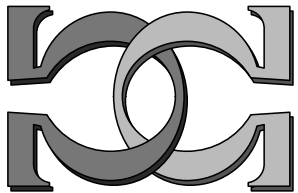
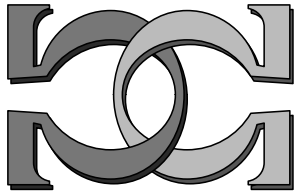
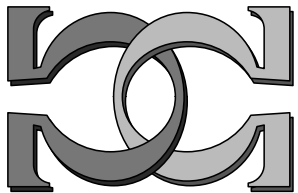


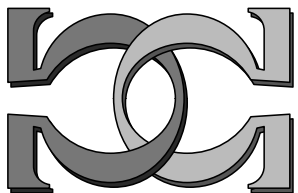
**CDMTCS  
Research  
Report  
Series**



**Finite-State Complexity and  
Randomness**

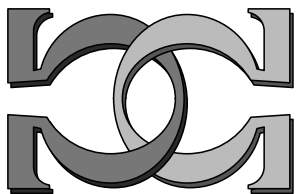


**Cristian S. Calude<sup>1</sup>, Kai Salomaa<sup>2</sup>,  
Tania K. Roblot<sup>1</sup>**



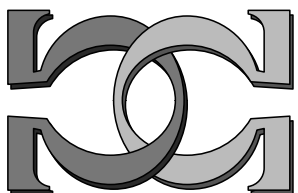
<sup>1</sup>University of Auckland, NZ

<sup>2</sup>Queen's University, Ontario, Canada



CDMTCS-374

December 2009/revised June 2010



Centre for Discrete Mathematics and  
Theoretical Computer Science

# Finite-State Complexity and Randomness

Cristian S. Calude<sup>1</sup>, Kai Salomaa<sup>2</sup>, Tania K. Roblot<sup>1</sup>

<sup>1</sup>Department of Computer Science  
University of Auckland, New Zealand

<sup>2</sup>School of Computing  
Queen's University, Kingston, Ontario, Canada

May 31, 2010

## Abstract

In this paper we develop a version of Algorithmic Information Theory (AIT) by replacing Turing machines with finite transducers; 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 descriptive 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. Explicit examples of finite-state incompressible strings are given as well as upper and lower bounds for finite-state complexity of arbitrary strings and for strings of particular types. Various open problems are discussed throughout the paper.

## 1 Introduction

Algorithmic Information Theory [7, 5] uses various measures of descriptive complexity to define and study various classes of “algorithmically random” finite strings or infinite sequences. The 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, 11].

The incomputability of all descriptive 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 descriptive complexity [4]. 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 descriptonal complexity of  $x$ . This model has been investigated since the 70's which recently received much attention [8, 10, 16, 15, 19] (also because of connections with Lempel-Ziv encodings [15, 19]). Further restricting the computational power, from context-free grammars to finite automata (DFA), one obtains automatic complexity [24]. Since a DFA recognising the singleton language  $\{x\}$  needs always  $|x| + 1$  states, the automatic complexity of a string  $x$  is defined as the smallest number of states of a DFA that accepts  $x$  and does not accept any other string of length  $|x|$ . A similar descriptonal complexity measure for languages was considered in [23].

Computations with finite transducers are used in [13] for the definition of finite-state dimension of infinite sequences. The NFA-complexity of a string [8] can be defined in terms of finite transducers that are called in [8] "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 string  $x$  in terms of an enumeration of finite transducers and 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 9). To overcome this negative result we show that the set of finite transducers can be enumerated by a computable (even a regular) set (Theorem 2, Theorem 6), and, based on it, we prove the Invariance Theorem (Theorem 13) for finite-state complexity. Finally, we show that finite-state complexity is computable and we present an algorithm for computing it. The complexity of testing whether the finite-state complexity of a string  $x$  is less or equal to  $n$  is NP; it is open whether this decision problem is NP-complete.

Our notation is standard [3, 5]. If  $X$  is a finite set then  $X^*$  is the set of all strings (words) over  $X$  with  $\varepsilon$  denoting the empty string. The length of  $x \in X^*$  is denoted by  $|x|$ . A prefix-free set  $S \subset X^*$  is a set with the property that for all strings  $p, q \in X^*$ , if  $p, pq \in S$  then  $p = pq$ . By  $H$  we denote the prefix-complexity.

The paper is organised in the following way. In the next section we present the basic facts about finite transducers. In section 3 we present examples of useful enumerations of finite transducers, a regular set and a sequence of computable sets of binary strings  $S$  enumerating all finite transducers. In section 4 we define and study the finite-state complexity of a (binary) string with respect to an arbitrary enumeration  $S$ . A string  $x$  is *represented* by a pair  $(T_\sigma^S, p)$  where  $T_\sigma^S$  is the  $\sigma$ 's finite transducer enumerated in  $S$  and  $p$  is a binary string with  $T_\sigma^S(p) = x$ . Pairs representing strings are enumerated themselves by a computable (even regular, in some cases) set and the *finite-state complexity* of a string  $x$  is defined by the "length" of the smallest pair (in that enumeration) representing  $x$ . The remainder part of the paper uses the regular enumeration  $S_0$ . In section 5 we establish some basic upper and lower bounds for finite-state complexity of arbitrary strings, as well as, for strings of

particular types. In section 6 we show that finite-state complexity is a rich complexity measure also with respect to the number of states of the transducers, in the sense that—in sharp contrast with AIT, where the number of states of a universal Turing machine can be fixed—there is no *a priori* upper bound for the number of states used for minimal descriptions of arbitrary strings. In section 7 we discuss finite-state incompressibility and in section 8 we present an algorithm for computing finite-state complexity; we give examples of finite-state incompressible strings. In section 9 we use lower bounds developed for grammar-based compression to obtain non-trivial bounds for finite-state complexity of explicitly constructed strings. Various open problems are discussed throughout the paper. We conclude the paper with a short summary of results.

## 2 Finite transducers

A generalised finite transducer [3] is a tuple

$$T = (X, Y, Q, q_0, Q_F, E), \quad (1)$$

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  $u/v$ .

A transducer  $T$  realises the transduction  $\tau_T \subseteq X^* \times Y^*$  that consists of all pairs of strings  $(x, y)$ ,  $x \in X^*$ ,  $y \in Y^*$ , such that the start state  $q_0$  is connected to an accepting state  $q_f \in Q_F$  by a sequence of transitions  $e_1, \dots, e_k$  and  $x$  (respectively,  $y$ ) is the concatenation of input labels (respectively, output labels) of  $e_1, \dots, e_k$ . For a more formal definition of the transduction realised by  $T$  we refer the reader to [3]. It is well-known that finite transducers realise exactly the rational relations:

**Lemma 1** ([3], Corollary 6.2) *Any rational relation can be realised by a transducer where the transitions are a subset of  $Q \times (X \cup \{\varepsilon\}) \times (Y \cup \{\varepsilon\}) \times Q$ .*

A generalised transducer  $T$  is said to be *functional* if  $\tau_T$  is a partial function  $X^* \rightarrow Y^*$ . If  $T$  is functional, we denote by  $T(x)$  the string produced by  $T$  when it receives  $x$  as input ( $T(x)$  may be undefined).

Unless otherwise mentioned, we always consider a binary input and output alphabet  $X = Y = \{0, 1\}$  and denote a transducer simply as a four-tuple  $T = (Q, q_0, Q_F, E)$ .

