University of Auckland
Faculty of Science
Computer Science

Honours Thesis

Finite-State Descriptional Complexity

by

Tania K. Roblot

under the supervision of
Prof. Cristian S. Calude and Dr. André Nies

A dissertation submitted in partial fulfilment of the requirements
for the degree of Bachelor of Science (Honours) in Computer Science,
The University of Auckland, 2009.

To my mother, Sylvie Roblot

Abstract

Algorithmic information theory is a specific branch of computational complexity theory, which is concerned with randomness and the complexity of strings (or sequences). Randomness and complexity are intertwined topics of research, as one highly affects the other. In our research, we are concerned with defining a computable analogue to Kolmogorov complexity. In order to do so, we explore the concept of depth and have taken inspiration from Doty and Moser's work, who touch on this idea of an analogous complexity, but only briefly define it in order to define their *feasible depth*. In this dissertation, we give a literature review of the work done on depth, then move on to defining and proving a computable complexity for finite strings, *finite-state descriptional complexity*. We also offer an implementation of this complexity via a brute-force algorithm.

Acknowledgements

I would like to give special thanks to Prof. Kai T. Salomaa for his precious input in this work, his expertise in finite automata was crucial to its success. I would also like to thank Prof. Cristian S. Calude and Dr. André Nies for this fantastic opportunity to work on this exciting project.

Contents

Chapter 1

Introduction

In 1988, Bennett explored the notion of depth as a measure of a string or sequence's useful information content, which he defined as *logical depth* [3]. His goal was to mathematically represent our intuitive notion of depth. This concept is based on Kolmogorov complexity, a compressibility measure of a string (respectively, sequence), which implicitly measures the redundancy within a given string (sequence). The formal definition will be given in Chapter 2. As logical depth is dependent upon Kolmogorov complexity, one main issue arises: logical depth is uncomputable. Therefore, it has no practical application. In this dissertation, we present a literature review for both Bennett's work and its continuation, through Juedes *et al.* and Doty and Moser's works, namely *Computational Depth and Reducibility* [9] and *Feasible Depth* [7].

Doty and Moser's paper is of particular interest to us as they have defined an analogous notion of depth with respect to finite automata, particularly finite-state transducers. These results have inspired us to follow a similar approach in order to define a computable complexity measure, analogous to Kolmogorov's complexity, in the domain of finite-state transducers, namely *finite-state complexity*. Succeeding in defining and proving such a complexity would imply that we could finally compute complexity values for strings in a practical sense — as well as extract relevant properties.

In this dissertation, we will first introduce, in Chapter 2, the notion of depth in both its classical sense, from Bennett and Juedes *et al.*'s main results, and its finite-state machine context, Doty and Moser's feasible depth. We will have then covered all of the necessary background to present our own findings on finite-state complexity in Chapter 3. Finally, in Chapter 4, we will present our conclusions and thoughts on further research.

Chapter 2

Depth: A Literature Review

2.1 Motivation

As expressed in the introduction, logical depth is a measure of a string's (respectively, sequence's) useful information content. Bennett also expressed that his logical depth is meant as "a formal measure of value", meaning that it uses the *information content*, given by the Kolmogorov complexity, and identifies its "buried redundancy" or computational history. In other words, logical depth attempts to identify and measure the amount of computational work it took for the originator to build the given string (sequence). Thus, sequences obtain their information content from a long and complex computation, and require the same for one's information to be fully extracted and revealed.

In order to understand what Bennett means by our intuitive notion of depth, consider the following two examples given by Li and Vitányi [11]: DNA and the Set Theory textbook.

The current state of our DNA is a result of evolution. It has taken a significantly long time to reach its current state and has resulted in a complex set of information on how to 'construct' and maintain a surviving organism. As for the example of the Set Theory textbook, Li and Vitányi explain how it takes a long time to extract all of the (useful) information from this given textbook, because of the complex and dense material encapsulated in it. Although it could be 'compressed' to a few main theorems, it would come at the cost of losing precision and therefore some information.

In this chapter, we will first explicitly present the necessary definitions and theorems from the classical notion of depth in order to prove and comprehend the main results which concern our work: Ω is shallow, whereas χ_K is strongly deep. Then, we will also present the 'finite-state' notion of depth, feasible depth, and its analogous properties to the classical notion of depth.

2.2 Classical Notion of Depth

2.2.1 Classical Complexity Concepts

Kolmogorov Complexity

The Kolmogorov complexity is a core concept of algorithmic information theory and is defined as follows:

Definition 1 (Kolmogorov Complexity)**.** *Let $x \in \{0,1\}^*$, let p be a program and let U be a (prefix-free) Universal Turing machine.*
The (self-delimiting) Kolmogorov complexity of x (relative to U) is

$$K(x) = \min_{p \in \{0,1\}^*} \Big\{ |p| \;\Big|\; U(p) = x \Big\}. \tag{2.1}$$

Kolmogorov complexity is the size of the minimal program p such that U runs p and halts with x on its tape [14]. Kolmogorov complexity, $K(x)$, is also known as the *algorithmic entropy* or *algorithmic information content* of x.

In this paper, we will omit the adjective "self-delimiting" as it is the only form of Kolmogorov complexity encountered here. However, Kolmogorov complexity is self-delimiting when the Universal Turing machine runs a *self-delimiting program* [3], meaning that U will halt in t steps, and so needs to compute x from p in that amount of steps. This t therefore allows us to measure the computation time.

The Halting Problem

Definition 2 ($HALT$)**.** *$HALT$ is a TM which takes as input the pair $\langle M, x \rangle$ where M is a TM and x is a string, and accepts if M halts on x; i.e. if M halts and accepts, or halts and rejects on input x.*

Theorem 3 (The Halting Problem)**.** *$HALT$ is uncomputable.*

This is a well-known result in Computational Complexity Theory. The proof of this problem is found in the following textbooks: *Introduction to Computational Complexity* by M. Sipser [14] and *Computational Complexity: A Modern Approach* by S. Arora and B. Barak [1].

To give a more intuitive idea of the implications of this problem, note that if $HALT$ were computable, we could predict the future.

Definition 4 (Chaitin's Ω)**.** *Ω is a real number between 0 and 1, which represents the Halting problem. It is of the form $0.u_1u_2u_3...u_n...$ where $u_i \in \{0,1\}$ determines whether the program of length i halts or not.*

In other words, knowing the first n bits of Ω would mean that we could solve the Halting problem for the all programs (of Turing machines) of length up to n. Ω is a true random number and, thus, is uncomputable. Being a random number implies that Ω cannot be compressed.

The characteristic sequence of the diagonal halting problem is denoted as χ_K. It represents a specific pairing function between the input and the TM for the input of $HALT$. We define χ_K as follows:

Definition 5 (Characteristic Representation of the Diagonal Halting Problem)**.** *Let $M_0, M_1, ..., M_i, ...$ be a standard enumeration of TMs.*
Let $n \in \mathbb{N}$ and $\chi_K \in \mathbb{C}$ (where $\mathbb{C} = \{0,1\}^\infty$ is the Cantor space) such that

$$\chi_K[n] = \begin{cases} 1 & \text{if } M_n(n) \text{ halts,} \\ 0 & \text{otherwise.} \end{cases}$$

Again, since the halting problem is uncomputable, so is χ_K. However, unlike Ω, χ_K has some redundancy. In fact, χ_K's compressed form is Ω.

2.2.2 Bennett's Notion of Depth

Each of the following definitions and theorems in this subsection have been extracted from Bennett's work [3].

Finite Strings

Bennett defines two types of depths of finite strings: the depth and the relative depth of a string. We are only concerned with the depth of finite strings. For more information on the relative depth of a finite string, refer to Bennett's *Logical Depth and Physical Complexity* paper [3].

