An Investigation of Feasible Logical Depth and Complexity Measures via Automata and Compression Algorithms

LIAM JORDON

A thesis submitted for the degree of

Doctor of Philosophy

Department of Computer Science

Maynooth University

Supervisor: Dr. Philippe Moser

Second Supervisor: Dr. Phil Maguire

Head of Department: Dr. Joseph Timoney

February 2022

Contents

Abstract

When presented with a string or sequence of zeros and ones, that is an element of $\{0,1\}^{\leq\omega}$, it is often of interest to know how complex the object is. Was it created from some simple process or was it generated randomly? Kolmogorov Complexity is a fundamental tool of Algorithmic Information Theory which measures the complexity of such objects. However, there is one major problem: Kolmogorov Complexity is uncomputable. As such, the complexity of such objects are often studied in lower computable settings. This thesis aims to extend the study of such objects at lower complexity levels via finite-state automata, transducers and compression algorithms. We pay particular attention to normal sequences.

One measurement which relies on Kolmogorov Complexity is Bennett's Logical Depth, which has been described in the past as measuring how useful an object is. The primary objective of this thesis is to examine feasible notions of Bennett's depth. We first extend an already studied infinitely often finite-state notion of depth to an almost everywhere notion. Secondly, we develop new notions based on pushdown compressors, the Lempel-Ziv 78 compression algorithm, and pebble transducers. We demonstrate the existence of deep sequences in each of these notions, and show which properties of Bennett's original notion they satisfy. We also determine the differences between some of the depth notions we examine as follows: For a pair of depth notions (A, B), we construct a sequence S such that its A-depth level and B-depth level are unequal, i.e. S is 'more' deep in one notion than the other. This demonstrates that there is not a single unifying

notion of depth.

Our secondary objective is to examine normal sequences at lower complexity levels. Normal sequences are of interest to us as they are often considered finite-state random as they cannot be compressed by any information lossless finite-state compressor. We do this by, when appropriate, identifying normal sequences which are deep in the depth notions we develop. Furthermore, we prove the existence of a normal sequence which can be compressed by the PPM* compression algorithm which is incompressible by the Lempel-Ziv 78 algorithm. This, to our knowledge, is the first example of a mathematical proof differentiating the two algorithms on specific inputs as opposed to experimental results. We conclude by giving examples of normal sequences which have non-maximal automatic complexity, a complexity measurement based on finite-state automata.

Acknowledgements

To start with, I would like to sincerely thank my supervisor Dr. Philippe Moser for his guidance and support during this journey. Your depth of knowledge has been invaluable. Your dry wit and sense of humour was always thoroughly appreciated, and very much needed at times. I regret that our time together for in-person meetings was cut short.

Secondly, I'd like to thank Dr. Phil Maguire for coming aboard on short notice as supervisor for my final months. From my time in Computational Thinking, to giving me my first experience as a researcher as part of SPUR, to now, you have always been full of enthusiasm and encouragement. Thank you for the positivity.

I'm grateful to all the staff of Maynooth University's Department of Computer Science for their help during my studies. Thank you for the working atmosphere you provided, aiding me with my enquiries, and trusting in me to tutor students over the years. A special thanks goes to Dr. Joseph Timoney for offering to proof read a near final draft of the thesis. A similar thanks goes to Mr. Joseph Duffin for being a cheerleader of mine during the final half of my studies during the pandemic.

I'm similarly indebted to all the staff of the Department of Mathematics and Statistics for providing me with a home for my first five years in Maynooth. I wouldn't be where I am today without my days spent in Logic House.

I would like to acknowledge the Irish Research Council's Government of Ireland Postgraduate Scholarship Programme's (Grant: GOIPG/2017/1200) role in

providing the funding for this research. This thesis would not have been possible without it.

To all the friends I made along the way during my time in Maynooth, whether we knew each other for several years or just a semester, I love you all. From *Fun Tuesdays* to *Cook Tuesdays*, from *News of the Day* to *#IFTYBACY*, I had an absolute blast! The CTs, the TPs, the Mathsies, the Eds, the M9s, MUMS Social Club, the folks in the MSC/CS355/CS370, my fellow CS postgrads, the regulars from nights out in the Roost/Mantra/Brady's, the lads who came with me to Maynooth from secondary school - there are too many of you to name! A deserving shout-out also goes to the supportive friends from outside of the university who helped me greatly during tough times. Saying all that, I would like to specifically name Aisling, Alice, Emmet, and Jack for keeping me somewhat sane and grounded during the lockdowns. Thanks for putting up with my rants, raves, and woe is me stories on a weekly basis.

Most importantly, to my parents and to my brother John, thank you for your love, support, and belief in me for the past many years. You had faith in me when I often had none. Similar thanks also go towards my two wonderful grandparents, Raymond and Cella, and to my Aunt Colette for their love and support.

Finally, to Holly, Ivy, and Gabby, thanks for making me smile. 😸😸😸

Declaration

I hereby declare that this thesis is my own work. I confirm that this thesis has not been submitted in any form for another degree or diploma at any university or other institution of tertiary education. Information derived from the published or unpublished work of others has been acknowledged in the text and a list of references is given.

Liam Jordon - February 2022

List of Publications

The following is a list of accepted and submitted papers based on the work of this thesis:

- **International Conferences:**
  - Liam Jordon and Philippe Moser: On the Difference between Finite-State and Pushdown Depth. In: *46th International Conference on Current Trends in Theory and Practice of Informatics, SOFSEM 2020, Limassol, Cyprus, January 20–24, 2020, Proceedings*, LNCS, vol. 12011, pp. 187–198. Springer (2020). doi: 10.1007/978-3-030-38919-2_16
  - Liam Jordon and Philippe Moser: A Normal Sequence Compressed by PPM* but not by Lempel-Ziv 78. In: *47th International Conference on Current Trends in Theory and Practice of Computer Science, SOFSEM 2021, Bolzano-Bozen, Italy, January 25–29, 2021, Proceedings*, LNCS, vol. 12607, pp. 389–399. Springer (2021). doi: 10.1007/978-3-030-67731-2_28
  - Liam Jordon and Philippe Moser: Normal Sequences with Non Maximal Automatic Complexity. In: *41st IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2021, Gao, India, December 15-17, 2021, Proceedings.* LIPIcs, vol. 213, pp. 47:1-47:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik (2021). doi: 10.4230/LIPIcs.FSTTCS.2021.47

- **Journals in Submission / Prepared:**
  - Liam Jordon and Philippe Moser: Pushdown and Lempel-Ziv Depth, Under review at *Information and Computation*, Submitted August 2020. Preprint: arXiv.
  - Liam Jordon and Philippe Moser: Pebble Depth, Under review at *Theoretical Computer Science*, Submitted December 2020. Preprint: arXiv.

Chapter 1

Introduction

1.1 Background

When presented with a piece of data, such as a sequence of ones and zeroes, it is often asked how complex the data is. Are there patterns in the data which can be easily spotted? Similarly, are there hidden regularities in the data which are more difficult to notice? Was the data created from some simple quick process which allows us to recreate the data with relative ease or was it created by some long complicated process? Does the data simply appear random to the casual observer or is it truly random?

An intuitive way to quantify how complex a piece of data is by measuring how difficult it is to describe. Suppose we had a set of descriptions X, some method to read and comprehend the descriptions R, and some target set of data Y, and assume each description in X maps to at most one piece of data in Y via R. Then, for each piece of data y in Y, we can measure the complexity of the data y by examining the length of the all descriptions from X which are mapped to y via R. It is therefore reasonable to define the complexity of y as being the length of the description which is the shortest among all of the descriptions which map to y. Intuitively, the simpler y is, the shorter the descriptions for it would need

to be. The more complex y is, the longer the descriptions for it need to be. Of course in such a scenario, the chosen method to translate the descriptions R plays a role in the complexity of the objects.