For our finite-state complexity model we use a restricted type of functional transducers where the corresponding transduction can be computed deterministically. A transducer  $T$  is said to be a *deterministic sequential transducer* [3] if it has no transitions with input label  $\varepsilon$  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 represented by a function

$$\Delta : Q \times \{0, 1\} \rightarrow Q \times \{0, 1\}^*. \quad (2)$$

As we use mainly deterministic sequential transducers, we drop the adjectives, i.e., in the following by a *transducer* we mean a deterministic sequential transducer.

When using the general model (1) we refer to them as *generalised transducers*.

For a transducer all states are considered to be final. Hence a transducer can be given by a triple  $(Q, q_0, \Delta)$  where  $\Delta$  is as in (2). In details, the function  $T : \{0, 1\}^* \rightarrow \{0, 1\}^*$  computed by the transducer  $(Q, q_0, \Delta)$  is defined by  $T(\varepsilon) = \varepsilon$ ,  $T(xa) = T(x) \cdot \mu(\hat{\delta}(q_0, x), a)$ , where  $\delta(q, x) = \pi_1(\Delta(q, x))$ ,  $\mu(q, x) = \pi_2(\Delta(q, x))$ ,  $q \in Q$ ,  $x \in \{0, 1\}^*$ ,  $a \in \{0, 1\}$ .<sup>1</sup> Here  $\hat{\delta} : Q \times \{0, 1\}^* \rightarrow Q$  is defined by  $\hat{\delta}(q, \varepsilon) = q$ ,  $\hat{\delta}(q, xa) = \delta(\hat{\delta}(q, x), a)$ ,  $q \in Q$ ,  $x \in \{0, 1\}^*$ .

### 3 Regular enumerations of transducers

We use binary strings to encode transducers and prove that the set of all legal encodings of transducers is a regular language. We encode a transducer by listing for each state  $q$  and input symbol  $i \in \{0, 1\}$  the output and target state corresponding to the pair  $(q, i)$ , that is,  $\Delta(q, i)$ . Thus, the encoding of a transducer is a list of (encodings of) states and output strings. Furthermore, to minimise the length of the encoding in the list we omit states that correspond to self-loops. We verify that every string in the regular language of legal encodings corresponds to a unique transducer and, conversely, every transducer has a legal encoding.

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

For  $v = v_1 \cdots v_m$ ,  $v_i \in \{0, 1\}$ ,  $i = 1, \dots, m$ , we use the following functions producing self-delimiting versions of their inputs (see [5]):  $v^\dagger = v_1 0 v_2 0 \cdots v_{m-1} 0 v_m 1$  and  $v^\diamond = (1v)^\dagger$ , where  $\bar{\phantom{x}}$  is the negation morphism given by  $\bar{0} = 1, \bar{1} = 0$ . It is seen that  $|v^\dagger| = 2|v|$ , and  $|v^\diamond| = 2|v| + 2$ .

In the next table we will present the encodings of the first binary strings:

---

<sup>1</sup>Sometimes we use  $\cdot$  to denote the concatenation of strings;  $\pi_1$  and  $\pi_2$  are the first two projections on  $Q \times \{0, 1\}^*$ .

$n$	$\text{bin}(n)$	$\text{bin}(n)^\dagger$	$\text{string}(n)$	$\text{string}(n)^\diamond$	$ \text{bin}(n)^\dagger  =  \text{string}(n)^\diamond $
1	1	11	$\varepsilon$	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

Table 1:  $S_0$  encoding

Consider a transducer  $T$  with the set of states  $Q = \{1, \dots, n\}$ . The transition function  $\Delta$  of  $T$  (as in (2)) is encoded by a binary string

$$\sigma = \text{bin}(i_1)^\dagger \cdot \text{string}(i'_1)^\diamond \cdot \text{bin}(i_2)^\dagger \cdot \text{string}(i'_2)^\diamond \cdots \text{bin}(i_{2n})^\dagger \cdot \text{string}(i'_{2n})^\diamond, \quad (3)$$

where  $\Delta(j, k) = (\widehat{i_{2j-1+k}}, \text{string}(i'_{2j-1+k}))$ ,  $i_t, i'_t \geq 1$ ,  $t = 1, \dots, 2n$ ,  $j = 1, \dots, n$ ,  $k \in \{0, 1\}$ , and  $\widehat{m}$  denotes the smallest positive integer congruent to  $m$  modulo  $n$ .<sup>2</sup> In (3),  $\text{bin}(i_t)^\dagger = \varepsilon$  if the corresponding transition of  $\Delta$  is a self-loop, i.e.  $\pi_1(\Delta(j, k)) = j$ ; otherwise,  $\text{bin}(i_t)^\dagger = \text{bin}(i_t)^\dagger$ .

The transducer  $T$  encoded by  $\sigma$  is called  $T_\sigma^S$ . Here  $S$  refers to the particular set of encodings based on (3) used in the proof of the below theorem and throughout the whole paper.

**Theorem 2** *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  $T$  on can compute a code  $\sigma \in S_0$  such that  $T = T_\sigma^{S_0}$ .*

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

$$S_0 = (((X \cup \{\varepsilon\})Y)^2)^*. \quad (4)$$

The languages  $X$  and  $Y$  are regular, hence  $S_0$  is regular by (4).

The claim a) follows from (3). 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  $x_1, \dots, x_{2n} \in X \cup \{\varepsilon\}$  and  $y_1, \dots, y_{2n} \in Y$ . So, from  $\sigma$  we uniquely get the length  $n$  and the codes  $x_s \cdot y_s$ , for  $s = 1, 2, \dots, 2n$ . Every  $x_s$  can be uniquely written in the form  $x_s = \text{bin}(i_s)^\dagger$  and every  $y_s$  can be uniquely written in the form  $y_s = \text{string}(r_s)^\diamond$ .

<sup>2</sup>In (3) we use  $i_t$  instead of  $\widehat{i_t}$  in order to guarantee that the set of legal encodings of all transducers is regular, cf. Theorem 2.

Next we compute the unique transition encoded by  $x_s \cdot y_s$  according to (3). We denote by  $m \bmod n$  the smallest positive integer congruent to  $m$  modulo  $n$ . 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  $\sigma$  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.

