_\sigma(w)$.

Proof. For the sake of contradiction assume that U exists and without loss of generality we assume that the transitions of U are in the normal form of Lemma 3.1. Let M_U be the corresponding constant given by Lemma 3.2.

Let $\sigma_i \in S$, $i \geq 1$, be the encoding of the single-state transducer where the two self-loops are labeled by $0/0^i$ and $1/\varepsilon$, i.e. $\Delta(1,0) = (1, 0^i)$, $\Delta(1,1) = (1,\varepsilon)$.

Define the function $g : \mathbb{N} \to \mathbb{N}$ by setting

$$g(i) = |\sigma_i^\dagger| \cdot M_U + 1, \quad i \geq 1.$$

Let D_i be an accepting computation of U that corresponds to the input ${\sigma_i}^\dagger \cdot 0^{g(i)}$, $i \geq 1$. Let q_i be the state of U that occurs in the computation D_i immediately after consuming the prefix ${\sigma_i}^\dagger$ of the input. Since U is in the normal form of Lemma 3.1, q_i is defined.

Choose $j < k$ such that $q_j = q_k$. We consider the computation D of U on input $\sigma_j^\dagger \cdot 0^{g(k)}$ that reads the prefix $\sigma_j^\dagger$ as D_j and the suffix $0^{g(k)}$ as D_k. Since $q_j = q_k$ this is a valid computation of U ending in an accepting state.

On prefix $\sigma_k^\dagger$ the computation D_k produces an output of length at most $M_U \cdot |\sigma_k^\dagger|$ and, hence, on the suffix $0^{g(k)}$ the computation D_k (and D) outputs 0^z where

$$z \geq k \cdot g(k) - |\sigma_k^\dagger| \cdot M_U (k-1) \cdot g(k).$$

The last inequality follows from the definition of the function g. Hence the output produced by the computation D is longer than $j \cdot g(k) = |T^S_{\sigma_j}(0^{g(k)})|$ and U does not simulate $T^S_{\sigma_j}$ correctly. $\square$

Conjecture 3.1. *No two-way finite transducer can simulate an arbitrary transducer* T^S_σ *when it receives* σ *as part of the input (in the sense of Theorem 3.1).*

For the rest of this section we fix an enumeration S satisfying Theorem 2.2.

Proposition 3.1. *For every string* $x, y \in \{0,1\}^*$ *there exist infinitely many transducers* T^S_σ *such that* $T^S_\sigma(x) = y$.

In spite of the negative result stated in Theorem 3.1 the Invariance Theorem from AIT is true for C. To this aim we define the complexity associated with a transducer T^S_σ by

$$C_{T^S_\sigma}(x) = \inf_{p \in \{0,1\}^*} \left\{ \,|\, p \,|\, : T^S_\sigma(p) = x \right\}.$$

Theorem 3.2 (*Invariance*). *For every* $\sigma_0 \in S$ *we have* $C_S(x) \leq C_{T^S_{\sigma_0}}(x) + |\sigma_0|$, *for all* $x \in \{0,1\}^*$.

Proof. Using the definitions of C_S and $C_{T^S_{\sigma_0}}$ we have:

$$\begin{aligned} C_S(x) &= \inf_{\sigma \in S,\ p \in \{0,1\}^*} \left\{ \|(T^S_\sigma, p)\| : T^S_\sigma(p) = x \right\} \\ &= \inf_{\sigma \in S,\ p \in \{0,1\}^*} \left\{ |\sigma| + |p| : T^S_\sigma(p) = x \right\} \\ &\leq |\sigma_0| + \inf_{p \in \{0,1\}^*} \left\{ |p| : T^S_{\sigma_0}(p) = x \right\} \\ &= C_{T^S_{\sigma_0}}(x) + |\sigma_0|. \quad \square \end{aligned}$$

Corollary 3.1. *If* $T^S_{\sigma_0}(x) = x$, *then* $C_S(x) \leq |x| + |\sigma_0|$, *for all* $x \in \{0,1\}^*$. *In particular, using Example 2.1 (last transducer) we deduce that* $C_{S_0}(x) \leq |x| + 8$, *for all* $x \in \{0,1\}^*$.