Kolmogorov complexity, denoted by K, is a descriptional complexity based on this idea. Developed by Solmonoff in 1960/64 [137, 138, 139], Kolmogorov in 1965 [93] and Chaitin in 1966/69 [36, 37], and further expanded to a prefix-free notion, denoted by H, by Levin in 1974 [100] and Chaitin again in 1975 [38], it has acted as a foundational tool of measurement in computer science and mathematics. It has a wide number of applications in the field of Algorithmic Information Theory [61, 104, 118].

One of the drawbacks of Kolmogorov complexity is that it is solely interested in the length of short descriptions. In a sense it simply measures how difficult a sequence is to describe, i.e. how random it is. It provides little insight into the time and effort required to produce the data. In this thesis we focus much of our attention to its use in Charles Bennett's measurement called *Logical Depth* [17] which attempts to overcome this weakness.

In contrast to Kolmogorov complexity, Bennett's depth also takes into account the minimum time taken to produce a piece of data from short descriptions. Bennett's goal was to create a notion which could measure *physical* complexity. The more complex an object is, the longer it must have taken to be created. Such objects are called *deep* while non-deep objects are called *shallow*. For instance, human beings may be considered deep structures as our current bodies are the result of millions of years of evolution.

Intuitively an infinite sequence is deep if its prefixes take a long time to be produced from their short descriptions. In particular, we are interested in the minimum length of time taken as one can have a program which runs for a long time by having it wait an arbitrarily long period before beginning its computation. From this, it is easy to understand the three fundamental properties Bennett's

depth satisfies. Firstly, *random* sequences are not deep. Randomness here means Martin-Löf-random, (denoted as ML-random) by the equivalent definitions of Martin-Löf [108], Chaitin [38], Levin [99], and Schnorr [126, 127]. Intuitively, ML-random sequences are those such that the lengths of the shortest descriptions of its prefixes are roughly equal to the lengths of the prefixes themselves. Hence a simple `print` program can generate prefixes of random sequences quickly from their shortest description. Secondly, trivial sequences are not deep. This is because the patterns contained in trivial sequences are easy to identify and thus not computationally expensive to create. In Bennett's case, trivial sequences are the computable ones. Thirdly, Bennett's depth satisfies a *Slow Growth Law*. This means that deep sequences must be difficult to produce, one cannot transform a shallow structure into a deep structure via a quick and easy process. Bennett's depth being invariant under truth-table reductions demonstrates this [17].

Every object which is Bennett deep is intrinsically *useful* in the sense of Juedes, Lathrop and Lutz [86]. This is best demonstrated by the diagonal Halting Problem whose characteristic sequence $\emptyset'$ is such that its n^{th} bit is 1 if and only if the n^{th} Turing Machine M_n (in some enumeration of all Turing Machines) halts on input n. While it is famously uncomputable [142], it is an extremely useful problem in computability and complexity theory as it NP-hard. While an n bit prefix of $\emptyset'$ has low Kolmogorov complexity, since by Barzdin's Lemma [8] the prefix can be reproduced from a description of $O(\log n)$ bits long, Bennett showed it is deep [17]. In contrast to this, Chaitin's constant Ω [38] which contains the exact same information as $\emptyset'$ (in fact they are weak truth table equivalent [30]) is not deep. Ω contains this information in a much more compressed manner which makes it unfeasible to extract information from, thus it making it useless. In fact, Ω is ML-random [16, 38].

Kolmogorov complexity is unfortunately uncomputable. In fact, it is not even partially computable in the sense that no partial computable function with infinite

domain exists which coincides with Kolmogorov complexity on every element of its domain [158]. A proof deriving the uncomputability of Kolmogorov complexity via the uncomputability of the Halting Problem can be found in [39]. As a result, many authors have devised approaches to either estimate Kolmogorov complexity or have substituted with more computable approaches.

One approach is to estimate Kolmogorov complexity from above via monotonic decreasing functions [158]. *Resource Bounded* Kolmogorov Complexities which limit the amount of time Turing Machines are allowed to run and the amount of space they are allowed to use for memory is another popular approach. Early examples of this can be found in [50, 91, 101, 134].

In [29] it is shown that regardless of what optimal universal machine is used as a description method, computable functions exist which map infinitely many strings to their shortest programs for both the plain and prefix-free version of Kolmogorov complexity.

List Approximations is a method where for each string x, a list of programs which can reproduce x are computed such that one of the programs has length close to the Kolmogorov complexity of x [9, 10, 141, 155].

Using known values of Radó's Busy Beaver function [122] calculated by Brady in [27], the *Coding Theorem Method* has been used to estimate the Kolmogorov complexity for short strings (up to length 16) in [54, 136, 154]. Briefly, for a universal prefix-free machine U, the algorithmic probability of x is the sum $\mathbf{m}(x) = \sum_{p:U(p)=x} 2^{-|p|}$. Intuitively, this is the probability that the universal machine U will produce x if fed random bits as input. The Coding Theorem provides a relationship between prefix-free Kolmogorov complexity, denoted by H, and $\mathbf{m}(x)$ from which it can be deduced that $H(x) = -\log(\mathbf{m}(x)) + O(1)$ [38, 100]. Exploiting this relationship, the authors of [54, 136, 154] numerically estimate $\mathbf{m}(x)$ by running simulations on machines of small size to see if they produce x.

Other approaches such as by Becher and Heiber [13] and Lempel and Ziv [98] measure the complexity of a finite string x by the number of substrings x contains under certain parsing conditions.

Many authors have replaced Turing Machines with finite-state transducers to define descriptional finite-state based complexities [31, 34, 56, 95]. The common theme in these notions is that the complexity of a string x is measured based on the shortest input y into some finite-state transducer T such that $T(y) = x$.

Other finite-state based approaches are more concerned with the number of states in the finite-state machines examined. In [32], Calude, Salomaa and Roblot examine the number of states required to output strings based on their minimal finite-state descriptional complexity as defined in [31]. Similarly, Shallit and Wang introduced a notion called *Automatic Complexity* [130] which is a distinguishing based complexity analogous to Sipser's *Distinguishing Kolmogorov complexity* [134]. Shallit and Wang's notion differs from Sipser's in that they replace Turing Machines with finite-state automata. It measures the minimal number of states required such that for a string x of length n, x is the only string of length n that some finite-state automaton accepts. For a function f from strings to strings, *Automaticity* is another notion which measures the number of states required by a finite-state transducer T such that f and T agree on all inputs of length n or less [73, 120, 129]. Automaticity can be used to define an analogous measurement to automatic complexity which examines languages of strings instead of a single string.

The Kolmogorov complexity of a piece of data can be viewed as a lower bound for how much the data can be compressed such that it is still retrievable from this compressed form. As such, some authors have used compression algorithms to approximate Kolmogorov complexity. One place where compressors replaced Kolmogorov complexity was in the development of the *Similarity Metric* between two pieces of data which, as the name suggests, measures how similar two pieces

of data are [43, 103]. This idea has been applied to many areas such as to detect plagiarism [42], to compare pieces of music [44], and to monitor fetal heartbeats [48]. Many more examples are reported in [43]. Compression algorithms they used included `gzip` which is based on the Lempel-Ziv 77 algorithm [156], the PPMZ variant [21] of Cleary and Witten's original PPM algorithm [45], and `bzip2` which is based on the Burrows Wheeler transform [28].