The simplest transducer  $T$  has one state and produces always the empty string:

**Example 3** Let  $\Delta: \{1\} \times \{0, 1\} \rightarrow \{1\} \times \{0, 1\}^*$  be defined by  $\Delta(1, 0) = \Delta(1, 1) = (1, \varepsilon)$ . Its code is  $\sigma = \text{bin}(1)^\ddagger \cdot \varepsilon^\diamond \cdot \text{bin}(1)^\ddagger \cdot \varepsilon^\diamond = 0000$ . ■

**Example 4** A few more simple examples follow.

transducer	code	code length
$\Delta_1(1, 0) = (1, \varepsilon), \Delta_1(1, 1) = (1, 0)$	$\sigma = \text{bin}(1)^\ddagger \cdot \varepsilon^\diamond \cdot \text{bin}(1)^\ddagger \cdot 0^\diamond = 000110$	6
$\Delta_2(1, 0) = (1, 0), \Delta_2(1, 1) = (1, \varepsilon)$	$\sigma = \text{bin}(1)^\ddagger \cdot 0^\diamond \cdot \text{bin}(1)^\ddagger \cdot \varepsilon^\diamond = 011000$	6
$\Delta_3(1, 0) = \Delta_3(1, 1) = (1, 0)$	$\sigma = \text{bin}(1)^\ddagger \cdot 0^\diamond \cdot \text{bin}(1)^\ddagger \cdot 0^\diamond = 01100110$	8
$\Delta_4(1, 0) = \Delta_4(1, 1) = (1, 1)$	$\sigma = \text{bin}(1)^\ddagger \cdot 1^\diamond \cdot \text{bin}(1)^\ddagger \cdot 1^\diamond = 01000100$	8

Table 2: Transducers  $S_0$  encodings ■

**Example 5** The identity transducer  $T$  is given by  $\Delta(1, 0) = (1, 0), \Delta(1, 1) = (1, 1)$ . Its code is  $\sigma = \text{bin}(1)^\ddagger \cdot 0^\diamond \cdot \text{bin}(1)^\ddagger \cdot 1^\diamond = \varepsilon \cdot 0^\diamond \cdot \varepsilon \cdot 1^\diamond = 01100100$ . ■

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

$$x^\S = 0^{|\text{string}(|x|+1)|} \cdot 1 \cdot \text{string}(|x| + 1) \cdot x \quad (5)$$

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  $2 \cdot \text{Log}(\text{Log}(i+1) + 1) + \text{Log}(i+1) + 2 \cdot \text{Log}(\text{Log}(j+1) + 1) + \text{Log}(j+1) + 2$ , which is smaller almost everywhere than  $2(\text{Log}(i) + \text{Log}(j)) + 4$ .

By iterating the formula (5) we can indefinitely improve almost everywhere the encoding of the pairs  $(i, \text{string}(j))$  obtaining more and more efficient variants of Theorem 2.

$n$	$\text{bin}(n)$	$\text{bin}^\#(n)$	$\text{string}(n)$	$\text{string}^{\mathfrak{s}}(n)$	length
1	1	0	$\varepsilon$	$0^0 1 \varepsilon \varepsilon = 1$	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

Table 3:  $S_1$  encoding

**Theorem 6** 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  $T$  on 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  $\infty$  with  $n$ .

## 4 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 or Theorem 6. Then, we say that a pair  $(T_\sigma^S, p)$ ,  $\sigma \in S, p \in \{0, 1\}^*$ , defines the string  $x$  provided  $T_\sigma^S(p) = x$ ; the pair  $(T_\sigma^S, p)$  is called a *description* of  $x$ . As the pair  $(T_\sigma^S, p)$  is uniquely represented by the pair  $(\sigma, p)$  we define the size of the description  $(T_\sigma^S, p)$  of  $x$  by

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

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:

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

**Comment 7** 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  $|v| + \text{Log}(\text{Log}(|v|)) + 2$ .

How ‘objective’ is the above definition? First, finite-state complexity depends on the enumeration  $S$ ; if  $S$  and  $S'$  are encodings then  $C_{S'} = f(C_S(x))$ , 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 [5]. 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,<sup>3</sup> but can be more lax in working with the pair  $(\sigma, p)$ . The reason is that there is no “universal” transducer.

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  $\varepsilon$ -randomness [6], the difference between two prefix complexities induced by different  $\varepsilon$ -universal prefix-free Turing machines can be arbitrarily large. In the same way here it is possible to construct two enumerations  $S_1, S_2$  satisfying Theorem 2 such that the difference between  $C_{S_1}$  and  $C_{S_2}$  is arbitrarily large.

Below we establish, slightly more generally, that no finite generalised transducer can simulate a transducer on a given input—not an unexpected result.

We start with the following lemma that follows from the observation that a functional generalised transducer cannot have a loop where all transitions have input label  $\varepsilon$ .

**Lemma 8** *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  $M_T \cdot |x|$ .*

The pair  $(\sigma, w)$  can be uniquely encoded into the string  $\sigma^\dagger w$ .

**Theorem 9** *There is no functional generalised transducer  $U$  such that for all  $\sigma \in S_0$  and  $w \in \{0, 1\}^*$ ,  $U(\sigma^\dagger w) = T_\sigma^{S_0}(w)$ .*

**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 1. Let  $M_U$  be the corresponding constant given by Lemma 8.

Let  $\sigma_i \in S_0$ ,  $i \geq 1$ , 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  $g : \mathbb{N} \rightarrow \mathbb{N}$  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 1,  $q_i$  is defined.

---

<sup>3</sup>One can use this approach also in AIT [20].

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  $j \cdot g(k) = |T_{\sigma_j}^{S_0}(0^{g(k)})|$  and  $U$  does not simulate  $T_{\sigma_j}^{S_0}$  correctly. ■

**Comment 10** The hypothetical generalised transducer in Theorem 9 receives the input transducer encoded using the standard regular enumeration  $S_0$  (as in 4); the regularity of  $S_0$  eliminates the possibility that the negative result is artificially derived from the complex enumeration of transducers. However, the proof does not use any properties of this particular encoding and, in fact, the result holds for any encoding of the input transducers.

**Conjecture 11** *No two-way finite transducer can simulate an arbitrary transducer  $T_\sigma$  when it receives  $\sigma$  as part of the input, i.e. there does not exist a universal two-way finite transducer in the sense of Theorem 9.*

Obviously  $T_\sigma(w)$  can be computed from input  $\sigma^\dagger w$  by a two-way transducer that can use a logarithmic amount of read/write memory, or by a read-only multi-head finite transducer with two two-way heads and one one-way head.

A weak form of universality can be proven for transducers:

**Proposition 12** *For every strings  $x, y \in \{0, 1\}^*$  there exist infinitely many transducers  $T_\sigma$  such that  $T_\sigma(x) = y$ .*

**Proof.** Given  $x, y$  with  $x = x_1 x_2 \cdots x_n$  of length  $n$  we construct the transducer  $\Delta$  having  $n+1$  states acting as follows:  $\Delta(i, \bar{x}_i) = (n+1, \varepsilon)$ ,  $1 \leq i \leq n$ ,  $\Delta(1, x_1) = (2, y)$ ,  $\Delta(j, x_j) = (j+1, \varepsilon)$ ,  $2 \leq j \leq n$ ,  $\Delta(n+1, 0) = \Delta(n+1, 1) = (n+1, \varepsilon)$ . ■

Let  $\mathcal{T}_{\text{DGSM}}$  denote the set of all transducers. By a *computable encoding* of all transducers we mean a pair  $S = (D_S, f_S)$  where  $D_S \subseteq \{0, 1\}^*$  is a decidable set and  $f_S : D_S \rightarrow \mathcal{T}_{\text{DGSM}}$  is a computable onto mapping that associates to each  $\sigma \in D_S$  a transducer  $T_\sigma^S$ . Theorem 2 or Theorem 6 give examples of computable encodings of all transducers.

For the rest of the paper, unless explicitly stated, we fix a computable encoding of all transducers  $S$  and we write  $T_\sigma$  and  $C$  instead of  $T_\sigma^S$  and  $C_S$ .