Comment 3.3. In view of Corollary 3.1 in the definitions of finite-state complexity we can replace inf by min.

Corollary 3.2. *The complexity* C_S *is computable.*

In Tables 4 and 5 we present a few initial values of C_{S_0} and C_{S_1}, respectively. "Complementary" strings are omitted. A plot containing the values of C_{S_0} and C_{S_1} appears in Fig. 1.

Conjecture 3.2. *The computational complexity of testing whether the finite-state complexity of a string x is less or equal to n is in NP; it is open whether this decision problem is NP-hard.* (See also *[3,15].)*

[3] We use a regular enumeration to avoid that the non-existence of a universal transducer is simply caused by the fact that a finite transducer cannot recognise legal encodings of transducers.

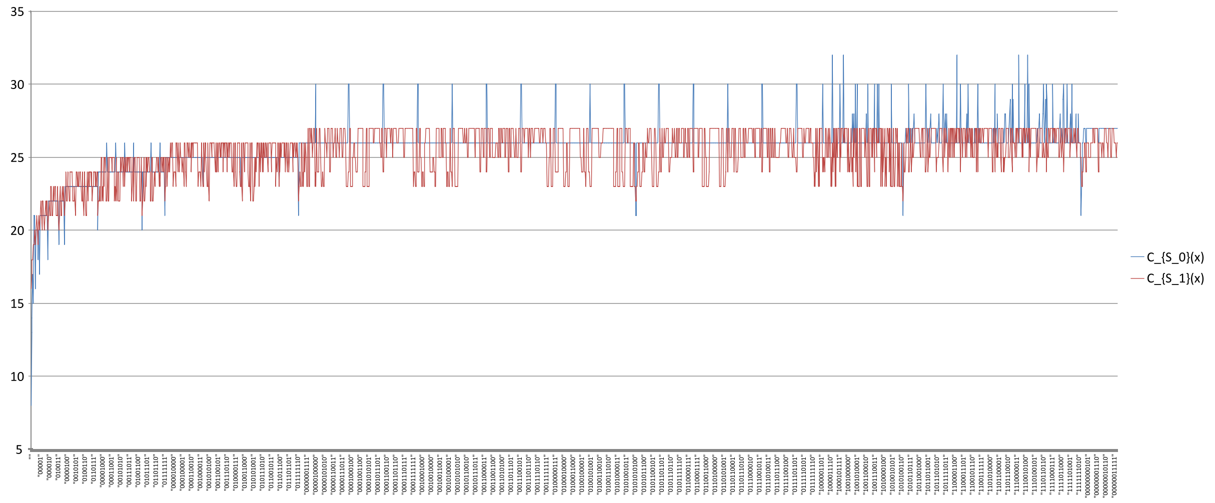


Fig. 1. Plot of comparative data in Tables 4 and 5

Table 4
Finite-state complexity (w.r.t. S_0) of all strings in lexicographic order from ε to 01111.

x	$C_{S_0}(x)$	(σ, p)	x	$C_{S_0}(x)$	(σ, p)
ε	4	(0000, ε)	00000	11	(000110, 11111)
0	7	(000110, 1)	00001	13	(01000110, 11110)
00	8	(000110, 11)	00010	13	(01000110, 11101)
01	9	(00011100, 1)	00011	13	(01000110, 11100)
000	9	(000110, 111)	00100	13	(01000110, 11011)
001	11	(01000110, 110)	00101	13	(01000110, 11010)
010	11	(01000110, 101)	00110	13	(01000110, 11001)
011	11	(01000110, 100)	00111	13	(01000110, 11000)
0000	10	(000110, 1111)	01000	13	(01000110, 10111)
0001	12	(01000110, 1110)	01001	13	(01000110, 10110)
0010	12	(01000110, 1101)	01010	13	(01000110, 10101)
0011	12	(01000110, 1100)	01011	13	(01000110, 10100)
0100	12	(01000110, 1011)	01100	13	(01000110, 10011)
0101	10	(00011100, 11)	01101	13	(01000110, 10010)
0110	12	(01000110, 1001)	01110	13	(01000110, 10001)
0111	12	(01000110, 1000)	01111	13	(01000110, 10000)

4. Quantitative estimates

Here we establish basic upper and lower bounds for the finite-state complexity of arbitrary strings, as well as for strings of particular types. For the rest of this section we use the enumeration S_0 and we write T_σ and C instead of $T_\sigma^{S_0}$ and C_{S_0}.