For the continuation of this dissertation, let $T : \{0,1\}^* \to \mathbb{N}$ be a function of time, which is related to the number of steps taken by the machine to run on its input w. We will denote $T_M(w)$ as the measure of the time taken by the machine M to run w. If the machine is implicit, then we denote the function as $T(w)$. If M does not halt, then $T_M(p) = \infty$.

Definition 6 (Logical Depth)**.** *Let x be a string, p be some TM description (or program) and $s \in \mathbb{N}$ a significance parameter. Let p^* be a minimal program for x. A string's* logical depth at a significance level s *is defined as*

$$D_s(x) = \min\left\{T(p) \;\middle|\; (|p| - |p^*| < s) \wedge U(p) = x\right\}.$$

This is the form of the definition as stated in Bennett's paper [3]. However, note that it is equivalent to the following:

$$D_s(x) = \min\left\{T(p) \;\middle|\; (|p^*| > |p| - s) \wedge U(p) = x\right\},$$

and, we know that $|p^*| = K(p)$, hence,

$$D_s(x) = \min\left\{T(p) \;\middle|\; (K(p) > |p| - s) \wedge U(p) = x\right\}. \tag{2.2}$$

Intuitively, the depth of a finite string x is the shortest time needed for the Universal Turing machine to obtain x on its tape from some s-incompressible program p. Note that the program does not need to be minimal, it simply needs to be executable as fast as possible to obtain x and the fastest running such program may not be the minimal program for x. Therefore, the longer the computation to obtain x, the deeper x is. Note that this notion of depth is also related to some "significance level", which acts as a threshold to how deep a string we are testing for.

As far as strings are concerned, depth is a binary property. Therefore a string is shallow if it is not deep.

Sequences

Sequences are infinite strings. Accordingly, the concept of depth becomes more complex and is in fact a ternary relation: strong depth, weak depth and shallowness.

Definition 7 ([3], Strongly Deep)**.** [1] *A sequence S is called strongly deep if for all s, where s is a significance level, for every recursive function f and for all but finitely many $n \in \mathbb{N}$,*

$$D_s(S \upharpoonright n) > f(n). \tag{2.3}$$

A sequence is strongly deep if it takes a long time to compute it. This definition implies that the larger the segment or substring of S, then the time needed to compute that segment grows significantly.

Definition 8 ([3], Weakly Deep)**.** *A sequence is weakly deep if it is not computable in recursively bounded time from any algorithmically random infinite sequence.*

We will not focus on weakly deep sequences in this dissertation, but it is a necessary concept in order to define shallow sequences.

Theorem 9. *Every strongly deep sequence is weakly deep, but the converse does not hold.*

Definition 10 (Shallow)**.** *A sequence is shallow if it is neither weakly nor strongly deep.*

[1] Please see the Appendix A for the notation references.

2.2.3 Juedes, Lathrop and Lutz's Notion of Depth

Finite Strings

Juedes, Lathrop and Lutz re-express Bennett's notion of depth by first introducing a new set, $PROG$ [9].

Definition 11 ($PROG$)**.** *This set introduces a standard representation for the set of programs for some string x relative to a TM, M, as follows:*

$$PROG_M(x) = \Big\{ p \in \{0,1\}^* \;\Big|\; M(p) = x \Big\},$$

and similarly, the set of t-fast programs for x relative to M is

$$PROG^t_M(x) = \Big\{ p \in PROG_M(x) \;\Big|\; T_M(p) \leq t(|x|) \Big\},$$

where, t is a time bound and T_M is the same time function as with Bennett's work.

When relative to the Universal Turing machine, we simply write $PROG(x)$ and $PROG^t(x)$.

Using the above sets, the definition of depth for initial segments of sequences (and therefore finite strings), as expressed in Juedes *et al.*'s paper [9], is as follows:

Definition 12 (Depth)**.** *For $t, g : \mathbb{N} \to \mathbb{N}$ and $n \in \mathbb{N}$,*

$$D^t_g(n) = \Big\{ S \in \{0,1\}^\infty \;\Big|\; \forall p \in PROG^t(S \upharpoonright n), K(p) \leq |p| - g(n) \Big\},$$

where $g(n)$ evaluates to some constant. Thus, $\exists c \in \mathbb{N}$ such that

$$D^t_c(n) = \Big\{ S \in \{0,1\}^\infty \;\Big|\; \forall p \in PROG^t(S \upharpoonright n), K(p) \leq |p| - c \Big\}. \tag{2.4}$$

In fact, this definition of depth is very similar to Bennett's, as expected, but returns a *set of sequences* whose initial segment, $S \upharpoonright n$, is deep, in the sense of finite string depth.

Sequences

From the above definition of depth, we can simply test that (almost) all initial segments of a given sequence are deep. Also note that if $\forall n \in \mathbb{N},\ S_1 \in D^t_c(n)$, then $S_1 \in D^t_c$. So, we obtain the following formal definition:

Definition 13 (Strongly Deep)**.** *A sequence $S \in \mathbb{C}$ is strongly deep, denoted $S \in$ strDEEP, if $\forall t : \mathbb{N} \to \mathbb{N}$ and $\forall c \in \mathbb{N}$, $S \in D^t_c$.*

Intuitively, a sequence is strongly deep if given more time, it can always be compressed further.

Definition 14 (Weakly Useful)**.** *Let $r : \mathbb{N} \to \mathbb{N}$ be a recursive time bound and let σ be the set of all sequences S' that are Turing reducible to S in polynomial time, bounded by r. If r exists such that σ does not have measure 0 in* REC*, in other words, $\sigma \cap$ REC $\neq \emptyset$. Then a sequence S is weakly useful. We will denote $S \in$ wkUSE.*

2.2.4 Theorems

In this section, we will include the necessary theorems to prove the main results which concern this research. However, note that not all of the theorems will be proven. In this literature review, we will only prove those which have the most impact on our findings. Note that the proof to each theorem may be found in its corresponding paper [3, 9].

Theorem 15 ([3], Slow Growth Law)**.** *$\exists c_1, c_2 \in \mathbb{N}$ such that $\forall x \in \{0,1\}^*$ and $\forall s_1, s_2 \in \mathbb{N}$, pairs of significant parameters where $s_2 > s_1$, the prefix-free set of (self-delimiting) programs*

$$\left\{ p \;\middle|\; D_{s_2}(U(p)) > D_{s_1}(x) + T(p) + c_1 \right\} \tag{2.5}$$

has measure less than $2^{-(s_2 - s_1) + c_2}$. Here, T is the function of time defined for Definition 6.

This simply means that obtaining a deep string from a shallow one is very unlikely and to do so would take an extremely long computation; thus, naming it the *slow growth law*. Considering the very small probability involved to obtain a deep string from a shallow string, we can think of this law as the *invariance theorem of depth*, which consequently also applies to strongly deep sequences.

Juedes, Lathrop and Lutz [9] give their own version of the *invariance theorem for strongly deep sequences* as follows:

Theorem 16 ([9], Depth Invariance Theorem)**.** *Let $x, y \in \mathbb{C}$.*

$$\textit{If } y \leq_{tt} x \textit{ and } y \in \textsf{strDEEP}\textit{, then } x \in \textsf{strDEEP},$$

where $y \leq_{tt} x$ means that y is truth-table reducible to x.

Let us now consider the sets of sequences and their depth properties.

RAND is the set of random sequences, as expressed by Juedes *et al.* [9]. We give you an intuitive proof to the following theorem.

Theorem 17 ([9])**.** *RAND is shallow.*

Proof. Given any sequence $S_1 \in \mathsf{RAND}$, all of its information is explicitly given (i.e. the entire sequence). So, S_1 cannot be compressed. Also, S_1 is built arbitrarily, thus, not through a long and complex process. So, S_1 is shallow.