In spite of the negative result stated in Theorem 9 the Invariance Theorem from AIT is true for  $C$ . To this aim we define the complexity associated with a transducer  $T_\sigma$  by

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

**Theorem 13** *For every  $\sigma_0 \in S$  we have  $C(x) \leq C_{T_{\sigma_0}}(x) + |\sigma_0|$ , for all  $x \in \{0,1\}^*$ .*

**Proof.** Using the definitions of  $C$  and  $C_{T_{\sigma_0}}$  we have:

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

■

**Corollary 14** *If  $T_{\sigma_0}(x) = x$ , then  $C(x) \leq |x| + |\sigma_0|$ , for all  $x \in \{0,1\}^*$ . In particular, using Example 5 we deduce that  $C(x) \leq |x| + 8$ , for all  $x \in \{0,1\}^*$ .*

**Comment 15** In view of Corollary 14, in the definitions of finite-state complexity we can replace inf by min.

In contrast with the incomputability results of AIT, by Corollary 14 and the fact that the output of a transducer corresponding to a given input is easily determined, we get:

**Corollary 16** *The complexity  $C$  is computable.*

Obviously, the problem of deciding whether  $C(x) \leq m$  for given  $x \in \{0,1\}^*$ ,  $m \in \mathbb{N}$  is in NP. It may be difficult to establish an NP-hardness result since even the NP-hardness of grammar-based compression remains open in the binary case [2, 15]. The relationship of our model with grammar-based compression will be discussed in section 9.

**Open problem 17** *Is it NP-hard to compute finite-state complexity of a given string?*

## 5 Finite-state compression: an example

The complexity  $C$  measure the “recourses” necessary to decode a given encoding of a string. It is natural to ask how powerful are finite-state transducers to compress strings? The following example—in which we use the encoding  $S_0$ —illustrates the power of transducer-based compression.

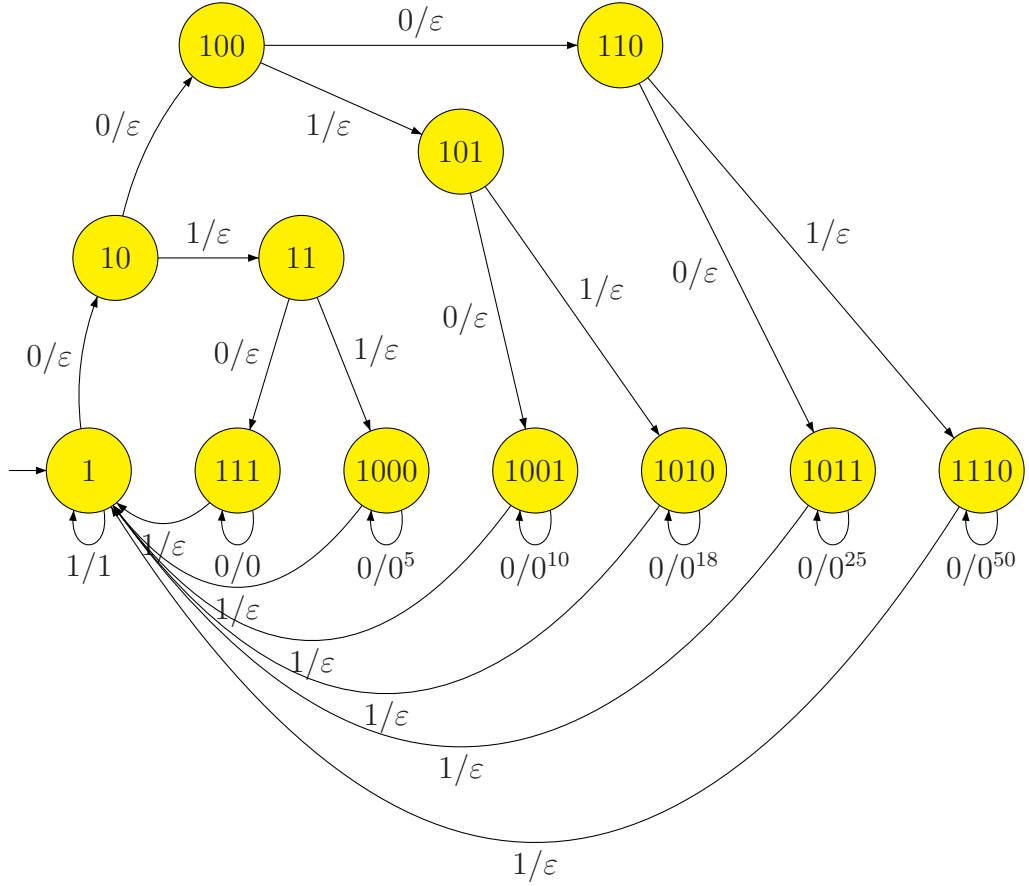


Figure 1: The transducer  $T_1$  for Example 18.

**Example 18** Define the sequence of strings

$$w_m = 010^210^31 \cdot \dots \cdot 0^{m-1}10^m1, \quad m \geq 1.$$

Using the transducer  $T_1$  of Figure 1 we produce an encoding of  $w_{100}$ , where  $|w_{100}| = 5150$ .

With the encodings of the states indicated in the Figure 1,  $T_1$  is encoded by a string  $\sigma_1 \in S_0$  of length 352. Each number  $1 \leq i \leq 100$  can be represented as a sum of, on average, 3.18 numbers from the multi-set  $\{1, 5, 10, 18, 25, 50\}$  [22]. Thus, when we represent  $w_{100}$  in the form  $T_1(p_{100})$ , we need on average at most  $6 \cdot 3.18$  symbols in  $p_{100}$  to output each substring  $0^i$ ,  $1 \leq i \leq 100$ . (This is only a very rough estimate since it assumes that for each element in the sum representing  $i$  we need to make a cycle of length six through the start state, and this is of course not true when the sum representing  $i$  has some element occurring more than once.) Additionally we need to produce the 100 symbols “1”, which means that the length of  $p_{100}$  can be chosen to be at most 2008. Our estimate gives that

$$|(T_{\sigma_1}^{S_0}, p_{100})| = |\sigma_1| + |p_{100}| = 2360,$$

which is a very rough upper bound for  $C^{S_0}(w_{100})$ . ■

**Comment 19** It is easy to see that, using the same transducer,  $C^{S_1}(w_{100}) \leq 2381$ .

The above estimation could obviously be improved using more detailed information from the computation of the average from [22]. Furthermore, [22] does not claim that  $\{1, 5, 10, 18, 25, 50\}$  would, on average, be the most efficient way to represent numbers from 1 to 100 as the sum of the least number of summands.<sup>4</sup> These types of constructions can be seen to hint that computing the value of finite-state complexity may have connections to so called postage stamp problems considered in number theory, with some of their variants known to be computationally hard [14, 21].

For the remanding part of the paper we will use the encoding  $S_0$ .

## 6 Quantitative estimates

Here we establish basic upper and lower bounds for finite-state complexity of arbitrary strings, as well as, for strings of particular types.

**Theorem 20** *For  $n \geq 1$  we have:  $C(0^n) \in \Theta(\sqrt{n})$ .*

**Proof.** It is sufficient to establish that

$$2 \cdot \lfloor \sqrt{n} \rfloor \leq C(0^n) \leq 4 \cdot \lfloor \sqrt{n} \rfloor + c, \tag{6}$$

where  $c$  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  $4 \cdot \lfloor \sqrt{n} \rfloor + c$  where  $c$  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  $\|(T', p')\| \geq 2 \cdot \lfloor \sqrt{n} \rfloor$ . ■

Using a more detailed analysis, the upper and lower bounds of (6) could be moved closer to each other. Because the precise multiplicative constants depend on the particular encoding chosen for the pairs  $(T, \sigma)$  and the language  $S$ , it may not be very important to try to improve the values of the multiplicative constants.