With regards to Bennett's Logical Depth, various authors have developed new formulations of depth by replacing Kolmogorov complexity with their own chosen complexity notions. One of the first was Lathrop and Lutz's *Recursive Computational Depth* which replaced Kolmogorov complexity with computable time bounded Kolmogorov complexity [96]. While computable, some of the time-scales are still impractical to work with. Subsequently Antunes et al. introduced various notions in [5] which avoid impractical time bounds, one of which is based off polynomial time bounded distinguishers. It is possible that this notion of depth is redundant, however this is unlikely as it would imply that factorisation is in the complexity class P.

In an attempt to address this problem, Doty and Moser restricted themselves to define a depth notion based solely on finite-state transducers in [57]. However, their notion defines a sequence to be deep if infinitely many of its sequences satisfy a depth property, which is in contrast to Bennett's notions which is an 'all but finitely many' prefixes notion.

Moser and Stephan replace prefix-free Kolmogorov complexity with the plain version in [114], but this has the same uncomputable problem. So too does their notion called *Limit-depth* which provides access to $\emptyset'$ as an oracle [115]. This idea of providing access to an oracle has been expanded upon in a recent preprint by Bienvenu et al. [20].

Other approaches include Doty and Moser replacing Kolmogorov complexity with polynomial time predictors in [57], and Moser replacing it with polynomial

monotone compressors [112] and with polylog time bounded Kolmogorov complexity [113].

An experimental approach by Zenil, Delahaye and Gaucherel to estimate the logical depth of images based the run time of decompression algorithms on compressed images can be found in [153]. Further experimental attempts by Gaucherel to examine the logical depth of ecosystems can be found in [71].

In light of these examples, it seems reasonable to consider other approaches of defining depth which avoid impractical time bounds and the uncomputability of Kolmogorov complexity, and to explore their characteristics.

Borel *normal* [25] sequences also play a central role in this thesis. These are sequences such that if they are constructed from an alphabet of size k it holds that for all n, every string of length n occurs as a substring in the sequence with asymptotic frequency k^{-n}. With regards to Kolmogorov complexity, every sequence which is ML-random must be normal. However, the converse is not true.

An important result regarding normal sequences is that they characterise all sequences which have a *finite-state dimension* (Definition 3.2.6) of 1 in the sense of Dai et al. [49]. This means that normal sequences may be considered finite-state random as, for instance, they are incompressible by any information lossless finite-state compressor [13]. As such, the complexity of normal sequences has been studied in various compression based and finite-state based settings to see whether they have maximal complexity in these other settings [4, 11, 34, 35, 95, 97].

In particular, Becher, Carton and Heiber [11] have examined the compression of normal sequences in scenarios such as when finite-state compressors have access to one or more counters or a stack, and what happens when the compressor is not required to run in real-time nor be deterministic. They showed that normal sequences are either incompressible or that a compressible sequence exists based on the combination chosen. Carton and Heiber showed that deterministic

and non-deterministic two-way finite-state compressors cannot compress normal sequences [35]. Lathrop and Strauss showed that every sequence incompressible by the Lempel-Ziv 78 algorithm must be normal and that the converse is not true in [97]. Pierce and Shields gave examples of normal sequences which are compressible by a variation of the Lempel-Ziv 77 algorithm in [119].

1.2 Objectives of This Thesis

In this thesis we shall consider complexity notions based on finite automata and popular compression algorithms and examine the complexity of various sequences in each notion. Particular attention will be paid to normal sequences. By using finite automata and compression algorithms, we shall avoid the problems of the uncomputability of Kolmogorov complexity and of using impractical time bounds.

The first area of investigation of this thesis shall be to focus on variants on Bennett's depth. Using Moser's general framework for logical depth [112], we shall define new feasible notions of depth based on automata and compressors.

While defining a new approach of depth is simple based on Moser's framework, checking whether or not that approach is meaningful is not as straightforward. The primary goal shall be to demonstrate that each of the depth notions developed are in a sense sound and worthwhile in that they satisfy all (or a subset) of the properties of Bennett's original definition. That is, for each of the notions we explore:

1. Do deep sequences exist?

2. Are trivial sequences non-deep?

3. Are random sequences non-deep?

4. Does the depth notion satisfy a slow growth type law?

Secondly, it is important to demonstrate that each of the depth notions explored have their own individual properties, i.e. they are not all equivalent. Hence for each notion, the *depth* of a sequence shall be defined. Using this, we shall attempt to differentiate two notions by demonstrating the existence of sequences which have high depth in one notion and low depth in the other, and vice versa. The notions we shall examine will be based on finite-state transducers, pushdown compressors, the Lempel-Ziv 78 compression algorithm, and pebble transducers. With regards to normal sequences, we aim to show whether or not deep normal sequences exist in each notion we explore.

As part of this exploration of depth, we shall compare the performance of different compression algorithms against each other on particular sequences. Continuing with this line of study, we follow up studying depth with an investigation into the Prediction by Partial Matching (PPM) family of compressors. Firstly we shall attempt to differentiate the PPM* compression algorithm from the Lempel-Ziv 78 algorithm by presenting a sequence which PPM* can compress which Lempel-Ziv 78 cannot. We also simultaneously aim to answer the question as to whether or not there exists normal sequences which PPM* can compress. A comparison of PPM* and its original counterpart, Bounded PPM, is also performed.

As part of our investigation into new depth notions, for the finite-state and pebble based approaches, the complexity of a string x will be defined based on the length of inputs required to output x. In conjunction with this, an alternative finite-state based complexity known as automatic complexity is also examined at the end of this thesis. As a finite-state automata based complexity, one may assume normal sequences must have maximal automatic complexity since they have a finite-state dimension of one. We aim to answer the question as to whether or not normal sequences must have maximal automatic complexity.

1.3 Outline of the Thesis

The thesis is structured as follows:

- In Chapter 2 we shall outline some relevant background material on Kolmogorov complexity and Bennett's Logical Depth. We shall briefly examine Moser's framework for depth and how it can be used to define new depth notions. We also will explore other topics such as normality which will appear frequently throughout the thesis.

- Chapters 3 through 6 explore new feasible notions of depth:

  - In Chapter 3 we present an almost everywhere notion of finite-state depth which is inspired by Doty's and Moser's infinitely often finite-state depth notion [57]. We demonstrate the existence of a deep sequence and examine which of the fundamental depth properties our notion satisfies. We also establish a difference between our notion and Doty and Moser's original notion by constructing a sequence which is deep in their notion but not in ours.

  - In Chapter 4 we present a new notion of depth based on information lossless pushdown compressors. We demonstrate the existence of a deep sequence and examine which of the fundamental depth properties our pushdown notion satisfies. We demonstrate its difference from Doty and Moser's finite-state depth by constructing a sequence which is finite-state deep but not pushdown deep. We also construct sequences which have a pushdown depth level of close to $1/2$ and finite-state depth level of at most $1/2$.

  - In Chapter 5 we present a notion of depth based on the Lempel-Ziv 78 compression algorithm [157]. We establish the existence of

a Lempel-Ziv deep normal sequence and examine which of the fundamental depth properties our notion satisfies. We separate it from finite-state depth by constructing Lempel-Ziv deep sequences which are not finite-state deep and vice versa. Furthermore we demonstrate the existence of Lempel-Ziv deep sequences which are not pushdown deep and sequences which have a pushdown depth level of close to 1/2 and a Lempel-Ziv depth level of at most 1/2.

- In Chapter 6 we derive a depth notion based on pebble transducers. Such machines were first viewed as acceptors by Globerman and Harel in [74]. We establish the existence of a pebble deep normal sequence and examine which of the fundamental depth properties our pebble notion satisfies. We exhibit pebble depth's difference from Lempel-Ziv depth by showing the existence of sequences with a pebble depth level of close to 1/2 and a Lempel-Ziv depth level of at most 1/2. We also begin a preliminary investigation to differentiate pebble depth from pushdown depth.