Hence RAND is shallow. ■

This theorem is formally proven by Juedes *et al.*, by proving that $\exists t, t(n) = O(n \log n)$, a recursive function, and $\exists c \in \mathbb{N}$ such that $\mathsf{RAND} \cap D_c^t = \emptyset$, which proves that

$$\mathsf{RAND} \cap \mathsf{strDEEP} = \emptyset.$$

Later in the paper, they prove that

$$\mathsf{RAND} \cap \mathsf{wkDEEP} = \emptyset,$$

where wkDEEP stands for the set of weakly deep sequences; thus, proving the theorem.

Let us now consider the set of recursive sequences, denoted REC [9].

Theorem 18 ([9])**.** *REC is shallow.*

Juedes *et al.* prove that

$$\mathsf{REC} \cap \mathsf{strDEEP} = \emptyset.$$

Proof. Fix some $y \in \mathsf{RAND}$, $\forall x \in \mathsf{REC}$, then $x \leq_{tt} y$. So, $y \notin \mathsf{strDEEP}$. Thus, by Theorem 16, $x \notin \mathsf{strDEEP}$. Therefore, $\mathsf{REC} \cap \mathsf{strDEEP} = \emptyset$. ■

Later, they also prove that

$$\mathsf{REC} \cap \mathsf{wkDEEP} = \emptyset,$$

and hence, REC is shallow.

Theorem 19 ([9])**.** *Every weakly useful sequence is strongly deep. Also written as*

$$\mathit{wkUSE} \subseteq \mathit{strDEEP}.$$

Proof. Let $x \in \mathbb{C}$ and that x is weakly useful. By definition, there is a recursive time bound $r : \mathbb{N} \to \mathbb{N}$ such that σ does not have measure 0 in REC. Since every recursive function is bounded above by a strictly increasing, time-constructible function, *wlog*, s is strictly increasing and time-constructible.

Let $t : \mathbb{N} \to \mathbb{N}$ be a recursive time bound and let $c \in \mathbb{N}$.

By Lemma 5.5 and Corollary 5.9 in *Computational Depth and Reducibility* [9], every recursive function with a certain property which holds for $\tilde{t}$ (a function of t from Lemma 5.5 of their paper), have their corresponding $D_c^{\tilde{t}}$ has measure 1 in REC.

So, $D_c^{\tilde{t}} \cap \sigma = \emptyset$. and by Lemma 5.5 (see [9]), $x \in D_c^t$.

Hence, $x \in \mathsf{strDEEP}$. ■

2.2.5 Main Results of Classical Depth

Chaitin's Ω

Theorem 20. *Ω is Shallow.*

In his paper, Bennett claims that Ω is shallow because it is algorithmically random. Juedes, Lathrop and Lutz prove this claim, thanks to Theorem 17; since $\Omega \in \mathsf{RAND}$, Ω is shallow.

This is an interesting result as it implies that although Ω represents a deep problem of Algorithmic Information Theory, i.e. the halting problem, it is shallow itself. This seems contradictory and suggests that maybe Chaitin's Ω is not the best representation of this problem. This result does not imply any incorrectness. It implies, however, that Ω is such a compressed version of the halting problem, that the information would be too easily retrieved from it — which is how we have defined Ω earlier in this chapter.

The Halting Problem's χ_K

Juedes *et al.* [9] prove that:

Theorem 21 ([9])**.** *χ_K is strongly deep.*

The sketch of the proof goes as follows: since the diagonal halting problem is polynomial-time complete for the set of all recursively enumerable subsets of $\mathbb{N}$, $\chi_K \in \mathsf{wkUSE}$. Therefore, by Theorem 19, $\chi_K \in \mathsf{strDEEP}$.

This implies that even though Ω is a shallow representation of the halting problem, χ_K is a strongly deep representation. Thus, to extract the information about the halting problem from χ_K would take a long computation. Additionally, χ_K witnesses the depth and complexity of the problem it represents, an uncomputable problem. This result also confirms our claim that Ω is the completely compressed 'version' of χ_K, since they represent the same problem.

2.3 Feasible Depth

2.3.1 Finite-State Transducers

A generalized finite transducer [4] is a tuple

$$T = (X, Y, Q, q_0, Q_F, E), \tag{2.6}$$

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 $e = (q_1, u, v, q_2) \in E$, $q_1, q_2 \in Q$, $u \in X^*$, $y \in Y^*$ is a transition, 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 $e_1, \ldots, e_k$ and x (respectively, y) is the concatenation of input labels (respectively, output labels) of $e_1, \ldots, e_k$. For a more formal definition of the transduction realised by T we refer the reader to [4]. It is well-known that finite transducers realise exactly the rational relations [4].

Lemma 22. ([4], Corollary 6.2) *Any rational relation can be realised by a transducer where the transitions are a subset of* $Q \times (X \cup \{\varepsilon\}) \times (Y \cup \{\varepsilon\}) \times Q$.

A generalised transducer T is said to be *functional* if τ_T is a partial function $X^* \to Y^*$. 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, $T = (Q, q_0, Q_F, E)$.

Doty and Moser work with a particular type of transducer: the *information lossless finite-state transducer*. In general, a transducer is information lossless if it represents an injective function from the input string x to the pair of the output and of the final state of the transducer on that input. In their paper, they write FST for the set of all finite-state transducers and ILFST for the set of all information lossless finite-state transducers.

They use ILFSTs as they have the following advantageous properties [7]:

Theorem 23 ([7], Theorem 3.1)**.** $\forall T \in$ *ILFST,* $\exists T^{-1} \in$ *ILFST,* $\exists c \in \mathbb{N}, \forall x \in \{0,1\}^* : x \upharpoonright (|x| - c) \sqsubseteq T^{-1}(T(x)) \sqsubseteq x$.[2]

Corollary 24 ([7], Corollary 3.2)**.** $\forall T \in$ *ILFST,* $\exists T^{-1} \in$ *ILFST,* $\forall S \in \mathbb{C} : T^{-1}(T(S)) = S$.

These results show that we can (partially, with c precision) recover an input string x for any given ILFST, by constructing its 'inverse' machine.

[2] See Appendix A for the details on the notation.

2.3.2 Doty and Moser's Notion of Depth

Finite-State Compression

Definition 25 ([7], k-FS Decompression Complexity)**.** *Let $k \in \mathbb{N}$ and $x \in \{0,1\}^*$,*

$$D_{FS}^{k}(x) = \min_{p \in \{0,1\}^*} \Big\{ |p| \;\Big|\; \exists T \in FST^{\leq k}, T(p) = x \Big\}, \tag{2.7}$$

where $FST^{\leq k}$ is the set of all FSTs such that the length of their binary representation is less or equal to k.

The above definition also shares similarities with Kolmogorov complexity; since a string's complexity relates to the size of the machine and of the program it is obtained from. Therefore, as claimed by Doty and Moser, it is a "finite-state analogue of Kolmogorov complexity" [7]. This particular definition and approach is what interests us in this dissertation. Ultimately, this is our goal: to effectively compute an analogous finite-state complexity to the Kolmogorov complexity. The authors of this paper have only briefly defined this analogue complexity, they do not offer any computed results, or prove that it is computable.

However, they obtain interesting results from their complexity, which relate to the concept of invariance. They state, and prove, that ILFSTs do significantly not alter the FS complexity of a string. Here are the two relevant lemmata.

Lemma 26 ([7], Lemma 3.3)**.** *Let $M \in ILFST$. $\exists c_1 \in \mathbb{N}, \forall k \in \mathbb{N}, \forall x \in \{0,1\}^*$:*

$$D_{FS}^{k+c_1}(M(x)) \leq D_{FS}^{k}(x). \tag{2.8}$$

Lemma 27 ([7], Lemma 3.4)**.** *Let $M \in ILFST$. $\exists c_2 \in \mathbb{N}, \forall k \in \mathbb{N}, \forall x \in \{0,1\}^*$:*

$$D_{FS}^{k+c_2}(x) \leq D_{FS}^{k}(M(x)). \tag{2.9}$$

Their FS complexity is a core concept to their notion of depth, which we introduce next.