The argument used to establish the lower bound in (6) gives directly the following:

---

<sup>4</sup>In [22] it is established that 18 is the optimal value to add to an existing system of  $\{1, 5, 10, 25, 50\}$ .

**Corollary 21** For any  $x \in \{0, 1\}^*$ ,  $C(x) \geq 2 \cdot \lfloor \sqrt{|x|} \rfloor$ .

The bounds (6) imply that the inequality  $H(xx) \leq H(x) + O(1)$  familiar for program-size complexity does not hold for finite-state complexity:

**Corollary 22** There is no constant  $c$  such that for all strings  $x \in \{0, 1\}^*$ ,  $C(xx) \leq C(x) + c$ .

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

**Corollary 23** For a given transducer  $T$  there is no constant  $c$  such that for all strings  $x \in \{0, 1\}^*$ ,  $C(T(x)) \leq C(x) + c$ .

In Corollary 23 we require only that  $c$  is independent of  $x$ , that is, the value  $c$  could depend on the transducer  $T$ . As in Theorem 20 we get estimations for the finite-state complexity of powers of a string.

**Proposition 24** For  $u \in \{0, 1\}^*$  and  $n \gg |u|$ ,

$$C(u^n) \leq 2 \cdot (\lfloor \sqrt{n} \rfloor + 1) \cdot |u| + 2\sqrt{n} + c, \quad (7)$$

where  $c$  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

$$\|(T, 0^{\lfloor \sqrt{n} \rfloor + y} 1^z)\| \leq 2 \cdot (\lfloor \sqrt{n} \rfloor + 1) \cdot |u| + 2\sqrt{n} + c,$$

where  $c$  is a constant. ■

The upper bound (7) 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  $C(u^n) \leq 2 \cdot |u| + n + c$ , with  $c$  constant.

**Corollary 25** We have:  $C(0^n 1^n) \in \Theta(\sqrt{n})$ .

**Proof.** The lower bound follows from Corollary 21. The string  $0^n 1^n$  has description

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

where  $0 \leq y_1, y_2 \leq 1$ ,  $1 \leq z_1, z_2 \leq \lfloor \sqrt{n} \rfloor$  and  $T$  is the transducer given in Figure 2.

Note that from the construction used in Theorem 20, the transducer in Figure 2 begins by outputting strings  $0^{\lfloor \sqrt{n} \rfloor - 1}$  (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,  $C(0^n 1^n) \leq 8 \cdot \lfloor \sqrt{n} \rfloor + c$ , where  $c$  is a constant. ■

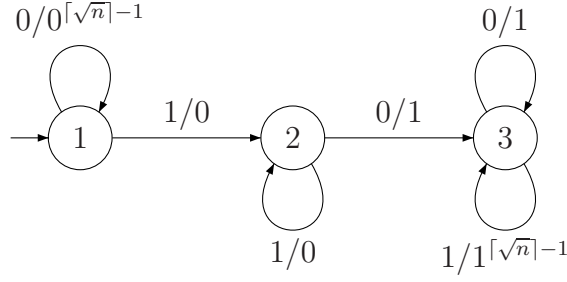


Figure 2: Transducer  $T$  in the proof of Corollary 25.

From Corollary 25 we note that the finite-state complexity of  $0^n 1^n$  is within a constant factor of the automatic complexity, as defined in [24], of the same string. This can be viewed merely as a coincidence since the two descriptive complexity measures are essentially different and have, in general, very different upper and lower bounds.

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

**Proposition 26** *For any  $\omega > 0$  there exists  $d(\omega) > 0$  such that for all  $x, y \in \{0, 1\}^*$ ,*

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

Here the value  $d(\omega)$  depends only on  $\omega$ , i.e., it is constant with respect to  $x, y$ .

**Proof.** Let  $(T, u)$  and  $(R, v)$  be minimal descriptions of  $x$  and  $y$ , respectively. Let  $u = u_1 \cdots u_m$ ,  $u_i \in \{0, 1\}$ ,  $i = 1, \dots, m$  and recall that  $u^\dagger = u_1 0 u_2 0 \cdots u_{m-1} 0 u_m 1$ .

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  $q$  to  $p'$  labeled by  $i/w$  and a transition labeled  $0/\varepsilon$  from  $p'$  to  $p$ .
- (ii) Each state  $p' \in Q'_T$  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  $\omega$  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  $2 \cdot (2|Q_T| + |Q_R|) \leq 6 \cdot \max\{|Q_T|, |Q_R|\}$  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  $|Q_R|$ . ■

It is known that deterministic transducers are closed under composition [3], that is, for transducers  $T_\delta$  and  $T_\gamma$  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 ([3] 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  $\Delta$  is a function  $Q \times \{0, 1\} \rightarrow Q \times \{0, 1\}^*$  and  $\Gamma$  is a function  $P \times \{0, 1\} \rightarrow P \times \{0, 1\}^*$ . The transition function  $\Delta$  is extended in the natural way as a function  $\hat{\Delta} : Q \times \{0, 1\}^* \rightarrow Q \times \{0, 1\}^*$ .

The composition of  $T_\gamma$  and  $T_\delta$  is computed by a transducer

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

The number of states of  $T_\sigma$  is upper bounded by  $|\delta| \cdot |\gamma|$ .<sup>5</sup> An individual output of  $T_\sigma$  consists of the output produced by  $T_\delta$  when it reads an output produced by one transition of  $T_\gamma$  (via the extended function  $\hat{\Delta}$ ). Thus, the length of the output produced by an individual transition of  $T_\sigma$  can be upper bounded by  $|\delta| \cdot |\gamma|$ . These observations imply that

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

---

<sup>5</sup>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.

The above estimate was obtained simply by combining the worst-case upper bound for the size of the encoding of the states of  $T_\sigma$  and the worst-case length of individual outputs of the transducers  $T_\delta$  and  $T_\gamma$ . 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  $\delta$  and  $\gamma$ . The overall upper bound could be somewhat improved using a more detailed analysis.

## 7 Number of states

From Theorem 20 we know that transducers used in minimal descriptions may need to output arbitrarily long strings in a single transition, and hence there is no upper bound for the size of encodings of such transducers. Next we establish that finite-state complexity is a rich complexity measure also with respect to the number of states of the transducers, in the sense that there is no *a priori* upper bound for the number of states used for minimal descriptions of arbitrary strings. This is in contrast to AIT where the number of states of a universal Turing machine can be fixed.