- In Chapter 7 we examine the Prediction by Partial Matching (PPM) family of compression algorithms. We demonstrate that the unbounded version of the algorithm known as PPM* developed by Cleary and Teahan [46] can fully compress a Champernowne style sequence. This demonstrates a difference with PPM* and the Lempel-Ziv 78 algorithm as Champernowne style sequences are worse case inputs for the algorithm. We also show that the original Bounded PPM algorithm of Cleary and Witten [45] cannot compress normal sequences.

- In Chapter 8 we shall examine Shallit and Wang's automatic complexity [130] to see if normal sequences must have maximal automatic complexity. We will answer this in the negative by constructing a sequence whose pre-

fixes have lower and upper automatic complexity ratios bounded between 0 and 1/2. We shall also demonstrate the existence of a normal sequence, specifically a Champernowne sequence, whose prefixes have an automatic complexity ratio bounded above by 2/3.

- Concluding remarks are given in Chapter 9.

Chapter 2

Preliminaries and Background

In this chapter we cover some relevant notation and ideas used throughout the thesis. We also provide a more detailed insight into topics mentioned in the introductory chapter.

2.1 Preliminaries

We work with the binary alphabet $\{0,1\}$ in this thesis. A (finite) *string* is an element of $\{0,1\}^*$. Strings will generally be denoted by lowercase letters. $\{0,1\}^n$ denotes the set of strings of length n. Thus $\{0,1\}^* = \bigcup_{n=0}^{\infty}\{0,1\}^n$. The set of (infinite) *sequences* is denoted by $\{0,1\}^\omega$. Sequences will generally be denoted by an uppercase letter. $\{0,1\}^{\leq\omega} = \{0,1\}^* \cup \{0,1\}^\omega$ denotes the set of all strings and sequences. $|x|$ denotes the length of the string x. We say $|S| = \infty$ for a sequence $S \in \{0,1\}^\omega$. We use λ to denote the *empty string*, that is, the string of length 0. For $x \in \{0,1\}^*$ and $y \in \{0,1\}^{\leq\omega}$, xy (occasionally written as $x \cdot y$) denotes the string (or sequence) of x concatenated with y. For $x \in \{0,1\}^*$, x^n denotes the string of x concatenated with itself n times, i.e. $x^n = \overbrace{x \cdot x \cdots x}^{n \text{ times}}$, and similarly, x^ω denotes the sequence composed of x concatenated with itself infinitely many times. For $x \in \{0,1\}^{\leq\omega}$ and $0 \leq i < |x|$, $x[i]$ denotes the i^{th} character of x

with the leftmost character being $x[0]$. For $x \in \{0,1\}^{\leq\omega}$ and $0 \leq i \leq j < |x|$, $x[i..j]$ denotes the *substring* of x consisting of the i^{th} through j^{th} bits of x. For $x \in \{0,1\}^*$ and $y, z \in \{0,1\}^{\leq\omega}$ such that $z = xy$, we call x a *prefix* of z and y a *suffix* of z. We write $x \sqsubseteq v$ if x is a prefix of v and $x \sqsubset v$ is x is a prefix of v but $x \neq v$. For $x \in \{0,1\}^{\leq\omega}$ we write $x \upharpoonright n$ to denote the prefix of length n of x, i.e. $x \upharpoonright n = x[0..n-1]$. For a string x and for $0 \leq j < |x|$, we write $x[j..]$ to denote the suffix of x beginning with bit $x[j]$.

For a string $x \in \{0,1\}^*$, we write $d(x)$ to denote the string formed by doubling every bit of x, that is, if $x = x_1 \dots x_m$, then $d(x) = x_1 x_1 \dots x_m x_m$. For a string $x \in \{0,1\}^*$, we write x^{-1} to denote the *reverse* of x, that is, if $x = x_1 \dots x_m$ where for each i $x_i \in \{0,1\}$, then $x^{-1} = x_m \dots x_1$.

The *lexicographic ordering* of strings is defined as follows. Given two strings x and y, if $|x| = |y|$, we say that x comes before y if for the least n such that $x[n] \neq y[n]$, $x[n] = 0$ and $y[n] = 1$. Else y comes before x. Similarly suppose $|x| < |y|$. If $x \sqsubset y$, then x comes before y. Else, let x' be the string $x0^{|y|-|x|}$. Note that $|x'| = |y|$. Then, if x' comes before y we say that x comes before y. Else y comes before x.

The *lexicographic-length ordering* of strings is defined as follows. Given two strings x and y, x comes before y if $|x| < |y|$ or for the least n such that $x[n] \neq y[n]$, $x[n] = 0$ and $y[n] = 1$. The *standard enumeration of strings* is the enumeration of strings in lexicographic-length order.

Much of the following is relevant background in computability theory. More background details can be found in the introductory texts by (but not limited to) Downey and Hirschfeld [61], Li and Vitányi [104] and Nies [118].

In general, we will consider functions $f : A \to \{0,1\}^*$ where $A \subseteq \{0,1\}^*$, i.e. *partial* functions on $\{0,1\}^*$. The *domain* of f, denoted by $\text{dom}(f)$, is the set A. If $A = \{0,1\}^*$, then f is called a *total* function. Intuitively, a function being computable means that on a given input, we can calculate the function's

output by following a finite number of steps, or in other words, by following a terminating algorithm. Specifically we define the *computable* functions to be the partial functions which can be computed by *Turing Machines*, as introduced by Turing in [142]. For more details and a history on the relationship between the intuitively (partial) computable functions and (partial) functions computed by Turing Machines, otherwise known as the Church-Turing Thesis, see Soare [135].

Definition 2.1.1. Let $A \subseteq \{0,1\}^*$ and $f : A \to \{0,1\}^*$ be a function. f is *partial computable* if there exists a Turing machine M such that for all $x \in A$, $f(x) = y$ if and only if M on input x halts and outputs y, i.e. $M(x) = y$. We say f is *computable* if it is *partial computable* and its domain is $\{0,1\}^*$.

We generally refer to Turing machines simply as machines.

Given a machine M, string x and some $s \in \mathbb{N}$, $M(x)[s]$ denotes running M on x for s steps of its computation. Note in this scenario, M's computation may not have halted. To clarify this, $M(x)[s]\downarrow$ denotes running M on x and that its computation halts within s steps. $M(x)\downarrow$ means that M eventually halts on x.

Occasionally we examine partial functions from $\mathbb{N}$ to $\mathbb{N}$ instead of over strings. In this case, we can simply pair each element of $\mathbb{N}$ with an element of $\{0,1\}^*$ by pairing n with the n^{th} string in lexicographic-length order. For example, λ is paired with 0 and 000 is paired with 7. Note that if n is paired with s_n in the ordering, then $|s_n| = \lfloor \log(n+1) \rfloor$.

Definition 2.1.2. A *time bound* is a non-decreasing, unbounded, computable function $f : \mathbb{N} \to \mathbb{N}$.

Definition 2.1.3. A machine M has *oracle* access to the sequence $X \in \{0,1\}^\omega$ if M has an additional *oracle tape* such that for all $n \in \mathbb{N}$, its n^{th} square has $X[n]$ written on it. M can query the oracle tape anytime during its computation to correctly answer questions of the form 'What is $X[n]$?'. M is referred to as an

oracle (Turing) machine. If M has oracle access to X, M^X is written instead of M.

One can similarly have a finite string x on the oracle tape. That is, for $0 \leq n < |x|$, the n^{th} square contains $x[n]$ and for $n \geq |x|$, the n^{th} square is left blank.