Theorem 4.1. *For $n \geq 1$ we have: $C(0^n) \in \Theta(\sqrt{n})$.*

Proof. It is sufficient to establish that

$$2 \cdot \lfloor\sqrt{n}\rfloor \leq C(0^n) \leq 4 \cdot \lfloor\sqrt{n}\rfloor + \alpha, \tag{5}$$

where α is a constant.

For the upper bound we note that 0^n can be represented by a pair (T, p) where T is a single state transducer having two self-loops labeled, respectively, $0/0^{\lfloor\sqrt{n}\rfloor}$ and $1/0$, and p can be chosen as a string $0^{\lfloor\sqrt{n}\rfloor+y}1^z$, where $0 \leq y \leq 1, 0 \leq z \leq \lfloor\sqrt{n}\rfloor$. By our encoding conventions the size of (T, p) is at most $4 \cdot \lfloor\sqrt{n}\rfloor + \alpha$ where α is a small constant.

To establish the lower bound, consider an arbitrary pair (T', p') representing 0^n. If v is the longest output of any transition of T', then $|v|\cdot|p'| \geq n$. On the other hand, according to our encoding conventions $||(T', p')|| \geq 2\cdot|v|+|p'|$. These inequalities imply $||(T', p')|| \geq 2 \cdot \lfloor\sqrt{n}\rfloor$. $\square$

Using a more detailed analysis, the upper and lower bounds of (5) could be moved closer to each other. Because the precise multiplicative constants depend on the particular enumeration S_0, it may not be very important to try to improve the values of the multiplicative constants.

Table 5
Finite-state complexity (w.r.t. S_1) of all strings in lexicographic order from ε to 01111.

x	$C_{S_1}(x)$	(σ, p)
ε	16	$(1010010010100100, \varepsilon)$
0	17	(1010010010100101, 1)
00	18	(1010010010100101, 11)
01	18	(10100100101001110, 1)
000	19	(1010010010100101, 111)
001	19	(10100101101001110, 01)
010	19	(10100101101001110, 10)
011	20	(10100101101001100, 011)
0000	19	(10100100101001101, 11)
0001	20	(10100101101001110, 001)
0010	20	(10100101101001110, 010)
0011	21	(10100101101001100, 0011)
0100	20	(10100101101001110, 100)
0101	19	(10100100101001110, 11)
0110	20	(101001110101001111, 01)
0111	21	(10100101101001100, 0111)
00000	20	(10100101101001101, 011)
00001	21	(10100101101001110, 0001)
00010	21	(10100101101001110, 0010)
00011	22	(10100101101001100, 00011)
00100	21	(10100101101001110, 0100)
00101	20	(10100101101001110, 011)
00110	22	(10100101101001100, 00110)
00111	22	(10100101101001100, 00111)
01000	21	(10100101101001110, 1000)
01001	20	(10100101101001110, 101)
01010	20	(10100101101001110, 110)
01011	21	(101001100101001110, 110)
01100	22	(10100101101001100, 01100)
01101	21	(101001100101001110, 101)
01110	22	(10100101101001100, 01110)
01111	22	(10100101101001100, 01111)

The argument used to establish the lower bound in (5) gives directly the following:

Corollary 4.1. *For any $x \in \{0, 1\}^*$, $C(x) \geq 2 \cdot \lfloor \sqrt{|x|} \rfloor$.*

Recall that H denotes the prefix-complexity in AIT [7]. The bounds (5) imply that the inequality $H(xx) \leq H(x) + O(1)$ from AIT does not hold for finite-state complexity:

Corollary 4.2. *There is no constant α such that for all strings $x \in \{0, 1\}^*$, $C(xx) \leq C(x) + \alpha$.*

The mapping $0^n \mapsto 0^{2 \cdot n}$ is computed by a transducer of small size. Hence we deduce:

Corollary 4.3. *For a given transducer T there is no constant α such that for all strings $x \in \{0, 1\}^*$, $C(T(x)) \leq C(x) + \alpha$.*

In Corollary 4.3 we require only that α is independent of x, that is, the value α could depend on the transducer T. As in Theorem 4.1 we get estimations for the finite-state complexity of powers of a string.

Proposition 4.1. *For $u \in \{0, 1\}^*$ and $n \gg |u|$,*

$$C(u^n) \leq 2 \cdot (\lfloor \sqrt{n} \rfloor + 1) \cdot |u| + 2\lfloor \sqrt{n} \rfloor + \alpha, \tag{6}$$

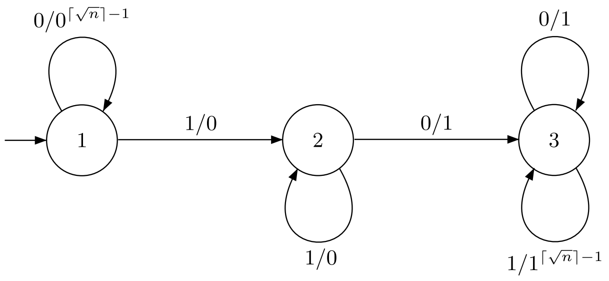
where α is a constant independent of u and n.

Proof. Let T be the single state transducer with two self-loops labeled, respectively, by $0/u^{\lfloor \sqrt{n} \rfloor}$ and $1/u$. The string u^n has a description $(T, 0^{\lfloor \sqrt{n} \rfloor + y} 1^z)$, where $0 \leq y \leq 1$, $0 \leq z \leq \lfloor \sqrt{n} \rfloor$. By our encoding conventions

$$||(T, 0^{\lfloor \sqrt{n} \rfloor} 1^z)|| = 2 \cdot (\lfloor \sqrt{n} \rfloor + 1) \cdot |u| + 4 + \lfloor \sqrt{n} \rceil + y + z.$$

Note that, when encoding self-loops, the state name is not part of the encoding and a self-loop with output string w contributes $2|w| + 2$ to the length of the encoding. The claim follows from the upper bounds for y and z. $\square$

The upper bound (6) is useful only when n is larger than $|u|^2$ because using a single state transducer with self-loop $0/u$ we get an upper bound $C(u^n) \leq 2 \cdot |u| + n + \alpha$, with α constant.



$0/0^{\lceil\sqrt{n}\rceil-1}$
1
$1/0$
2
$0/1$
3
$0/1$
$1/0$
$1/1^{\lceil\sqrt{n}\rceil-1}$

Fig. 2. Transducer T in the proof of Corollary 4.4.

Corollary 4.4. *We have:* $C(0^n1^n) \in \Theta(\sqrt{n})$.

Proof. The lower bound follows from Corollary 4.1. The string 0^n1^n has description

$$(T, 0^{\lceil\sqrt{n}\rceil-1+y_1}1^{z_1}0^{z_2}1^{\lceil\sqrt{n}\rceil-1+y_2}),$$

where $0 \le y_1, y_2 \le 1$, $1 \le z_1, z_2 \le \lceil\sqrt{n}\rceil$ and T is the transducer given in Fig. 2.

Note that unlike the construction used in Theorem 4.1, the transducer in Fig. 2 begins by outputting strings $0^{\lceil\sqrt{n}\rceil-1}$ (instead of $0^{\lfloor\sqrt{n}\rfloor}$). This is done in order to guarantee that z_1 can be chosen to be at least 1 also when n is a perfect square.