Finite-State Depth

Definition 28 ([7], Finite-State Depth)**.** *A sequence S is* finite-state deep *if*

$$(\exists \alpha > 0)(\exists k' \in \mathbb{N})(\exists^{\infty} n \in \mathbb{N}) D_{FS}^{k}(S \upharpoonright n) - D_{FS}^{k'}(S \upharpoonright n) \geq \alpha n. \tag{2.10}$$

A sequence is finite-state shallow *if it is not finite-state deep.*

Basically, a sequence S is finite-state deep if given more time (a larger machine, therefore more steps to execute), it can always be decompressed further. In other words, a larger portion of S will be obtained with a shorter input for the transducer. This is a very similar concept to Bennett's and Juedes *et al.*'s notions

of depth, as expected. The depth of a sequence depends on the FS complexity of its initial segments. Moreover, since the FS complexity represents how much a finite string can be compressed via a k-sized FST, it represents some concept of depth for finite strings. It is therefore expected that we find the same aspects of depth discussed in the above sections of this chapter in finite-state depth.

In particular, we find that for finite-state depth, "trivial" (i.e. 'highly' recursive) and random sequences are finite-state shallow [7]. Doty and Moser reinforce this claim with their Proposition 3.6 [7], which we will not state here.

Finally, we can introduce the key results of Doty and Moser's paper, which are direct analogues to Bennett's work in a finite-state application.

2.3.3 Main Results of Feasible Depth

We note that each of the following theorems in this subsection have been extracted from Doty and Moser's work [7].

Theorem 29 (Finite-State Slow Growth Law)**.** *Let $S \in \mathbb{C}$, let $f : \mathbb{C} \to \mathbb{C}$ be ILFS-computable, and let $f(S) = S'$. If S' is finite-state deep, then S is finite-state deep.*

This theorem is proven using the two lemmata, Lemma 26 and Lemma 27, and then fitting the results to (2.10). Theorem 29 implies that a ILFST cannot build a deep sequence from a shallow sequence. Therefore, the 'hardness' of obtaining a deep sequence holds in this context, thanks to the property of invariance.

Theorem 30. *There exists a finite-state deep sequence.*

This is a crucial result as it makes this research non-trivial. Moreover, it increases the interest of computing theses complexities. The proof of this theorem can be found in *Feasible Depth* [7].

Chapter 3

Finite-State Descriptional Complexity

This chapter is heavily based on *Finite-State Complexity and Randomness* by Calude, Roblot and Salomaa [6]. In this chapter, we introduce the regular set of finite-state transducers which we have use in our work. We then present our finite-state complexity, based on this set of machines, and its remarkable properties. Finally, we offer some practical estimates of its measures and an algorithmic implementation of this complexity.

3.1 Regular Enumeration of Transducers

3.1.1 A Particular Definition for Finite Transducers

For our finite-state complexity model, we use a transducer where the corresponding transduction can be computed deterministically. A 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 represented by a function

$$\Delta : Q \times \{0,1\} \to Q \times \{0,1\}^*. \tag{3.1}$$

As we use mainly deterministic sequential transducers, we drop the adjectives, i.e., in the rest of the paper, unless stated otherwise, by all transducers are deterministic sequential transducers. When using the general model (2.6) we refer to them as generalised transducers.

For a transducer, all states are considered to be final. Thus, a transducer can be denoted by a triple (Q, q_0, Δ) where Δ comes from (3.1).

In details, the function $T : \{0,1\}^* \to \{0,1\}^*$ computed by the transducer (Q, q_0, Δ) is defined by

$$T(\varepsilon) = \varepsilon,\ T(xa) = T(x) \cdot \mu(\hat{\delta}(q_0, x), a),$$

where, $\delta(q, x) = \pi_1(\Delta(q, x))$,
$\mu(q, x) = \pi_2(\Delta(q, x))$,
for $q \in Q$, $x \in \{0,1\}^*$ and $a \in \{0,1\}$.[1]
Here, $\hat{\delta} : Q \times \{0,1\}^* \to Q$ is defined by:
$\hat{\delta}(q, \varepsilon) = q$,
$\hat{\delta}(q, xa) = \delta(\hat{\delta}(q, x), a)$,
for $q \in Q$ and $x \in \{0,1\}^*$.
Also, note that π is the standard notation for a projection function, such that $\pi_1(a, b) = a$ and $\pi_2(a, b) = b$.

3.1.2 A Regular Enumeration

We use binary strings to code transducers. By $\mathrm{bin}(i)$ we denote the binary representation of $i \geq 1$. Note that for $i \geq 1$, $\mathrm{bin}(i)$ always begins with a 1; i.e. $\mathrm{bin}(1) = 1, \mathrm{bin}(2) = 10, \mathrm{bin}(3) = 11, \ldots$

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 [5]): $v^\dagger = v_1 0 v_2 0 \cdots v_{m-1} 0 v_m 1$ and $v^\diamond = v_1 v_1 \cdots v_m v_m 01$. For example, $\varepsilon^\dagger = \varepsilon$, $0^\dagger = 01$, $1^\dagger = 11$, $(00)^\dagger = 0001$, $\varepsilon^\diamond = 01$, $0^\diamond = 0001$, $1^\diamond = 1101$, ...

We have: $|v^\dagger| = 2 \cdot |v|, |v^\diamond| = 2 \cdot |v| + 2$.

Consider a transducer T with the set of states $Q = \{1, \ldots, n\}$. The transition function Δ of T, as defined in (3.1), is encoded as the binary string

$$\sigma = \mathrm{bin}(i_1)^\ddagger \cdot u_{i_1}^\diamond \cdot \mathrm{bin}(i_2)^\ddagger \cdot u_{i_2}^\diamond \cdots \mathrm{bin}(i_{2n})^\ddagger \cdot u_{i_{2n}}^\diamond, \tag{3.2}$$

where $\Delta(j, k) = (i_{2j-1+k}, u_{i_{2j-1+k}})$, $j = 1, \ldots, n$ and $k \in \{0,1\}$. In (3.2), $\mathrm{bin}(i_t)^\ddagger = \varepsilon$ if the corresponding transition of Δ is a self-loop, i.e. $\Delta(j, k) = (i_t, u_{i_t}) = (j, u_{i_t})$; otherwise, $\mathrm{bin}(i_t)^\ddagger = \mathrm{bin}(i_t)^\dagger$.

Conversely, any string (3.2) encodes a unique transducer T_σ with n states when the (arbitrary) number represented by $\mathrm{bin}(i_r)$, $1 \leq r \leq n$, is replaced by the smallest positive integer congruent to $\mathrm{bin}(i_r)$ modulo n. The function computed by the transducer coded by σ is denoted by T_σ.

Let

$$\mathfrak{S} = \left\{ \mathrm{bin}(i_1)^{\ddagger}\cdot u_{i_1}^{\diamond} \cdots \mathrm{bin}(i_k)^{\ddagger}\cdot u_{i_k}^{\diamond} \;\middle|\; k \geq 2 \text{ is even},\ u_j \in \{0,1\}^*, i_j \in \mathbb{N}, j = 1,\ldots,k \right\} \tag{3.3}$$

be the set of all codes for transducers based on (3.2).