**Theorem 27** *For any  $n \in \mathbb{N}$  there exists a string  $x_n \in \{0, 1\}^*$  such that whenever  $C(x_n) = \|(T_\sigma, p)\|$  the transducer  $T_\sigma$  has more than  $n$  states.*

**Proof.** Consider an arbitrary but fixed  $n \in \mathbb{N}$ . We define  $2n + 1$  strings of length  $2n + 3$ ,

$$u_i = 10^i 1^{2n+2-i}, \quad i = 1, \dots, 2n + 1.$$

Choose

$$m = 32n^2 + 68n + 29 \tag{8}$$

and define

$$x_n = u_1^{m^2} u_2^{m^2} \cdots u_{2n+1}^{m^2}.$$

Let  $(T_\sigma, p)$  be an arbitrary encoding of  $x_n$  where the transducer encoded by  $\sigma$  has at most  $n$  states. We claim that

$$\|(T_\sigma, p)\| > \frac{m^2}{2}. \tag{9}$$

The set of transitions of  $T_\sigma$  can be written as a disjoint union  $\theta_1 \cup \theta_2 \cup \theta_3$ , where

- $\theta_1$  consists of the transitions where the output contains a unique  $u_i$ ,  $1 \leq i \leq 2n + 1$ , as a substring, that is, for any  $j \neq i$ ,  $u_j$  is not a substring of the output;
- $\theta_2$  consists of the transitions where for distinct  $1 \leq i < j \leq 2n + 1$ , the output contains both  $u_i$  and  $u_j$  as a substring;
- $\theta_3$  consists of transitions where the output does not contain any of the  $u_i$ ’s as a substring,  $i = 1, \dots, 2n + 1$ .

Note that if a transition  $\alpha \in \theta_3$  is used in the computation  $T_\sigma(p)$ , the output produced by  $\alpha$  cannot completely overlap any of the occurrences of  $u_i$ 's,  $i = 1, \dots, 2n + 1$ . Hence

$$\text{each transition of } \theta_3 \text{ that is used in the computation } T_\sigma(p) \text{ has length at most } 4n + 4. \quad (10)$$

Since  $T_\sigma$  has at most  $n$  states, and consequently at most  $2n$  transitions, it follows by the pigeon-hole principle that there exists  $1 \leq k \leq 2n + 1$  such that  $u_k$  is not a substring of any transition of  $\theta_1$ . We consider how the computation of  $T_\sigma$  on  $p$  outputs the substring  $u_k^{m^2}$  of  $x_n$ . Let  $z_1, \dots, z_r$  be the minimal sequence of outputs that ‘‘covers’’  $u_k^{m^2}$ . Here  $z_1$  (respectively,  $z_r$ ) is the output of a transition that overlaps with a prefix (respectively, a suffix) of  $u_k^{m^2}$ .

Define

$$\Xi_i = \{1 \leq j \leq r \mid z_j \text{ is output by a transition of } \theta_i\}, \quad i = 1, 2, 3.$$

By the choice of  $k$  we know that  $\Xi_1 = \emptyset$ . For  $j \in \Xi_2$ , we know that the transition outputting  $z_j$  can be applied only once in the computation of  $T_\sigma$  on  $p$  because for  $i < j$  all occurrences of  $u_i$  as substrings of  $x_n$  occur before all occurrences of  $u_j$ . Thus, for  $j \in \Xi_2$ , the use of this transition contributes at least  $2 \cdot |z_j|$  to the length of the encoding  $\|(T_\sigma, p)\|$ .

Finally, by (10), for any  $j \in \Xi_3$  we have  $|z_j| \leq 4n + 4 < 2|u_k|$ . Such transitions may naturally be applied multiple times, however, the use of each transition outputting  $z_j$ ,  $j \in \Xi_3$ , contributes at least one symbol to  $p$ .

Thus, we get the following estimate:

$$\|(T_\sigma, p)\| \geq \sum_{j \in \Xi_2} 2 \cdot |z_j| + |\Xi_3| > \frac{|u_k^{m^2}|}{2|u_k|} = \frac{m^2}{2}.$$

To complete the proof it is sufficient to show that  $C(x_n) < \frac{m^2}{2}$ . The string  $x_n$  can be represented by the pair  $(T_0, p_0)$  where  $T_0$  is the  $2n$ -state transducer from Figure 3 and  $p_0 = (0^m 1)^{2n-1} 0^m 1^m$ .

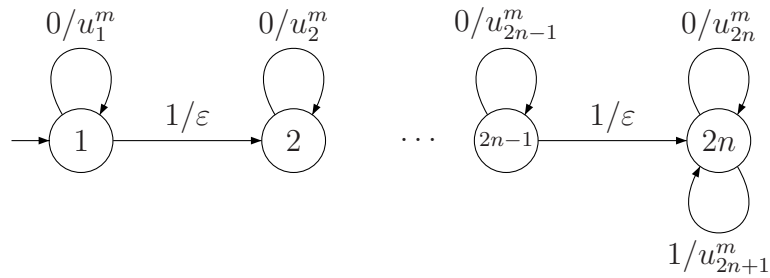


Figure 3: The transducer  $T_0$  from the proof of Theorem 27.

Each state of  $T_0$  can be encoded by a string of length at most  $\lceil \log_2(2n) \rceil$ , so we get the following upper bound for the length of a string  $\sigma_0 \in S$  encoding  $T_0$ :

$$|\sigma_0| \leq (8n^2 + 16n + 6)m + (4n - 2)\lceil \log_2(2n) \rceil + 8n.$$

Noting that  $|p_0| = (2n + 1)m + 2n - 1$  we observe that

$$\|(T_{\sigma_0}, p_0)\| = 2|\sigma_0| + |p_0| = (8n^2 + 18n + 7) \cdot m + f(n),$$

and by using (8) we verify that the term  $f(n)$  is less than  $m$ . Using again (8) we get  $C(x_n) \leq \|(T_{\sigma_0}, p_0)\| < \frac{m^2}{2}$ , which completes the proof. ■

Theorem 27 establishes that there is no upper bound on the number of states of transducers used in minimal descriptions.

**Open problem 28** *It is open whether for every  $n \in \mathbb{N}$  there exists a string  $x_n$  such that a minimal description of  $x_n$  uses a transducer with exactly  $n$  states.*

**Comment 29** Intuitively, the following property would probably seem natural or desirable. If  $u$  is a prefix of  $v$ , then  $C(u) \leq C(v) + c$  where  $c$  is a constant independent of  $u$  and  $v$ . However, the below example, based on a construction similar to the one used in Theorem 27, seems to contradict this property.

For  $m \geq 1$ , define

$$u_m = 0^{m^2} 1^{m^2} 0^{m^2-1}, \quad v_m = 0^{m^2} 1^{m^2} 0^{m^2}.$$

Now  $v_m$  has a description  $(T, 0^m 1^m 0^m)$  where  $T$  is the single-state transducer with two self-loops labeled by  $0/0^m$  and  $1/1^m$ . Thus,  $C(v_m) \leq 7 \cdot m + c$  where  $c$  is a constant independent of  $m$ .

We do not know how to construct a description for  $u_m$ ,  $m \geq 1$ , having size  $7 \cdot m + O(1)$ .

**Open problem 30** *Obtain a reasonable upper bound for  $C(u)$  in terms of  $C(v)$  when  $u$  is a prefix of  $v$ .*

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*.

**Fact 31** *There exist finite-state incompressible strings of any length.*

**Proof.** We note that the set  $\{x \mid |x| = n, C(x) \leq |x| - i\}$  has at most  $2^{n-i+1} - 1$  elements. ■

## 8 An algorithm for computing finite-state complexity

A “brute-force” algorithm computing finite-state complexity is presented as pseudocode in Algorithm 8.1.