Thus, $C(0^n1^n) \le 8 \cdot \lceil\sqrt{n}\rceil + \alpha$, where α is a constant. □

Comment 4.1. From Corollary 4.4 we note that the finite-state complexity of 0^n1^n is within a constant factor of its automatic complexity, as defined in [22]. This can be viewed merely as a coincidence since the two descriptional complexity measures are essentially different and have, in general, very different upper and lower bounds. For example, the automatic complexity of 0^n is constant (independent of n), while, in line with AIT, the finite-state complexity of 0^n is not constant, as we shave shown to be $\Theta(\sqrt{n})$.

The following result gives an upper bound for finite-state complexity of the catenation of two strings.

Proposition 4.2. *For any* $\omega > 0$ *there exists* $d(\omega) > 0$ *such that for all* $x, y \in \{0,1\}^*$,

$$C(xy) \le (1+\omega)\cdot(4C(x) + C(y)) + d(\omega).$$

Here the value $d(\omega)$ *depends only on* ω*, i.e., it is independent of* x *and* y.

Proof. Let (T, u) and (R, v) be minimal descriptions of x and y, respectively. Let $u = u_1 \cdots u_m$, $u_i \in \{0,1\}$, $i = 1, \ldots, m$ and recall that $u^\dagger = u_10u_20\cdots u_{m-1}0u_m1$.

Denote the sets of states of T and R, respectively, as Q_T and Q_R, and let $Q'_T = \{q' \mid q \in Q_T\}$.

We construct a transducer W with set of states $Q_T \cup Q'_T \cup Q_R$ as follows.

(i) For each transition of T from state p to state q labeled by i/w ($i \in \{0,1\}$, $w \in \{0,1\}^*$), W has a transition from p to q' labeled by i/w and a transition labeled $0/\varepsilon$ from q' to q.
(ii) Each state $q' \in Q'_T$ has a transition labeled $1/\varepsilon$ to the starting state of R.
(iii) The transitions originating from states of Q_R are defined in W in the same way as in R.

Now $|u^\dagger| = 2\cdot|u|$ and

$$W(u^\dagger v) = T(u)R(v) = xy.$$

It remains to verify that the size of the encoding of W is, roughly, at most four times the size of T plus the size of R.

First assume that

(*) the states of W could have the same length encodings as the encodings used for states in T and R.

We note that the part of W simulating the computation of T has simply doubled the number of states and for the new states of Q'_T the outgoing transitions have edge labels of minimal length ($0/\varepsilon$ and $1/\varepsilon$). An additional increase in the length of the encoding occurs because each self-loop of T is replaced in W by two transitions that are not self-loops. It is easy to establish, using induction on the number of states of T, that if all states of T are reachable from the start state and T has t non-self-loop transitions, the number of self-loops in T is at most $t + 2$.

Thus, by the above observations with the assumption (*), $C(xy)$ could be upper bounded by $4C(x) + C(y) + d$ where d is a constant. Naturally, in reality the encodings of states of W need one or two additional bits added to the encodings of the corresponding states in T and R. The proportional increase of the state encoding length caused by the two additional bits for the states of $Q_T \cup Q'_T$, (respectively, states of Q_R) is upper bounded by $2\cdot(\lceil\log(|Q_T|)\rceil)^{-1}$ (respectively, $2\cdot(\lceil\log(|Q_R|)\rceil)^{-1}$). Thus, the proportional increase of the encoding length becomes smaller than any positive ω when $\max\{|Q_T|, |Q_R|\}$ is greater than a suitably chosen threshold $M(\omega)$. On the other hand, the encoding of W contains at most $2\cdot(2|Q_T| + |Q_R|) \le 6\cdot\max\{|Q_T|, |Q_R|\}$ occurrences of substrings encoding the states. This means that by choosing $d(\omega) = 12\cdot M(\omega)$ the statement of the lemma holds also for small values of $|Q_T|$ and $|Q_R|$. □

The proof of Proposition 4.2 relies on an estimation that the part of the transducer W simulating the computation of R has an encoding at most 4 times the size of the encoding of R. The additional increase is caused by the complication that each self-loop is simulated by two non-self-loops and the encoding of transitions that are not self-loops needs to include the state names. Using a more detailed analysis the constant 4 could likely be improved.