Definition 2.1.4. Let $X, Y \in \{0,1\}^\omega$.

- We say that X is *(Turing) reducible to* Y, denoted by $X \leq_{\text{T}} Y$, if there exists an oracle machine M^Y such that for all $n \in \mathbb{N}$, $X[n] = M^Y(n)\downarrow$.
- For a time bound $t : \mathbb{N} \to \mathbb{N}$ we say that X is *(Turing) reducible to* Y in time t, denoted by $X \leq_{\text{T}}^t Y$, if there exists an oracle machine M^Y such that for all $n \in \mathbb{N}$, $X[n] = M^Y(n)[t(|s_n|)]\downarrow$.

For $Y \in \{0,1\}^*$ and time bound $t : \mathbb{N} \to \mathbb{N}$, let

$$\text{DTIME}^Y(t) = \{X \in \{0,1\}^\omega : X \leq_{\text{T}}^t Y\}$$

denote the set of sequences Turing reducible to X in time t.

2.2 Kolmogorov Complexity

In the following subsection we review some of the basics of the descriptional complexity known as Kolmogorov Complexity.

Given a string x and a machine M, a common question is whether x is a string that can be outputted by M. That is, does there exist a string p such that $M(p) = x$? Such a p is referred to as an *M-description* of x. Quite often we are interested in the shortest p that when inputted into M gives x. From this, we define the *M-complexity* of x.

Definition 2.2.1. For a machine M, the *M-complexity* of a string $x \in \{0,1\}^*$ is defined to be the length of a shortest string p such that $M(p) = x$. The notation used is

$$K_M(x) = \min\{|p| : M(p) = x\}.$$

If no such p exists we say $K_M(x) = \infty$.

If $|p| < |x|$, then p can be viewed as a compressed version of x and M as the decompressor that when given p re-obtains x.

Example 2.2.2. Consider the identity machine I, i.e. the machine such that for all $x \in \{0,1\}^*$, $I(x) = x$. Then $K_I(x) = |x|$ for all x.

Instead of examining the complexity of strings on a machine by machine basis, it is often useful to examine the complexity of a string via an *optimal* machine.

Definition 2.2.3. A machine N is called an *optimal* machine if for each machine M, there exists a constant $c_M \in \mathbb{N}$ such that for all $x \in \{0,1\}^*$

$$K_N(x) \leq K_M(x) + c_M.$$

In his paper [142], Turing shows that each of his machines has a *standard description* composed of a string of characters which fully describes how the machine operates. Thus, enumerating all strings and examining which ones correspond to a standard description of a machine, one can enumerate all machines. This in turn means the partial computable functions can be enumerated in some order too. This enumeration gives rise to the existence of an optimal machine.

Theorem 2.2.4. *There exists an optimal machine.*

Proof. Let $M_1, M_2, M_3, \ldots$ be an enumeration of all machines. Consider the machine N such that $N(0^{e-1}1\sigma) = M_e(\sigma)$ for all $e > 0$ and strings σ. Consider the

machine M_d and $x, y \in \{0,1\}^*$ such that $M_d(y) = x$ and $K_{M_d}(x) = |y|$. Hence, $N(0^{d-1}1y) = M_d(y) = x$ and so $K_N(x) \leq K_{M_d}(x) + d$.

□

In the above proof $0^{d-1}1$ can be thought of as a pointer to the machine M_d. It should be noted that optimal machines are sometimes referred to as *universal machines*.

We can now define the *(plain) Kolmogorov complexity* of a string. We fix some optimal machine U.

Definition 2.2.5. The *(plain) Kolmogorov complexity* of a string x is defined to be the U-complexity of x.

In other words, the Kolmogorov complexity of x is the value $K_U(x)$. As the choice of U only impacts Kolmogorov complexity up to an additive constant [93], we write $K(x)$ instead of $K_U(x)$.

When concatenating two strings together, one would hope for nice results such as $K(xy) \leq K(x) + K(y) + d$, for some constant d. That is, the shortest description of the string xy can be found simply from the shortest description of x and the shortest description of y. Unfortunately, equalities such as this do not hold for all strings (see Corollary 2.2.2 of [118]). If we simply concatenate the two shortest descriptions, one cannot tell where the description of x ends and the description of y begins. This is one of the many reasons why *prefix-free* machines are studied.

Definition 2.2.6. A machine M is *prefix-free* if for all $x, y \in \mathrm{dom}(M)$, if $x \sqsubseteq y$ then $x = y$.

When referring to the *M-complexity* of a string x when M is prefix-free, we write H_M instead of K_M. As with the plain version, there exists a prefix-free optimal machine [38], i.e. there exists a prefix-free machine V such that for each

prefix-free machine M, there exists a constant c_M such that for all $x \in \{0,1\}^*$, $H_V(x) \leq H_M(x)+c_M$. We can now define the *(prefix-free) Kolmogorov complexity* of a string. We fix some optimal prefix-free machine V.

Definition 2.2.7. The *(prefix-free) Kolmogorov complexity* of a string x is defined to be the V-complexity of x.

As with the plain version, the choice of V only impacts Kolmogorov complexity up to an additive constant. Hence we write $H(x)$ instead of $H_V(x)$.

We note that other authors, such as in [61, 118], use C to denote the plain version of Kolmogorov complexity and K to denote the prefix-free version. We however use K and H instead respectively as we reserve C to denote compressors later in this thesis. Chaitin, for instance, uses H to denote the prefix-free version in [38].

Both K and H are not computable. The following theorem strengthens this result further.

Theorem 2.2.8 ([158])**.** *K is not partial computable. Furthermore, no partial computable function f exists such that $|\mathrm{dom}(f)| = \infty$ and for all $x \in \mathrm{dom}(f)$, $f(x) = K(x)$. This is similarly true for H.*

Time Bounded Kolmogorov Complexity

We are also interested in the restricted form of Kolmogorov Complexity known as *Time Bounded Kolmogorov Complexity*. Intuitively, we aim to find the shortest description of a string if we are given a finite number of steps to identify it.

Definition 2.2.9. Given a computable function $t : \mathbb{N} \to \mathbb{N}$, for all $x \in \{0,1\}^*$ the *(plain) t-time bounded Kolmogorov complexity* of x, denoted by $K^t(x)$, is given by

$$K^t(x) = \min\{|p| \ : \ U(p)[t(|x|)] \downarrow = x\}.$$

If no such p exists we say $K^t(x) = \infty$.

In other words, $K^t(x)$ is the shortest program for which U has halted and outputted x within $t(|x|)$ steps. $H^t(x)$ for the prefix-free version is similarly defined.

2.2.1 Distinguishing Complexity

In 1983, Sipser introduced a variant of Kolmogorov complexity known as *Distinguishing complexity* [134]. Intuitively, while $K(x)$ is the shortest program which generates x, its distinguishing complexity $KD(x)$ is the length of the shortest program which 'distinguishes' x from all other strings. In Chapter 8 we will examine a computable variant of Distinguishing complexity. Sipser's definition is as follows.

Definition 2.2.10 ([134])**.** Let $t : \mathbb{N} \to \mathbb{N}$ be a time bound and $x \in \{0,1\}^*$. The *Distinguishing complexity* of x is given by

$$KD^t(x) = \min\{|p| : \forall y \in \{0,1\}^* \left[U^y(p)[t(|y|)] \downarrow \wedge (U^y(p) = 1 \iff y = x)\right]\}.$$

Note if no time bounds are used, distinguishing complexity differs from Kolmogorov complexity by an additive constant [134].

2.3 Bennett's Depth

In his original paper, Bennett devised a notion of depth for both strings and sequences [17]. We shall briefly examine the notion for strings but our main focus shall be on the notion for sequences.