Proposition 31. *The set of all transducers can be enumerated by a regular language. More precisely, a) the set $\mathfrak{S}$ in (3.3) is regular, b) for every $\sigma \in \mathfrak{S}$, T_σ is a uniquely defined transducer and c) for every transducer T there exists (and can be computed) $\sigma \in \mathfrak{S}$ such that $T = T_\sigma$.*

Proof. a) The set $\mathfrak{S}$ is denoted by the regular expression $(((r+\varepsilon)s)^2)^*$ where r is a regular expression for $\{v^{\dagger} \mid v \in 1(0+1)^*\}$ and s is a regular expression for $\{v^{\diamond} \mid v \in (0+1)^*\}$.

b) Clearly any sequence of pairs (i_j, u_j), $i_j \in \mathbb{N}$, $u_j \in \{0,1\}^*$, $j = 1,\ldots,k$, for even k encodes a list of transitions of a transducer with $\frac{k}{2}$ states when each i_j is interpreted as a number of $1 + (\{0,\ldots,\frac{k}{2}-1\}$ modulo $\frac{k}{2})$. We need to verify that any string $\sigma \in \mathfrak{S}$ has a unique factorization as a concatenation of strings from the set $H = \{\mathrm{bin}(i_j)^{\ddagger} \cdot u_j^{\diamond} \mid i_j \geq 1, u_j \in \{0,1\}^*\}$.

To find the factorization of σ we look at the first two bits of σ. First, if the first two bits are 10, σ begins with a string $u^{\dagger}$, $u \in 1\{0,1\}^*$, which encodes a target state of the transition. The end of $u^{\dagger}$ is uniquely determined by the first bit of value 1 occurring in an even position. After this, the subsequent string $u_1^{\diamond}$ is uniquely determined by the first substring 01 starting from an even position.

Second, if the first two bits of σ are 00, 11 or 01, σ begins with a string of the form $v^{\diamond}$, $v \in \{0,1\}^*$, and the end of $v^{\diamond}$ is uniquely determined by the first occurrence of 01 starting in an odd position. Note that the binary numbers used as names of states do not have leading zeros and, hence, 00 or 01 cannot be a prefix of any state name. This case corresponds to a situation where the first string of H encodes a self-loop and omits the target state.

After having found the unique element of H that is a prefix of σ we continue in the same way with the remaining suffix of σ.

Finally, the claim c) is obvious. ∎

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

Example 32. Let $\Delta : \{1\} \times \{0,1\} \to \{1\} \times \{0,1\}^*$ be defined by $\Delta(1,0) = \Delta(1,1) = (1,\varepsilon)$. The code of T is $\sigma = \mathrm{bin}(1)^{\ddagger} \cdot \varepsilon^{\diamond} \cdot \mathrm{bin}(1)^{\ddagger} \cdot \varepsilon^{\diamond} = 0101$. ∎

A few more simple examples follow.

Example 33. The code of the transducer given by $\Delta_1 : \{1\} \times \{0,1\} \to \{1\} \times \{0,1\}^*$ where $\Delta_1(1,0) = (1,\varepsilon), \Delta_1(1,1) = (1,0)$ is $\sigma = \text{bin}(1)^\ddagger \cdot \varepsilon^\diamond \cdot \text{bin}(1)^\ddagger \cdot 0^\diamond = 010001$.

The code of the transducer given by $\Delta_2 : \{1\} \times \{0,1\} \to \{1\} \times \{0,1\}^*$ where $\Delta_2(1,0) = (1,0), \Delta_2(1,1) = (1,\varepsilon)$ is $\sigma = \text{bin}(1)^\ddagger \cdot 0^\diamond \cdot \text{bin}(1)^\ddagger \cdot \varepsilon^\diamond = 000101$.

The code of the transducer given by $\Delta_3 : \{1\} \times \{0,1\} \to \{1\} \times \{0,1\}^*$ where $\Delta_3(1,0) = \Delta_3(1,1) = (1,0)$ is $\sigma = \text{bin}(1)^\ddagger \cdot 0^\diamond \cdot \text{bin}(1)^\ddagger \cdot 0^\diamond = 00010001$.

The code of the transducer given by $\Delta_4 : \{1\} \times \{0,1\} \to \{1\} \times \{0,1\}^*$ where $\Delta_4(1,0) = \Delta_4(1,1) = (1,1)$ is $\sigma = \text{bin}(1)^\ddagger \cdot 1^\diamond \cdot \text{bin}(1)^\ddagger \cdot 1^\diamond = 11011101$. ■

Example 34. The identity transducer T is given by $\Delta : \{1\} \times \{0,1\} \to \{1\} \times \{0,1\}^*$, where $\Delta(1,0) = (1,0), \Delta(1,1) = (1,1)$. Its code is $\sigma = \text{bin}(1)^\ddagger \cdot 0^\diamond \cdot \text{bin}(1)^\ddagger \cdot 1^\diamond = 00011101$. ■

3.2 Finite-State Complexity

We use transducers to 'define' or 'represent' strings in the following way: a pair (T_σ, p), $p \in \{0,1\}^*$, defines the string x provided $T_\sigma(p) = x$. Then, the pair (T_σ, p) is called a *r*epresentation of x. We code the pair (T_σ, p) in a unique way by the binary string $\sigma^\dagger \cdot x$. Accordingly, we define the size of the representation (T_σ, p) of x, denoted by $||(T_\sigma, p)||$, by $|\sigma^\dagger \cdot x| = 2\,|\sigma| + |x|$.

Based on the above, we define the *finite-state complexity* of a string $x \in \{0,1\}^*$ by the formula:

$$\begin{aligned} C(x) &= \inf_{\sigma \in \mathfrak{S},\, p \in \{0,1\}^*} \Big\{ ||(T_\sigma, p)|| \;\Big|\; T_\sigma(p) = x \Big\} \\ &= \inf_{\sigma \in \mathfrak{S},\, p \in \{0,1\}^*} \Big\{ 2|\sigma| + |p| \;\Big|\; T_\sigma(p) = x \Big\}. \end{aligned}$$

Note that in our binary encoding of the language $\mathfrak{S}$, we need to, roughly, double the lengths of the strings in $\mathfrak{S}$. The choice of the encoding means that when a transducer T is used to encode a string x, a string v occurring as the output of a transition in T 'contributes' roughly $4 \cdot |v|$ to the size of an encoding $||(T, p)||$.

How 'objective' is the above definition? Our complexity is defined as an analogue of the complexity used in algorithmic information theory, 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, our definition of finite-state complexity counts the size of the transducer as part of the encoding length.[2] The reason is that there is no "universal" transducer. 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 ε.

Lemma 35. *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 $M_T \cdot |x|$.*

Theorem 36. *There does not exist a functional generalised transducer U such that for all $\sigma \in \mathfrak{S}$ and $w \in \{0,1\}^*$,*

$$U(\sigma^\dagger w) = T_\sigma(w).$$

Proof. For the sake of contradiction, assume that U exists and *wlog* we assume that the transitions of U are in the normal form of Lemma 22. Let M_U be the corresponding constant given by Lemma 35.

Let $\sigma_i \in \mathfrak{S}$, $i \geq 1$, 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 22, 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 $j \cdot g(k) = |T_{\sigma_j}(0^{g(k)})|$ and U does not simulate T_{σ_j} correctly. ∎

[2] One can use this approach also in algorithmic information theory [14].

We conjecture that there does not even exist a universal two-way finite transducer in the sense of Theorem 36:

Conjecture 37. *No two-way finite transducer can simulate an arbitrary transducer T_σ when it receives σ as part of the input.*

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 finite transducer with two two-way heads and one one-way head.