It is known that deterministic transducers are closed under composition [4], that is, for transducers T_δ and T_γ there exists $\sigma \in S$ such that $T_\sigma(x) = T_\delta(T_\gamma(x))$, for all $x \in \{0, 1\}^*$. Using the construction from ([4] Proposition 2.5, page 101, (translated into our notation)) we give an upper bound for $|\sigma|$ as a function of $|\delta|$ and $|\gamma|$.

Let $T_\delta = (Q, q_0, \Delta)$ and $T_\gamma = (P, p_0, \Gamma)$, where Δ is a function $Q \times \{0, 1\} \to Q \times \{0, 1\}^*$ and Γ is a function $P \times \{0, 1\} \to P \times \{0, 1\}^*$. The transition function Δ is extended in the natural way as a function $\hat{\Delta} : Q \times \{0, 1\}^* \to Q \times \{0, 1\}^*$.

The composition of T_γ and T_δ is computed by a transducer $T_\sigma = (Q \times P, (q_0, p_0), \Xi)$ where $\Xi : Q \times P \times \{0, 1\} \to Q \times P \times \{0, 1\}^*$ is defined by setting for $q \in Q, p \in P, a \in \{0, 1\}$,

$$\Xi\left((q, p), a\right) = \left(\left(\pi_1\left(\hat{\Delta}\left(q, \pi_2\left(\Gamma\left(p, a\right)\right)\right)\right), \pi_1\left(\Gamma\left(p, a\right)\right)\right), \pi_2\left(\hat{\Delta}\left(q, \pi_2\left(\Gamma\left(p, a\right)\right)\right)\right)\right).$$

The number of states of T_σ is upper bounded by $|\delta| \cdot |\gamma|$.[4] An individual output of T_σ consists of the output produced by T_δ when it reads an output produced by one transition of T_γ (via the extended function $\hat{\Delta}$). Thus, the length of the output produced by an individual transition of T_σ can be upper bounded by $|\delta| \cdot |\gamma|$. These observations imply that

$$|\sigma| = O(|\delta|^2 \cdot |\gamma|^2).$$

The above estimate was obtained simply by combining the worst-case upper bound for the size of the encoding of the states of T_σ and the worst-case length of individual outputs of the transducers T_δ and T_γ. The worst-case examples for these two bounds are naturally very different, as the latter corresponds to a situation where the encoding of individual outputs 'contributes' a large part of the strings δ and γ. The overall upper bound could be somewhat improved using a more detailed analysis.

Comment 4.2. Intuitively, the following property would probably seem natural or desirable. If u is a prefix of v, then $C(u) \leq C(v) + \alpha$ where α is a constant independent of u and v. However, the finite-state complexity of v does not, at least not directly, give an upper bound for the finite-state complexity of prefixes of v because the minimal description of v may involve a transducer where an individual transition t produces a very long output, and there seems no straightforward way to replace a prefix of the last application of t without increasing the length of the encoding by a nonconstant amount.

Open problem 4.1. *Obtain a reasonable upper bound for $C(u)$ in terms of $C(v)$ when u is a prefix of v.*

5. Incompressibility and lower bounds

As in the case of incompressibility in AIT we define a string x to be *finite-state i–compressible* ($i \geq 1$) if $C(x) \leq |x| - i$. A string x is *finite-state i–incompressible* if $C(x) > |x| - i$; if $i = 1$, then the string is called *finite-state incompressible*.

Lemma 5.1. *There exist finite-state incompressible strings of any length.*

Lemma 5.1 relies on a standard counting argument and does not give a construction of incompressible strings. By relying on results on grammar-based compression we can get lower bounds for finite-state complexity of explicitly constructed strings.