---

**Algorithm 8.1:** FINITESTATECOMPLEXITY(String  $x$ )

---

**procedure** FSCOMPLEXITY(String  $x$ )

$upperBd \leftarrow \text{UPPERBOUND}(x)$

**for each** string enumeration  $\rho$  such that  $|\rho| \leq upperBd$

**do**  $\left\{ \begin{array}{l} \text{if } \text{CORRECTENCODING}(\rho) \text{ and } \text{OBTAINSX}(T_\sigma, p, x) \\ \text{then return } ( |\rho| ) \end{array} \right.$

**procedure** UPPERBOUND(String  $x$ )

$upBd \leftarrow |x| + 8$

**return** (  $upBd$  )

**procedure** CORRECTENCODING(String  $\rho$ )

**if**  $\rho$  has correct pattern

$\left\{ \begin{array}{l} \sigma \leftarrow \text{prefix of } \rho \\ p \leftarrow \text{rest of } \rho \\ \text{comment: find and assign each } \text{bin}(i) \text{ and } u_i \text{ to their array} \\ \text{then } \left\{ \begin{array}{l} \text{for each } \text{bin}(i) \in \sigma \text{ and } u_i \in \sigma \text{ (i.e in } T_\sigma) \\ \text{do } \left\{ \begin{array}{l} \text{bins} \leftarrow \text{bin}(i) \\ \text{outs} \leftarrow u_i \end{array} \right. \\ \text{if } |\text{bins}| = |\text{outs}| \text{ and } |\text{bins}| \text{ is even and } |\text{bins}| > 0 \\ \text{then return } ( \text{true} ) \end{array} \right. \end{array} \right.$

**return** (  $\text{false}$  )

**procedure** OBTAINSX(Array  $bins$ , Array  $outs$ , String  $p$ , String  $x$ )

**for each**  $\text{bin}(i) \in bins$

**do**  $\left\{ \begin{array}{l} \text{if } i = 0 \text{ or } i \geq \frac{|\text{bins}|}{2} \\ \text{then return } ( \text{false} ) \\ \text{trans} \leftarrow i \text{ (where } \text{trans} \text{ is the transition array)} \end{array} \right.$

$output \leftarrow \varepsilon$

$k \leftarrow 0$

**for each** character  $p_i \in p$

$\left\{ \begin{array}{l} \text{if } p_i = '0' \\ \text{then } \left\{ \begin{array}{l} t \leftarrow \text{element of } \text{trans} \text{ at } k \\ \text{output} \leftarrow \text{output} + \text{element of } \text{outs} \text{ at } k \end{array} \right. \\ \\ \text{else } \left\{ \begin{array}{l} t \leftarrow \text{element of } \text{trans} \text{ at } k + 1 \\ \text{output} \leftarrow \text{output} + \text{element of } \text{outs} \text{ at } k + 1 \end{array} \right. \\ k \leftarrow 2 \times (t - 1) \end{array} \right.$

**if**  $output = x$

**then return** (  $\text{true}$  )

**return** (  $\text{false}$  )

---

The general idea of the algorithm is that given a binary input string  $x$ , it enumerates

all possible strings  $\rho$ , where  $\rho$  is intended to be of the form  $\sigma^\dagger \cdot p$ . Thus we must check, for every enumerated  $\rho$ , whether it is of the form  $\sigma^\dagger \cdot p$ . If so, then we extract from  $\rho$  the pair  $(T_\sigma, p)$  and test whether  $T_\sigma(p) = x$ . If we do obtain  $x$  from the extracted pair, then we can calculate the complexity,  $C_\sigma^S(x) = 2|\sigma| + |p|$ , and terminate the algorithm. In all other cases, we reject the current  $\rho$  and enumerate the next string  $\rho$ .

We now discuss each procedure of the algorithm separately.

The enumeration procedure of binary strings is done in lexicographical order, for every length, starting from the description of the smallest transducer, which outputs  $\varepsilon$  on all inputs.

The *FSDComplexity* procedure is the core of the algorithm. It takes a string  $x$  as input and manages both the enumeration and the other processes. Its role is to find the shortest string  $\rho$  which is the correct encoding of a pair  $(T_\sigma, p)$  that computes  $x$ . As the enumeration is in lexicographic order, the length of the first such pair is the complexity of the string.

The *CorrectEncoding* procedure first checks whether the enumerated  $\rho$  (passed as input to this procedure) is of the form  $\sigma^\dagger \cdot p$ . In the affirmative, it decodes  $\rho$  into  $\sigma^\dagger$  and  $p$  and then checks whether these components have the correct form and extracts  $T_\sigma$  from  $\sigma^\dagger$ . Finally, if still in the affirmative, this procedure extracts the transition and the output functions from  $T_\sigma$  and checks whether the size of their ranges are equal and even. If all of these conditions hold, the enumerated  $\rho$  is a correct encoding, i.e.  $\rho = \sigma^\dagger \cdot p$  where  $\sigma \in S$ . Otherwise, the procedure rejects the current  $\rho$ , returning to the *FSDComplexity* procedure to enumerate the next possible  $\rho$ .

The final procedure, *ObtainsX*, is only called if we have a potential pair  $(T_\sigma, p)$ . It uses the extracted functions (passed as inputs to the procedure along with our original input string  $x$  and  $p$ ) to simulate the computation of the transducer  $T_\sigma$  on the input  $p$  and stores the output. When the simulation is complete, *ObtainsX* compares the obtained output with  $x$ ; if they are identical, then we have found a minimal description  $(T_\sigma, p)$  for  $x$  and its length is calculated in the main procedure *FSDComplexity*, completing the process.

It is clear that since we enumerate through all possible binary strings, starting from the description of the smallest  $\sigma$  in  $S$ , every possible pair will eventually be enumerated exactly once. If any one of the steps in the ‘decoding’ fails, the algorithm rejects the considered  $\rho$  and enumerates the next string in lexicographic order. Thus, for every input string  $x$ , the algorithm finds a minimal description, computes the finite-state complexity of  $x$ , and halts.

We illustrate the Algorithm 8.1 in Table 4. Note that if  $x$  is in the table then  $\bar{x}$  is omitted as  $C(x) = C(\bar{x})$ .

Clearly, a minimal description  $(T_\sigma, p)$  of a string  $x$  need not be unique because we can simply interchange the input symbols 0 and 1 in the transitions of  $T_\sigma$  and in  $p$ .

**Open problem 32** *Describe the set of all strings having more than two different minimal descriptions.*

$x$	$C(x)$	$(\sigma, p)$	$x$	$C(x)$	$(\sigma, p)$
$\varepsilon$	4	(0000, $\varepsilon$ )	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	(01010110,10000)

Table 4: Finite-state complexity of all strings in lexicographic order from  $\varepsilon$  to 01111.

## 9 Grammar-based compression and lower bounds for finite-state complexity

A grammar  $G$ , or straight-line program [10, 16, 19], 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  $X_1, \dots, X_m$  such that the productions are of the form

$$X_1 \rightarrow \alpha_1, \dots, X_m \rightarrow \alpha_m$$

where  $\alpha_i$  contains only nonterminals from  $\{X_{i+1}, \dots, X_m\}$  and terminal symbols. The *size of the grammar*  $G$ ,  $\text{size}(G)$ , is defined to be  $\sum_{i=1}^m |\alpha_i|$ . In cases where the grammars are restricted to be in Chomsky normal form sometimes the size of the grammar is considered to be simply the number of nonterminals [19].

Grammar-based compression of a string  $x$  may result in exponential savings compared to the length of  $x$ : the grammar  $X_i \rightarrow X_{i+1}X_{i+1}$ ,  $i = 1, \dots, m-1$ ,  $X_m \rightarrow aa$  generates the string  $a^{2^m}$ . Also, it is known that the smallest grammar for a string of length  $n$  has size  $\Omega(\log n)$  [15].