A weak form of universality can be proven for transducers:

Proposition 38. *For every strings $x, y \in \{0,1\}^*$ there exist infinitely many transducers T_σ 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 Δ having $n+1$ states acting as follows: $\Delta(i, \overline{x_i}) = (n+1, \varepsilon), 1 \leq i \leq n$, $\Delta(1, x_1) = (2, y)$, $\Delta(j, x_j) = (j+1, \varepsilon), 2 \leq j \leq n$, $\Delta(n+1, 0) = \Delta(n+1, 1) = (n+1, \varepsilon)$. ■

Corollary 39. *For every string $y \in \{0,1\}^*, \sigma \in \mathfrak{S}$, we have $\lim_{|\sigma| \to \infty} C_{T_\sigma} = \infty$.*

In spite of the negative result presented in Theorem 36, the analogue of the invariance theorem in algorithmic information theory is true for C. Accordingly, we define the complexity associated to a transducer T_σ by

$$C_{T_\sigma}(x) = \inf_{p \in \{0,1\}^*} \left\{ \|(T_\sigma, p)\| \;\middle|\; T_\sigma(p) = x \right\} = 2|\sigma| + \inf_{p \in \{0,1\}^*} \left\{ |p| \;\middle|\; T_\sigma(p) = x \right\}.$$

Theorem 40. *For every $\sigma_0 \in \mathfrak{S}$ we have $C(x) \leq C_{T_{\sigma_0}}(x)$, for all $x \in \{0,1\}^*$.*

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

$$\begin{aligned} C(x) &= \inf_{\sigma \in \mathfrak{S},\ p \in \{0,1\}^*} \left\{ \|(T_\sigma, p)\| \;\middle|\; T_\sigma(p) = x \right\} \\ &= \inf_{\sigma \in \mathfrak{S},\ p \in \{0,1\}^*} \left\{ 2|\sigma| + |p| \;\middle|\; T_\sigma(p) = x \right\} \\ &\leq 2|\sigma_0| + \inf_{p \in \{0,1\}^*} \left\{ |p| \;\middle|\; T_{\sigma_0}(p) = x \right\} \\ &= C_{T_{\sigma_0}}(x). \end{aligned}$$

■

Corollary 41. *If $T_{\sigma_0}(x) = x$, then $C(x) \leq |x| + 2|\sigma_0|$, for all $x \in \{0,1\}^*$. In particular, using Example 34 we deduce that $C(x) \leq |x| + 16$, for all $x \in \{0,1\}^*$.*

In contrast with the complexities used in algorithmic information theory by Corollary 41 we get:

Lemma 42. *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 an open problem in the binary case [2, 10].

Open problem 43. *How difficult is it to compute C?*

We use finite-state transducers to find 'regularities' in strings to obtain shorter definitions of those strings. In other words, regularities are used to *compress* strings.
The following example illustrates the power of transducer compression.

Example 44. Define the sequence of strings

$$w_m = 010^2 10^3 1 \cdot \ldots \cdot 0^{m-1} 10^m 1, \quad m \geq 1.$$

Using the transducer T_1 of Figure 3.1 we produce an encoding of w_{100}, where $|w_{100}| = 5150$.

With the encodings of the states indicated in Figure 3.1, T_1 is encoded by a string $\sigma_1 \in \mathfrak{S}$ of length 346. Each number $1 \leq i \leq 100$ can be represented as a sum of, on average, 3.18 numbers from the multi-set $\{1, 5, 10, 18, 25, 50\}$ [13]. Thus, when we represent w_{100} in the form $T_1(p_{100})$, we need on average at most 6×3.18 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

$$||(T_{\sigma_1}, p_{100})|| = 2 \cdot |\sigma_1| + |p_{100}| = 2700,$$

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

The above estimation could obviously be improved using more detailed information from the computation of the average from [13]. Furthermore, [13] 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.[3] 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 [8, 12].

[3] In [13] it is proved that 18 is the optimal value to add to an existing system of $\{1, 5, 10, 25, 50\}$.

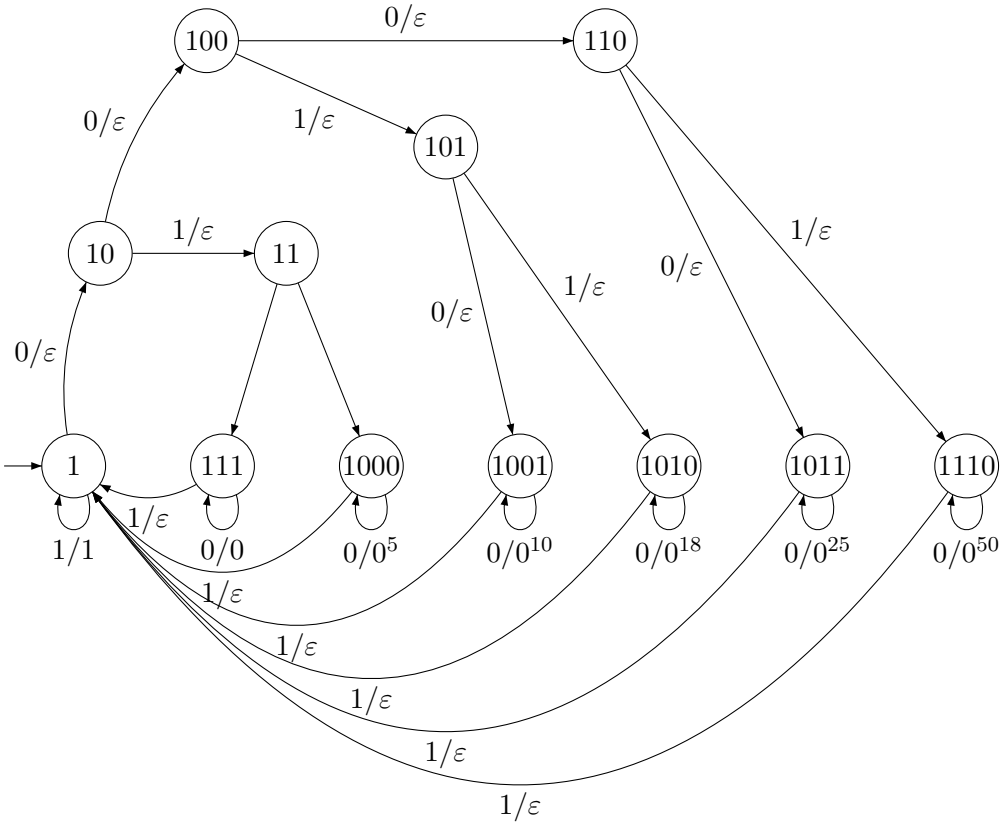

Figure 3.1: The transducer T_1 for Example 44.

3.3 Quantitative Estimates

Proposition 38 can be presented in a more precise manner:

Theorem 45. *For $n \geq 1$ we have: $C(0^n) \in \Theta(\sqrt{n})$.*

Proof. It is sufficient to establish that

$$4 \cdot \lfloor \sqrt{n} \rfloor \leq C(0^n) \leq 6 \cdot \lfloor \sqrt{n} \rfloor + c, \tag{3.4}$$

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 $6 \cdot \lfloor \sqrt{n} \rfloor + c$ 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 $||(T', p')|| \geq 4|v| + |p'|$. These

inequalities imply $||(T', p')|| \geq 4 \cdot \lfloor \sqrt{n} \rfloor$. ■

Using a more detailed analysis, the upper and lower bounds of (3.4) could be moved closer to each other. Because the precise multiplicative constants depend on the particular encoding chosen for the pairs (T, σ) and the language $\mathfrak{S}$, 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 (3.4) gives directly the following:

Corollary 46. *For any $x \in \{0,1\}^*$, $C(x) \geq 4 \cdot \lfloor \sqrt{|x|} \rfloor$.*

The bounds (3.4) imply that the inequality $H(xx) \leq H(x) + O(1)$ familiar to program-size complexity does not hold for finite-state complexity:

Corollary 47. *There is no constant c such that for all strings $x \in \{0,1\}^*$, $C(xx) \leq C(x) + c$.*

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

Corollary 48. *There is no constant c such that for all strings $x \in \{0,1\}^*$, $C(T(x)) \leq C(x) + c$.*

As in Theorem 45, we get estimations for the finite-state complexity of powers of a string.

Proposition 49. *For $u \in \{0,1\}^*$ and $n >> |u|$,*

$$C(u^n) \leq 4 \cdot (\lfloor \sqrt{n} \rfloor + 1) \cdot |u| + 2\sqrt{n} + c, \tag{3.5}$$

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 representation $(T, 0^{\lfloor \sqrt{n} \rfloor + y} 1^z)$, where $0 \leq y \leq 1$, $0 \leq z \leq \lfloor \sqrt{n} \rfloor$. By our encoding conventions

$$||(T, 0^{\lfloor \sqrt{n} \rfloor} 1^z)|| \leq 4 \cdot (\lfloor \sqrt{n} \rfloor + 1) \cdot |u| + 2\sqrt{n} + c,$$

where c is a constant. ■

The upper bound (3.5) 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 $C(u^n) \leq 4 \cdot |u| + n + c$, with c constant.