Understanding the philosophical idea of simplicity which states that given multiple plausible explanations for an event, the one with the fewest assumptions should be the preferred explanation acts as the basis of Bennett's depth. Extending this to strings, given a string x, we may prefer to assume that it was

generated by a string of length equal to $H(x)$. However, as Bennett noted, it is possible to have programs with length slightly longer than $H(x)$ which generate x much faster than a program with length $H(x)$. Thus, Bennett examines the length of time taken to generate x from programs with length close to their minimal descriptions. The longer it takes to generate x from such programs, the more 'evolved' or complex x can be thought of as being. More specifically, Bennett says that for a string x, its depth at significance level c is the minimum time taken to produce x from a c-incompressible program, i.e. from a string y whose minimal description is no more than c bits shorter than the y itself.

Definition 2.3.1 ([17])**.** Let $x \in \{0,1\}^*$ and $c > 0$. The *(Bennett) logical depth* of x at *significance level* c is defined as

$$depth_c(x) = \min_{p \in \{0,1\}^*} \{t(|x|) : V(p)[t(|x|)] \downarrow = x \wedge |p| - H(p) < c\}.$$

Note that $depth_c(x)$ is decreasing in c.

Definition 2.3.2. Let $t \geq 0$, $c > 0$ and $x \in \{0,1\}^*$. x is *t-deep* at significance level c if $depth_c(x) \geq t$. Otherwise x is *t-shallow* at significance level c.

A slight variation of Definition 2.3.1[1] is explored in [6] (with a correction in [144]) where it is shown that changing the significance parameter by 1 can drastically change the depth of a string. In particular it is shown that a sequence of strings of increasing length $x_1, x_2, x_3, \ldots$ can be built such that as a function of n, this difference in depth grows faster than any computable function. The same result holds using Definition 2.3.1 if the class of Turing Machines used is restricted [145].

Bennett also provides a notion of depth for infinite sequences. Specifically he introduces two notions, a *strongly* and *weakly* deep notion. We will be more

[1]The definition used is $depth_c(x) = \min_{p \in \{0,1\}^*} \{t(|x|) : p \in \{0,1\}^* \wedge V(p)[t] \downarrow = x \wedge |p| - H(x) < c\}$. Bennett presents it as tentative Definition 0.2 in [17].

interested in the strongly deep notion which states that an infinite sequence is strongly deep if for every computable time bound $t : \mathbb{N} \to \mathbb{N}$ and every significance level c, all but finitely many prefixes of S are t-deep at significance level c. Several equivalent definitions are also mentioned in [17, 86] one of which is as follows.

Definition 2.3.3. Let $S \in \{0,1\}^\omega$. S is *(Bennett) strongly deep* if for all computable time bounds $t : \mathbb{N} \to \mathbb{N}$ and all $c \geq 0$ it holds that for all but finitely many n

$$H^t(S \upharpoonright n) - H(S \upharpoonright n) \geq c.$$

Bennett's strong depth satisfies three basic properties. Firstly, random sequences are not deep. Randomness here means ML-randomness after Martin-Löf. While there are several equivalent characterisations of ML-randomness [38, 108, 99, 126, 127], we shall give the definition based on prefix-free Kolmogorov complexity.

Definition 2.3.4. $S \in \{0,1\}^\omega$ is *ML-random* if there is a constant $c \in \mathbb{N}$ such that for all $n \in \mathbb{N}$ it holds that $H(S \upharpoonright n) > n - c$.

Theorem 2.3.5 ([17, 86])**.** *If $S \in \{0,1\}^\omega$ is ML-random then S is not strongly deep.*

The second property is that computable sequences are not deep.

Definition 2.3.6. $S \in \{0,1\}^\omega$ is *computable* if there exists a Turing machine M such that for every non-negative integer n, M on input n computes the n^{th} bit of S, i.e. $M(n) \downarrow = S[n]$.

Definition 2.3.7. $S \in \{0,1\}^\omega$ is *computably enumerable* if the set $A = \{n : S[n] = 1\}$ is the domain of some machine M, i.e. $\text{dom}(M) = A$.

Theorem 2.3.8 ([17, 86])**.** *If $S \in \{0,1\}^\omega$ is computable then S is not strongly deep.*

The third property is the Slow Growth Law. This says that quick processes cannot transform shallow sequences into deep ones. This is demonstrated by depth being invariant under truth table reductions. Truth table reductions are equivalent to time bounded Turing reductions (see Proposition 1.2.22 of [118]). We present the definition of truth table reductions based on this fact.

Definition 2.3.9. We say that $X \in \{0,1\}^\omega$ is *truth-table reducible* to $Y \in \{0,1\}^\omega$, written as $X \leq_{\text{tt}} Y$, if there exists a computable time bound $t : \mathbb{N} \to \mathbb{N}$ such that $X \in \text{DTIME}^Y(t)$. That is, there exists an oracle machine M^Y such that for all $n \in \mathbb{N}$, $X[n] = M^Y(n)[t(|s_n|)] \downarrow$.

Theorem 2.3.10 (Slow Growth Law [17, 86])**.** *Let $X, Y \in \{0,1\}^\omega$. If $X \leq_{tt} Y$ and X is strongly deep, then Y is strongly deep.*

Bennett himself noticed how his strong depth notion related to the idea of a sequence being *useful*. Specifically, Bennett noted that the characteristic sequence $\emptyset'$ of the Halting problem is strongly deep [17, 86]. It is an extremely useful sequence in the sense that any computably enumerable sequence can be computed by an oracle machine with access to $\emptyset'$ in polynomial time.

Juedes, Lathrop, and Strauss explicitly described the relationship between usefulness and Bennett's strong depth in [86]. They gave two notions of usefulness, called *weakly useful* and *strongly useful*.

Definition 2.3.11. Let $S \in \{0,1\}^\omega$.

- S is *weakly useful* if there exists a computable time bound $t : \mathbb{N} \to \mathbb{N}$ such that the set $\text{DTIME}^S(t)$ has non-zero measure in the set of all computable sequences.

- S is *strongly useful* if there exists a computable time bound $t : \mathbb{N} \to \mathbb{N}$ such that the set $\text{DTIME}^S(t)$ contains every computable sequence.

Without going into details, here by measure we mean the resource bounded measure developed by Lutz [105, 106] which is a generalisation of Lebesgue measure theory. Intuitively, S is weakly useful if it means that a non-negligible subset of the computable sequences can be efficiently computed via the help of S. Similarly, strongly useful means all of the computable sequences can be computed in a time bound via the help of S. Juedes, Lathrop, and Strauss proved the following fact.

Theorem 2.3.12 ([86])**.** *Every weakly useful sequence is strongly deep.*

The converse is not true as it has been shown that there exists strongly deep sequences which are not weakly useful [96]. Weakly useful sequences are further explored in [66].

2.3.1 Moser's General Framework for Depth

Many variations of depth have been studied and new ones are introduced and compared in this thesis. While all different, they share a common theme. Moser provides a general framework for defining any depth notion in [112] which we will discuss now.

As Moser points out, depth notions can be defined relative to two classes of functions G and G' on strings. G and G' may be any type of function, such as decompressors or compression algorithms. Depth is then defined based on how much information a member of class G can extract from the sequence compared to members of class G'.

For instance, suppose three monolingual speakers who understand English, French and Arabic respectively are presented with a piece of text written in English which is aimed at the general public. In this case the English speaker should be able to extract all available information from the text. On the other end of the spectrum, as Arabic uses a different alphabet from English, to the

Arabic speaker the text will appear to be a random jumble of symbols which are meaningless. In between, as French and English both use the same Latin alphabet, there may exist certain combinations of letters the French person may guess the meaning of and so can extract some of the information from the text. For instance, the French for 'table' is 'tableau'. Likewise, many French words are also used in English such as 'faux pas'.