Comparing the above to Corollary 21, we note that the grammar-based complexity, or 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 a binary encoding of size  $O(C(x))$ .

**Lemma 33** *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 [8] 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  $(T_\sigma, p)$ , 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  $T_\sigma$  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  $T_\sigma$  in state  $s$  on input  $p_i$ .

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

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

■

The size of a grammar counts simply the number of occurrences of nonterminals in the productions. However, Lemma 33 holds also if we use the length of a binary description of  $G_x$  as  $\text{size}(G_x)$ . Note that in the construction the nonterminals of  $G_x$  are pairs consisting of a binary bit and a state of  $T_\sigma$ .

Using Lemma 33 and known results on grammar-based compression we get lower bounds for finite-state complexity of explicitly constructed strings. We already know that there exist finite-state incompressible strings of any length, however, the proof of Lemma 31 relies on counting arguments and does not give a construction of incompressible strings.

We recall the following notions. A binary *de Bruijn string* of order  $r \geq 1$  is a string  $w$  of length  $2^r + r - 1$  over alphabet  $\{0, 1\}$  such that any string of length  $r$  occurs as a substring of  $w$  (exactly once). It is well known that de Bruijn strings of any order exist, and have an explicit construction [9, 26].

**Proposition 34** *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*

$$C(w) \geq d \cdot \frac{|w|}{\log(|w|)}.$$

**Proof.** It is known that any grammar generating a de Bruijn string of order  $r$  has size  $\Omega\left(\frac{2^r}{r}\right)$  [1]. Grammars generating a singleton language are called string chains in [1], see also [12]. The claim follows by Lemma 33.

■

We do not have an explicit construction of provably incompressible strings.

**Conjecture 35** *de Bruijn strings are finite-state incompressible.*

It is known that computing the exact grammar-based complexity of a given string is NP-complete [25] and recent work has focused on approximating the size of the smallest grammar [2, 8, 16, 19]. However, it remains open whether the problem of finding the exact minimal grammar is NP-hard when the terminal alphabet is restricted to be binary [2, 15]. Since finite-state complexity is a special case of grammar-based complexity and we employ a binary alphabet it may not be easy to get an NP-hardness result for computing  $C(x)$ .

## 10 Summary

In this paper we have developed 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 descriptive complexities (plain, prefix-free) from AIT, there is no *a priori* upper bound for the number of states used for minimal descriptions of arbitrary strings. Explicit examples of finite-state incompressible strings have been given as well as upper and lower bounds for finite-state complexity of arbitrary strings and for strings of particular types.

Some open questions have been discussed throughout the paper. There are many possible continuations. Here are a few possibilities we are working on: a) further study finite-state random strings (in particular, check whether these strings are Borel normal [5], b) use finite-state complexity to define and study finite-state random sequences and compare results with the class of Borel normal sequences, c) classify infinite sequences according to various natural finite-state reducibility relations (e.g. the sequence  $\mathbf{x}$  is finite-state reducible to the sequence  $\mathbf{y}$  if there is a finite transducer transforming  $\mathbf{x}$  into  $\mathbf{y}$ ).

## Acknowledgments

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.

## References

- [1] I. Althöfer. Tight lower bounds on the length of word chains, *Inform. Proc. Lett.* 34 (1990) 275–276.
- [2] J. Arpe and R. Reischuk. On the complexity of optimal grammar-based compression, *Proc. of the Data Compression Conference, DCC'06*, 2006, 173–186.
- [3] J. Berstel. *Transductions and Context-free Languages*, Teubner, 1979.
- [4] H. Buhrman and L. Fortnow. Resource-bounded Kolmogorov complexity revisited, *Proceedings STACS'97*, LNCS 1200, Springer-Verlag, 1997, 105–116.

- [5] C. S. Calude. *Information and Randomness—An Algorithmic Perspective*, 2nd ed., Springer, Berlin, 2002.
- [6] C. S. Calude, N. J. Hay, F. C. Stephan. Representation of Left-Computable  $\varepsilon$ -Random Reals, *CDMTCS Research Report* 365, 2009, 11 pp.
- [7] G. Chaitin. *Algorithmic Information Theory*, Cambridge University Press, 1987.
- [8] M. Charikar, E. Lehman, D. Liu, R. Panigrahy, M. Prabhakaran, A. Rasala, A. Sahai and A. Shelat. Approximating the smallest grammar: Kolmogorov complexity in natural models, *Proceedings of STOC'02*, ACM Press, 2002, 792–801.
- [9] N. de Bruijn. A combinatorial problem, *Proc. Kon. Nederl. Akad. Wetensch.* 49 (1946), 758–764.
- [10] A. Diwan. A new combinatorial complexity measure for languages. Technical Report, Computer Science Group, TaTa Institute, Bombay, 1986.
- [11] R. Downey, D. Hirschfeldt. *Algorithmic Randomness and Complexity*, Springer, Heidelberg, 2010 (forthcoming)
- [12] M. Domaratzki, G. Pighizzini and J. Shallit. Simulating finite automata with context-free grammars, *Inform. Proc. Lett.* 84 (2002), 339–344.
- [13] D. Doty and P. Moser. Feasible depth, [arXiv:cs/0701123v3](https://arxiv.org/abs/cs/0701123v3), April 2007.
- [14] R. K. Guy. *Unsolved Problems in Number Theory*, 3rd ed., Springer, Berlin, 2004.
- [15] E. Lehman. *Approximation Algorithms for Grammar-based Compression*, PhD thesis, MIT, 2002.
- [16] E. Lehman and A. Shelat. Approximation algorithms for grammar-based compression, *SODA'02*, SIAM Press, 2002, 205–212.
- [17] A. Nies. *Computability and Randomness*, Clarendon Press, Oxford, 2009.
- [18] S. Porrot, M. Dauchet, B. Durand and N. Vereshchagin. Deterministic rational transducers and random sequences, *FoSSaCS'98*, LNCS 1378, Springer, 1998, 258–272.
- [19] W. Rytter. Application of Lempel-Ziv factorization to the approximation of grammar-based compression, *Theoret. Comput. Sci.* 302, 2002, 211–222.
- [20] M. Sipser. *Introduction to the Theory of Computation*, PWS 1997.
- [21] J. Shallit. The computational complexity of the local postage stamp problem, *SIGACT News* 33 (2002), 90–94.
- [22] J. Shallit. What this country needs is an 18 cent piece, *Mathematical Intelligencer* 25, 2003, 20–23.

- [23] J. Shallit and Y. Breitbart. Automacity I: Properties of a measure of descriptonal complexity. *J. Comput. System Sci.* 53 (1996), 10–25.
- [24] J. Shallit and M.-W. Wang. Automatic complexity of strings, *J. Automata, Languages and Combinatorics* 6, 2001, 537–554.
- [25] J. Storer. NP-completeness results concerning data compression, *Technical Report 234*, Dept. of Electrical Engineering and Computer Science, Princeton University, 1977.
- [26] J.H. van Lint and R.M. Wilson. *A Course in Combinatorics*, Cambridge University Press 1993.
- [27] S. Yu. Regular languages, in: G. Rozenberg, A. Salomaa (eds.). *Handbook of Formal Languages*, vol. I, Springer, Berlin, 1997, 41–110.