Corollary 50. *We have: $C(0^n 1^n) \in \Theta(\sqrt{n})$.*

Proof. The lower bound follows from Proposition 46. The string 0^n1^n has representation

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

where $0 \leq y_1, y_2 \leq 1$, $1 \leq z_1, z_2 \leq \lceil\sqrt{n}\rceil$ and T is the transducer given in Figure 3.2. Note that differing from the construction used in Theorem 45, the transducer in Figure 3.2 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 $z_1 \geq 1$ also when n is a perfect square.

Thus, $C(0^n1^n) \leq 12 \cdot \lceil\sqrt{n}\rceil + c$, where c is a constant. ∎

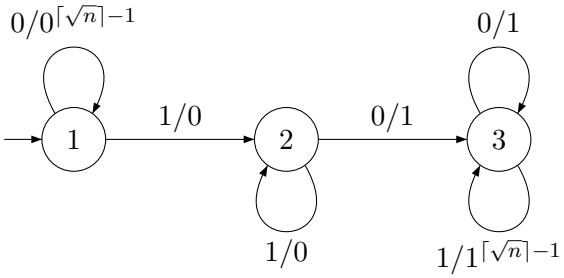

Figure 3.2: Transducer T in the proof of Corollary 50.

From Theorem 45, we know that transducers used in minimal encodings 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 encodings of arbitrary strings. This is in contrast to algorithmic information theory where the number of states of a universal Turing machine can be fixed.

Theorem 51. *$\forall n \in \mathbb{N}$, $\exists x_n \in \{0,1\}^*$ such that whenever $C(x_n) = ||(T_\sigma, p)||$ the transducer T_σ has more than n states.*

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

$$u_i = 10^i1^{2n+2-i}, \quad i = 1, \ldots, 2n+1.$$

Choose

$$m = 32n^2 + 68n + 29 \tag{3.6}$$

and define

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

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

$$||(T_\sigma, p)|| > \frac{m^2}{2}. \tag{3.7}$$

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

- θ_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;
- θ_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;
- θ_3 consists of transitions where the output does not contain any of the u_i's as a substring, $i = 1, \ldots, 2n+1$.

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

each transition of θ_3 used in the computation $T_\sigma(p)$ has length at most $4n+4$. (3.8)

Since T_σ 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 θ_1. We consider how the computation of T_σ on p outputs the substring $u_k^{m^2}$ of x_n. Let $z_1, \ldots, z_r$ 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 $u_k^{m^2}$.

We 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_σ 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 $4 \cdot |z_j|$ to the length of the encoding $||(T_\sigma, p)||$.

Finally, by (3.8), 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:

$$||(T_\sigma, p)|| \geq \sum_{j \in \Xi_2} 4 \cdot |z_j| + |\Xi_3| > \frac{|u_k^{m^2}|}{2|u_k|} = \frac{m^2}{2}.$$

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 3.3 and $p_0 = (0^m 1)^{2n-1} 0^m 1^m$.

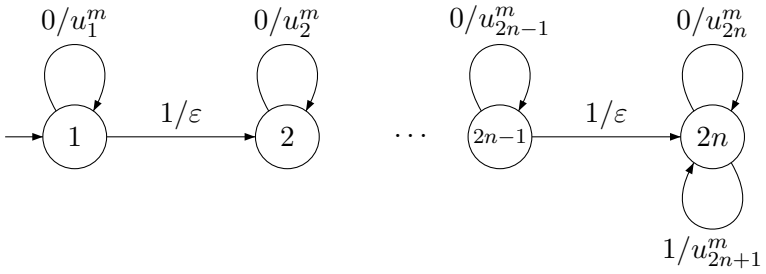

Figure 3.3: The transducer T_0 from the proof of Theorem 51.

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 $\sigma_0 \in \mathfrak{S}$ 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

$$||(T_{\sigma_0}, p_0)|| = 2|\sigma_0| + |p_0| = (16n^2 + 34n + 13) \cdot m + f(n),$$

and by using (3.6) we verify that the term $f(n)$ is less than m. Using again (3.6) we get

$$C(x_n) \leq ||(T_{\sigma_0}, p_0)|| < \frac{m^2}{2}. \tag{3.9}$$

■

Open problem 52. *It is an open problem whether, for every $n \in \mathbb{N}$, there exists a string x_n such that any minimal finite-state encoding of x_n has to use a transducer with exactly n states.*

Comment 53. Intuitively, the following type of 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 51, 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 representation $(T, 0^m 1^m 0^m)$ where T is the single-state transducer with two self-loops labeled by $0/0^m$ and $1/1^m$. Thus, $C(v_m) \leq 11 \cdot m + c$ where c is a constant independent of m.

We do not know how to construct a representation for u_m, $m \geq 1$, having size $11 \cdot m + O(1)$.

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

Proposition 54. *$\forall \varepsilon > 0$, $\exists d(\varepsilon) > 0$ such that $\forall x, y \in \{0,1\}^*$,*

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

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

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

We 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.

1. 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.

2. Each state $p' \in Q'_T$ has a transition labeled $1/\varepsilon$ to the starting state of R.

3. 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

$$W(\overline{u}v) = T(u)R(v) = xy.$$

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 together with 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) has the upper bound of $2 \cdot (\lceil \log(|Q_T|) \rceil)^{-1}$ (respectively, $2 \cdot (\lceil \log(|Q_R|) \rceil)^{-1}$). Thus, the proportional increase of the encoding length becomes smaller than any positive ε when $\max\{|Q_T|, |Q_R|\}$ is greater than a suitably chosen threshold $M(\varepsilon)$. 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 $d(\varepsilon) = 12 \cdot M(\varepsilon)$ the statement of the lemma holds also for small values of $|Q_T|$ and $|Q_R|$. ■

3.4 Closure Under Composition

It is known that deterministic transducers are closed under composition [4]. For transducers T_δ and T_γ, there exists $\sigma \in \mathfrak{S}$ 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, and 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 with $\Delta : Q \times \{0,1\} \to Q \times \{0,1\}^*$ and Γ is a function with $\Gamma : P \times \{0,1\} \to P \times \{0,1\}^*$. 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

$$T_\sigma = (Q \times P, \Xi),$$

where, $\Xi : Q \times P \times \{0,1\} \to Q \times P \times \{0,1\}^*$ is defined by setting for $q \in Q$, $p \in P$, $i \in \{0,1\}$,

$$\Xi((q,p),i) = \left(\left(\pi_1\left(\hat{\Delta}(q_1, \pi_2(\Gamma(p,i)))\right), \pi_1(\Gamma(p,i))\right), \pi_2\left(\hat{\Delta}(q, \pi_2(\Gamma(p,i)))\right)\right).$$

The number of states of T_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

[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.

$|\delta| \cdot |\gamma|$. These observations imply that

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

The above estimate was obtained 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.