More formally, let G, G' be two classes of competing functions on strings. Let $\text{Perf} : G \times G' \times \{0,1\}^* \to [0,1]$ be some function which measures how much better an algorithm $A' \in G'$ performs on an input string compared to an algorithm $A \in G$. Here, 0 means they perform as well as each other while 1 means A' optimally outperforms A. For instance, Perf may compute how much more A' can compress the string x compared to A. Let M be a family of computable functions where for all $m \in M$ and every positive integer n, $1 \leq m(n) \leq n$. M is used to measure how well A' performs compared to A. Then, with G, G', and M, one can define a depth notion.

Definition 2.3.13 (Framework for Depth [112])**.** A sequence $S \in \{0,1\}^\omega$ is *almost everywhere (a.e.)* (G, G', M)*-deep* if

$$(\forall m \in M)(\forall A \in G)(\exists A' \in G')(\forall^\infty n \in \mathbb{N})\, \text{Perf}(A, A', S \upharpoonright n) \geq \frac{m(n)}{n}.$$

Infinitely often (i.o.) (G, G', M)*-depth* is similarly defined by replacing the $\forall^\infty$ term in the above definition with $\exists^\infty$.

Many of the choices when developing a depth notion can seem arbitrary, for instance whether to have an a.e. or i.o. notion. Similarly one may choose to say 'there exists a bound $m \in M$' instead of 'for all bounds $m \in M$'. Another choice may say 'for all observers $A \in G$, there exists a bound $m \in M$' instead of 'for all bounds $m \in M$ and for all observers $A \in G$'. However, the challenge remains to show that the depth notions are meaningful in a sense that they satisfy results

analogous to Theorems 2.3.5, 2.3.8 and 2.3.10.

The following examples are how Moser's framework captures some depth notions.

Example 2.3.14. Bennett's strong depth satisfies Moser's framework with $G = \{H^t : t \text{ a time bound}\}$, $G' = \{H\}$, $M = \mathbb{N}$ and

$$\text{Perf}(H^t, H, x) = \frac{H^t(x) - H(x)}{|x|}.$$

Example 2.3.15. Stephan and Moser explore a plain version of Bennett's depth in [114] where $G = \{K^t\}$, $G' = \{K\}$ and various choices for M are examined.

Example 2.3.16. Stephan and Moser explore their notion called *limit depth* in [115] where $G = \{H\}$, $G' = \{H^{\emptyset'}\}$ and $M = \mathbb{N}$. Here, $H^{\emptyset'}(x)$ is the prefix-free Kolmogorov complexity of x in the sense of Definition 2.2.7 except the optimal prefix-free machine V becomes an oracle machine in the sense of Definition 2.1.3 with access to $\emptyset'$.

Example 2.3.17. In an effort to overcome the uncomputability of H, Moser defines a *polylog depth* notion in [113]. A variant of plain Kolmogorov Complexity from [3] is used, denoted by KT, which allows for logarithm time bounds. In particular, for a string x and a time bound $t : \mathbb{N} \to \mathbb{N}$ where $t(n) \geq \log n$, for all n we define

$$KT^t(x) = \min\{|p| : (\forall b \in \{0, 1, \lambda\})(\forall i \leq n)\, U^p(i, b)[t(n)] = \text{accepts iff}\, x[i] = b\}$$

where $n = |x|$, for all $i \geq |x|$ $x[i] = \lambda$, and $U^p(i, b)[t(|x|)]$ means U with oracle access to p and on input of the pair (i, b) runs for $t(|x|)$ steps. Note here that on input (i, b), U only needs to identify the value of $x[i]$ and by requiring that $U(|x|, b) = \lambda$, U identifies the length of x.

Let $\mathrm{PL} = \{c \log^c n : c \in \mathbb{N}\}$. Polylog depth is then defined using the framework where $G = \{KT^t : t \in \mathrm{PL}\}$, $G' = \{KT^s : s \in 2^{\mathrm{PL}}\}$, $M = \mathrm{PL}$ and

$$\mathrm{Perf}(KT^t, KT^s, x) = \frac{KT^t(x) - KT^s(x)}{|x|}.$$

2.4 Descriptional Transducer Based Complexity

Throughout this thesis (particularly in Chapters 3 and 6) we will use computable complexity approaches analogous to K based on a special class of partial computable functions over strings which can be computed by finite-state based transducers. Definitions of the transducers we use can be found in Definitions 3.2.1 and 6.2.1. For a transducer $T : \{0,1\}^* \to \{0,1\}^*$ and a string x, we will be interested in finding the minimum length of input required for T to output x, i.e. the value $\min\{|y| : y \in \{0,1\}^* \wedge T(y) = x\}$. Similar approaches encapsulating this theme to define finite-state complexity have previously been explored in [31, 34, 56, 57, 95].

In particular, we will be interested in restricting the sizes of the transducers we are examining also. This leads to the question as to how to define the size of a transducer. Probably the most intuitive approach would be to say that the size of a transducer is defined as the number of states that make up the transducer. The main problem with this is that for any string x, one can easily build a transducer with one state which outputs the string x for every bit of its input. This would result in every string having a complexity of 1.

Instead, for a class of transducers F, we will associate each transducer $T \in \mathrm{F}$ with a set of binary strings which fully describe T. Then the size of T will be the length of the minimal string associated with it. This approach to define the size

of transducers has been previously taken in [31, 34, 56, 57]

Definition 2.4.1. Let F be a class of transducers and $D \subseteq \{0,1\}^*$ be an infinite, computable set of strings. A *binary representation of* F*-transducers* σ is a computable map $\sigma : D \to \mathrm{F}$, such that for every transducer $T \in \mathrm{F}$, there exists some $x \in D$ such that $\sigma(x) = T$, i.e. σ is surjective. If $\sigma(x) = T$, we call x a σ *description* of T.

For a binary representation of F-transducers σ, we define

$$|T|_\sigma = \min_{x \in \mathrm{dom}(\sigma)} \{|x| : \sigma(x) = T\}$$

to be the *size of* T *with respect to* σ. For all $k \in \mathbb{N}$, define

$$\mathrm{F}_\sigma^{\leq k} = \{T \in \mathrm{F} : |T|_\sigma \leq k\}$$

to be the set of F-transducers with a σ description of size k or less. For all $k \in \mathbb{N}$ and $x \in \{0,1\}^*$, the *k-F complexity* of x with respect to binary representation σ is defined as

$$D_\sigma^k(x) = \min\left\{|y| \ : \ T \in \mathrm{F}_\sigma^{\leq k} \ \wedge \ T(y) = x\right\}.$$

Here, y is the shortest string that gives x as an output when inputted into an F-transducer of size k or less with respect to the binary representation σ. In later chapters we fix binary representations for the classes of transducers examine and will instead write $D_{\mathrm{F}}^k(x)$ in place of $D_\sigma^k(x)$.

We will use the following two ratios in Definition 2.4.2 to measure the compressibility of sequences with respect to a class of transducers. We use the term compressibility instead of randomness to avoid confusion with Definition 2.3.4.

Definition 2.4.2. Let $S \in \{0,1\}^\omega$ and F be a class of transducers. The *upper*

and *lower compressibility rates* of S with respect to the class F are given by

$$\rho_{\mathrm{F}}(S) = \lim_{k\to\infty} \liminf_{n\to\infty} \frac{D^k_{\mathrm{F}}(S \upharpoonright n)}{n}, \text{ and } R_{\mathrm{F}}(S) = \lim_{k\to\infty} \limsup_{n\to\infty} \frac{D^k_{\mathrm{F}}(S \upharpoonright n)}{n}$$

respectively.