A grammar G, or straight-line program [12,16,18], used as an encoding of a string has a unique production for each nonterminal and the grammar is acyclic. That is, there is an ordering of the nonterminals $X_1, \ldots, X_m$ such that the productions are of the form $X_1 \to \alpha_1, \ldots, X_m \to \alpha_m$, where α_i contains only nonterminals from $\{X_{i+1}, \ldots, X_m\}$ and terminal symbols. The *size of the grammar* G, $\text{size}(G)$, is defined to be $\sum_{i=1}^{m} |\alpha_i|$.

Grammar-based compression of a string x may result in exponential savings compared to the length of x. Comparing this to Corollary 4.1, we note that the size of the smallest grammar generating a given string may be exponentially smaller than the finite-state complexity of the string. Conversely, any string x can be generated by a grammar with size $O(C(x))$.

Lemma 5.2. *There exists a constant $d \geq 1$ such that for any $x \in \{0, 1\}^*$, $\{x\}$ is generated by a grammar G_x where $\text{size}(G_x) \leq d \cdot C(x)$.*

Proof. The construction outlined in [10] for simulating an "NFA with advice" by a grammar is similar. For the sake of completeness we include here a construction.

Assume x is encoded as a transducer-string pair (T_σ, p), where $p = p_1 \cdots p_n$, $p_i \in \{0, 1\}$. The initial nonterminal of the grammar G_x has a production with right side $(p_1, s_{i_1})(p_2, s_{i_2}) \cdots (p_n, s_{i_n})$ where s_{i_j} is the state of T_σ reached by the transducer after consuming the input string $p_1 \cdots p_{j-1}$, $1 \leq j \leq n$. After this the rules for nonterminals (p_i, s) simply simulate the output produced by T_σ in state s on input p_i.

4 Strictly speaking, this could be multiplied by $\frac{(\log\log|\delta|)\cdot(\log\log|\gamma|)}{\log|\delta|\cdot\log|\gamma|}$ to give a better estimate.

Let Q be the set of states of T_σ and, as usual, denote the set of transitions by $\Delta : Q \times \{0,1\} \to Q \times \{0,1\}^*$. The size of G_x, that is the sum of the lengths of right sides of the productions of G_x, is

$$\mathrm{size}(G_x) = |p| + \sum_{q\in Q, i\in\{0,1\}} |\pi_2(\Delta(q,i))|. \quad \square$$

Note that $\mathrm{size}(G_x)$ is defined simply as the sum of lengths of productions of G_x while $C(x)$ uses a binary encoding of a transducer T. In cases where minimal representations use transducers with fixed length outputs for individual transitions and large numbers of states, $\mathrm{size}(G_x)$ is less than a constant times $C(x) \cdot (\log C(x))^{-1}$.

A binary *de Bruijn word* of order $r \geq 1$ is a string w of length $2^r + r - 1$ over alphabet $\{0, 1\}$ such that any binary string of length r occurs as a substring of w (exactly once). It is well known that de Bruijn words of any order exist, and have an explicit construction [11,23].

Theorem 5.1. *There is a constant d such that for any $r \geq 1$ there exist strings w of length $2^r + r - 1$ with an explicit construction such that $C(w) \geq d \cdot |w| \cdot (\log(|w|))^{-1}$.*

Proof. It is known that any grammar generating a de Bruijn string of order r has size $\Omega(\frac{2^r}{r})$ [2]. Grammars generating a singleton language are called string chains in [2], see also [12]. The claim follows by Lemma 5.2. $\square$

Conjecture 5.1. *De Bruijn words are finite-state incompressible.*

6. Conclusion

In this paper we have developed the first steps of a variant of AIT based on finite transducers. The finite-state complexity, central to the new theory, is computable and satisfies a strong form of Invariance Theorem. In contrast with descriptional complexities from AIT, there is no *a priori* upper bound for the number of states used for minimal descriptions of arbitrary strings.

We have studied finite-state complexity from a theoretical point of view. In some sense the compression capabilities of finite transducers are weak, as indicated by the fact that the complexity of unary strings of length n is $\Theta(\sqrt{n})$. It remains to be seen whether this complexity measure could be useful in applications, as has been the case for grammar based compression.