3.5 Finite-State Incompressibility

As in the case of incompressibility in algorithmic information theory, we define the following concepts:

Definition 55. *A string x is* finite-state i–compressible*, for $i \geq 1$, if $C(x) \leq |x| - i$.*

Definition 56. *A string x is* finite-state i–incompressible*, for $i \geq 1$, if $C(x) > |x| - i$. If $i = 1$, then the string is called* finite-state incompressible*.*

Lemma 57. *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. ∎

Following is a notable fact about the asymptotic nature of incompressibility:

Fact 58. *For every string x and $i > 0$ there exists a string z such that $C(xz) > |xz| - i$.*

3.6 A Brute-Force Implementation

Finally, we would like to introduce our first implementation of the finite-state complexity presented in this dissertation. The pseudocode is as follows:

```
Procedure FSDComplexity(String x)
    Calculate upperBound for x's complexity
    for each string enumeration σ such that |σ| ≤ upperBound do
        if CorrectEncoding(σ) and ObtainsX(T, p, x) then
            return |σ|
        end if
    end for
```

```
    return false
end Procedure

Procedure CorrectEncoding(String σ)
    if σ has the correct pattern then
        T† = substring of σ up to first even positioned 1
        p = rest of σ
        Find each bin(i) and u_i in T and respectively assign each to the arrays:
            bins and outs
        if |bins| = |outs| and |bins| is even and |bins| > 0 then
            return true
        end if
    end if
    return false
end Procedure

Procedure ObtainsX(String T, String p, String x)
    for each bin(i) ∈ bins do
        if i = 0 or i ≥ |bins|/2 then
            return false
        end if
        add i to the transition array, trans
    end for
    String output = ϵ
    k = 0
    for each character p_i in p do
        if p_i = '0' then
            t = element of trans at k
            output = output + element of outs at k
        else
            t = element of trans at k + 1
            output = output + element of outs at k + 1
        end if
        k = 2 × (t − 1)
        end if
    end for
    if output = x then
        return true
    end if
    return false
end Procedure
```

We discuss each procedure separately.

Note that we have not included the pseudocode for the string enumeration procedure, as it is a known process. We will simply mention that the enumeration is done in lexicographical order, for every length, starting from the description of the identity transducer.

The *FSDComplexity* procedure is the core of the algorithm. It manages the enumerations and calls upon the other processes to check if we have reached the correct encoding for a desired pair and if this pair obtains our given input string. As the enumeration is in lexicographic order, the first such pair we find will clearly be the pair whose length is the complexity of our string. An open question comes into play.

Open problem 59. *Can and will some strings x correspond to several 'minimal' pairs (T, p)?*

There may be several pairs, of the same length, who obtain the same string. Therefore, further development in our implementation is to build a table, which will record the resulting complexities of the algorithm and allow us to empirically discover these strings, with multiple pairs describing them. Even though the resulting complexity would be the same, there may be distinctive properties between the pairs of such strings.

The *CorrectEncoding* procedure first checks the general pattern of the enumerated σ, to ensure it is in fact of the form $(T, p)^\dagger$. If this first check succeeds, it respectively assigns the parts of σ to $T^\dagger$ and p. It also, in that last process, checks whether or not $T^\dagger$ (from σ) is of the correct form. Finally, this procedure extracts the transition and the output functions from T. If there are $2n$ of them, then T is a correct encoding. Thus, σ was a correct encoding, according to our set $\mathfrak{S}$. Otherwise, the procedure will reject the current σ.

Our final procedure, *ObtainsX* uses the extracted functions from the last procedure and simulates the transducer T on the input p, then stores the output. When the simulation is complete, ObtainsX compares the obtained output with x, if they are identical, then we have found our pair (T, p). The success of the procedures is then returned to the main procedure FSDComplexity, which will return the length of this pair, completing the algorithm.

Chapter 4

Conclusion

4.1 Conclusions

In this dissertation, we give a literature review of three different papers covering the notion of depth from its original uncomputable conception, by Bennett in 1988, to its latest computable analogy, by Doty and Moser in 2007. Following Doty and Moser's example, we define a regular set of descriptions of finite-state transducers, from which we can define and compute the finite-state descriptional complexity of finite strings. We have had successful results both with the theoretical aspect of this research as well as the practical implementation. These findings have produced several new open questions, and we expect there are many more to come as we inspect the computational quality of this complexity and compare the results with its 'parent' complexity, Kolmogorov complexity.

4.2 Further Research

We have left to further work the modification of the current algorithm in order to construct a table of a wide range of strings, their complexities and their associated pairs, (T, p) (or σ's). This should reveal further properties and implications about the strings which have several associated pairs, or do not. Also, we have left to further research the implementation of an algorithm which does not take a brute-force approach and more effectively enumerates all the elements of our regular set $\mathfrak{S}$ of finite-state transducers.

In further research, we expect to explore and answer the open questions put forth in this dissertation. Hopefully, we will be able to explore in more depth the relations between this complexity and classical complexity, as well as where they differ. This research offers an important contribution in the field of algorithmic information theory. We expect it still has many promising results to unravel.

Bibliography

[1] S. Arora and B. Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, 2009. [cited at p. 4]

[2] J. Arpe and R. Reischuk. On the complexity of optimal grammar based compression. In *Proceedings of the Data Compression Conference, DCC'06*, pages 173–186, 2006. [cited at p. 21]

[3] C. H. Bennett. Logical depth and physical complexity. In Rolf Herken, editor, *The Universal Turing Machine - a Half-Century Survey*, pages 227–257. Oxford University Press, 1988. [cited at p. 1, 4, 5, 6, 8]

[4] J. Berstel. *Transductions and Context-free Languages*. Teubner, 1979. [cited at p. 10, 11, 15, 28]

[5] C. S. Calude. *Information and Randomness: An Algorithmic Perspective*. Springer, Berlin, 2nd edition, 2002. [cited at p. 16, 18]

[6] C. S. Calude, T. K. Roblot, and K. T. Salomaa. Finite-state complexity and randomness. 2009. [cited at p. 15]

[7] D. Doty and P. Moser. Feasible depth. *CoRR*, abs/cs/0701123, 2007. [cited at p. 1, 11, 12, 13]

[8] R. K. Guy. *Unsolved Problems in Number Theory*. Springer, Berlin, 3rd edition, 2004. [cited at p. 21]

[9] D. W. Juedes, J. I. Lathrop, and J. H. Lutz. Computational depth and reducibility. *Theoretical Computer Science*, 132:37–70, 1994. [cited at p. 1, 7, 8, 9, 10]

[10] E. Lehman. *Approximation algorithms for grammar-based compression*. PhD thesis, MIT, 2002. [cited at p. 21]

[11] M. Li and P. M. B. Vitányi. Logical depth. In *An Introduction to Kolmogorov Complexity and its Applications*, chapter 7. Resource-Bounded Complexity, pages 510–516. Springer, second edition, 1990. [cited at p. 3]

[12] J. Shallit. The computational complexity of the local postage stamp problem. *SIGACT News*, 33:90–94, 2002. [cited at p. 21]

[13] J. Shallit. What this country needs is an 18 cent piece. *Mathematical Intelligencer*, 25:20–23, 2003. [cited at p. 21]

[14] M. Sipser. *Introduction to the Theory of Computation*. Course Technology, second edition, 2005. [cited at p. 4, 19]

Appendices

Appendix A

Notation

ε	Empty string.
$\mathbb{N}$	Set of all non negative integers; the natural numbers.
$\mathbb{C} = \{0,1\}^\infty$	Cantor space.
TM	Turing Machine.
U	Efficient, fixed, prefix-free Universal TM (with time bound).
$\langle M \rangle$	(String) Description of a TM M.
p	Turing machine program, in this paper, $p = \langle M \rangle$.
$\|p\|$	Length of p.
$PROG_M(x)$	Set of programs for x relative to M.
$S[i..j]$	Substring of the i^{th} through j^{th} bits of S inclusive, where S is a string or sequence.
$S \upharpoonright n$	$= S[0..(n-1)]$; i.e. initial segments of S.
$S \sqsubseteq x$	where S is a string or sequence and x is a string, for all $n \in \mathbb{N}$, $x = S \upharpoonright n$.
REC	Set of recursive sequences; i.e. computable sequences.
RAND	Set of random sequences; i.e. uncomputable sequences.
wkDEEP	Set of weakly deep sequences.
strDEEP	Set of strongly deep sequences.
wkUSE	Set of weakly useful sequences.

List of Figures