Some open questions have been discussed throughout the paper. There are many possible continuations; for example, it will be interesting to check whether finite-state random strings are Borel normal [7]. A natural extension of the work could consider an analogous descriptional complexity measure based on two-way finite transducers and its relationship to the current measure.

Acknowledgements

C. Calude was supported by FRDF Grant of the UoA. Part of this research was done during the visit of K. Salomaa to CDMTCS in 2009. K. Salomaa was supported by an NSERC Research Grant. Thanks also go to R. Downey for his comments and the anonymous referees for constructive criticism.

References

[1] V.N. Agafonov, Normal sequences and finite automata, Sov. Math. Dokl. 9 (1968) 324–325.
[2] I. Althöfer, Tight lower bounds on the length of word chains, Inform. Proc. Lett. 34 (1990) 275–276.
[3] J. Arpe, R. Reischuk, On the complexity of optimal grammar-based compression, in: Proc. of the Data Compression Conference, DCC'06, 2006, pp. 173–186.
[4] J. Berstel, Transductions and Context-Free Languages, Teubner, 1979.
[5] C. Bourke, J.M. Hitchcock, N.V. Vinodchandran, Entropy rates and finite-state dimension, Theoret. Comput. Sci. 349 (3) (2005) 392–406.
[6] H. Buhrman, L. Fortnow, Resource-bounded Kolmogorov complexity revisited, in: Proceedings STACS'97, in: Lect. Notes Comput. Sci., vol. 1200, Springer, 1997, pp. 105–116.
[7] C.S. Calude, Information and Randomness—An Algorithmic Perspective, 2nd ed., Springer, Berlin, 2002.
[8] C.S. Calude, N.J. Hay, F.C. Stephan, Representation of left-computable ε-random reals, J. Comput. Syst. Sci. 77 (2011) 812–819.
[9] G. Chaitin, Algorithmic Information Theory, Cambridge University Press, 1987.
[10] M. Charikar, E. Lehman, D. Liu, R. Panigrahy, M. Prabhakaran, A. Rasala, A. Sahai, A. Shelat, Approximating the smallest grammar: Kolmogorov complexity in natural models, in: Proceedings of STOC'02, ACM Press, 2002, pp. 792–801.
[11] N. de Bruijn, A combinatorial problem, Proc. K. Ned. Akad. Wet. 49 (1946) 758–764.
[12] M. Domaratzki, G. Pighizzini, J. Shallit, Simulating finite automata with context-free grammars, Inform. Process. Lett. 84 (2002) 339–344.
[13] D. Doty, P. Moser, Feasible depth, arXiv:cs/0701123v3, 2007.
[14] R. Downey, D. Hirschfeldt, Algorithmic Randomness and Complexity, Springer, Heidelberg, 2010.
[15] E. Lehman, Approximation Algorithms for Grammar-based Compression, Ph.D. Thesis, MIT, 2002.
[16] E. Lehman, A. Shelat, Approximation algorithms for grammar-based compression, in: SODA'02, SIAM Press, 2002, pp. 205–212.
[17] A. Nies, Computability and Randomness, Clarendon Press, Oxford, 2009.
[18] W. Rytter, Application of Lempel-Ziv factorization to the approximation of grammar-based compression, Theoret. Comput. Sci. 302 (2002) 211–222.
[19] C.P. Schnorr, H. Stimm, Endliche Automaten und Zufallsfolgen, Acta Inform. 1 (1972) 345–359.
[20] M.L. Sipser, Introduction to the Theory of Computation, PWS, 1997.
[21] J. Shallit, Y. Breitbart, Automacity I: properties of a measure of descriptional complexity, J. Comput. Syst. Sci. 53 (1996) 10–25.
[22] J. Shallit, M.-W. Wang, Automatic complexity of strings, J. Autom. Lang. Comb. 6 (2001) 537–554.
[23] J.H. van Lint, R.M. Wilson, A Course in Combinatorics, Cambridge University Press, 1993.
[24] J. Ziv, A. Lempel, Compression of individual sequences via variable-rate coding, IEEE Trans. Inform. Theory 24 (1978) 530–536.