Definition 2.4.3. Let $S \in \{0,1\}^\omega$ and F be a class of transducers. We say S is F$-$*trivial* if $R_{\mathrm{F}}(S) = 0$ and F$-$*incompressible* if $\rho_{\mathrm{F}}(S) = 1$.

2.4.1 Compression Algorithms

Similarly to classes of transducers, we will also be examining the complexity of sequences with respect to classes of compression algorithms in Chapters 4, 5 and 7. For a compression algorithm $C : \{0,1\}^* \to \{0,1\}^*$, the complexity of a string x will be based on the length of the string that C maps x to, i.e. the length of $|C(x)|$. We will use the following two ratios to measure the randomness of sequences with respect to a compression algorithm.

Definition 2.4.4. Let $S \in \{0,1\}^\omega$ and $C : \{0,1\}^* \to \{0,1\}^*$ be a compression algorithm The *best-case* and *worst-case compression ratios* of S via C are given by

$$\rho_C(S) = \liminf_{n\to\infty} \frac{|C(S \upharpoonright n)|}{n}, \text{ and } R_C(S) = \limsup_{n\to\infty} \frac{|C(S \upharpoonright n)|}{n}$$

respectively.

Instead of examining the compression ratio via a single compression algorithm, occasionally we will also be interested in the compression ratio of sequences over a class of compression algorithms.

Definition 2.4.5. Let $S \in \{0,1\}^\omega$ and F be a class of compression algorithms. For a sequence S, the *best-case* and *worst-case compression ratios* of S with

respect to the class F are respectively given by

$$\rho_{\mathrm{F}}(S) = \inf\{\rho_C(S) : C \in \mathrm{F}\}, \text{ and } R_{\mathrm{F}}(S) = \inf\{R_C(S) : C \in \mathrm{F}\}$$

Definition 2.4.6. Let $S \in \{0,1\}^\omega$ and F be a class of compression algorithms. We say S is F$-$*trivial* if $R_{\mathrm{F}}(S) = 0$ and F$-$*incompressible* if $\rho_{\mathrm{F}}(S) = 1$.

2.5 Normal Sequences

In most chapters of this thesis we will examine the complexity and depth of a special class of sequences known as *normal sequences*. They were first defined by Borel in 1909 [25].

Prior to defining normal sequences, we require the following notation to represent the number of occurrences of a particular substring in a string.

Definition 2.5.1. Let $x, w \in \{0,1\}^*$.

1. The *block* number of occurrences of w in x is given by

$$\mathrm{occ}_b(w, x) = |\{i : x[i..i + |w| - 1] = w \wedge i \equiv 0 \bmod |w|\}|.$$

2. The *total* number of occurrences of w in x is given by

$$\mathrm{occ}(w, x) = |\{i : x[i..i + |w| - 1] = w\}|.$$

For instance $\mathrm{occ}(00, 0000) = 3$ while $\mathrm{occ}_b(00, 0000) = 2$. One way of characterising normal sequences is as follows.

Definition 2.5.2. A sequence $S \in \{0,1\}^\omega$ is *normal* if for all $x \in \{0,1\}^*$ it holds that

$$\lim_{n\to\infty} \frac{\mathrm{occ}(x, S \upharpoonright n)}{n} = 2^{-|x|}.$$

The above definition is a 'non-aligned' version of normality. The $\text{occ}(x, S \upharpoonright n)$ term could be replaced with $\text{occ}_b(x, S \upharpoonright n)$ to define an 'aligned' version. However, both versions along with Borel's original definition are equivalent. Kozachinskiy and Shen provide details on the history of showing these definitions are equivalent in Section 3 of [95]. It can easily be adapted to define normal numbers in any base k.

Due to their 'balanced' statistics, for a sequence to be ML-random, it must be normal. Intuitively this is true as if a sequence S were not normal, there exists some substring x of length n which appears more frequently than all other strings of length n in S. Hence a Turing Machine could be created which exploits this fact to compress prefixes of S.

Lemma 2.5.3. *If S is ML-random then S is normal.*

However, not all normal sequences are ML-random. The first sequence proven to be normal (in base 10) was the decimal sequence

$$S = 012345678910111213..$$

by Champernowne in 1933 [40]. This sequence was the concatenation of all non-negative integers in order of magnitude. It is clearly not ML-random as all you need to produce a prefix of the sequence is to know the length of the prefix to be constructed. Of course, this construction can be generalised to any base to produce a base-k normal sequence. This idea of listing numbers in order to form normal sequences can be generalised as follows. Given an infinite set $B \in \mathbb{Z}^+$, Copeland and Erdös provide the following condition that B must satisfy so that the sequence formed by concatenating the base-k representations of the elements of B in order is normal in base-k.

Theorem 2.5.4 ([47])**.** *Let B be an infinite set of positive integers. The sequence*

formed by concatenating the base-k representations of the elements of B in order of magnitude is normal in base-k if for all $\varepsilon < 1$, for all n sufficiently large it holds that $|B \bigcap \{1, 2, \ldots, n\}| > n^{\varepsilon}$.

The above theorem can be used for instance to show that the sequence

$$P = 23571113\ldots$$

formed by listing the prime numbers in order is a decimal normal number.

This idea of listing all strings in order of length classifies a subset of normal sequences which we call *Champernowne* sequences.

Definition 2.5.5. A sequence $C \in \{0,1\}^{\omega}$ is a *Champernowne* sequence if $C = C_1C_2C_3...$ such that

$$(\forall n \in \mathbb{N})(\forall x \in \{0,1\}^n)\, \mathrm{occ}_b(x, C_n) = 1.$$

Note in the above that for all n, $|C_n| = n \cdot 2^n$. Unlike the binary version of Champernowne's original sequence which was a concatenation of all strings in lexicogrpahic-length order (0100011011000...), we emphasise that the set of Champernowne sequences do not require strings to be in length-lexicographic order for the construction. There are $2^n!$ possible choices for zone C_n in a Champernowne sequence. For instance, 00011011 and 11100001 are two possibilities for C_2.

It is widely known that Champernowne sequences are incompressible by the Lempel-Ziv 78 compression algorithm. In [97], Lathrop and Strauss show that every sequence which is incompressible by the Lempel-Ziv 78 algorithm must be normal. Hence, this is one approach to prove that Champernowne sequences are normal.

2.6 de Bruijn Strings

The building blocks of many of the sequences constructed in this thesis are *de Bruijn* strings. A de Bruijn string of order n is a string that when viewed cyclically contains every string of length n as a substring once and once only.

Definition 2.6.1. A (binary) *de Bruijn* string of order n is a string $x \in \{0,1\}^{2^n}$ such that for all $w \in \{0,1\}^n$, $\text{occ}(w, x \cdot x[0..n-2]) = 1$.

For example, 00011101 and 00010111 are de Bruijn strings of order 3. Note that if $x \in \{0,1\}^{2^n}$ is a de Bruijn string of order n, for all $1 \leq j < 2^n$, $x[j..2^n - 1] \cdot x[0..j-1]$ is also a de Bruijn string of order n.

Such strings are named after Nicolaas de Bruijn for his work in 1946 [51], although the question about whether such strings exist was raised [53] and solved previously [52, 125]. The problem also was independently 're-discovered' and solved by Good in 1946 [75]. A history of the problem can be found in [123] and [68].

It is known that there are $2^{2^{n-1}-n}$ binary de Bruijn strings of order n unique up to cycling [51, 125, 143]. By this we mean that for example, the binary order 2 de Bruijn strings 0011 and 0110 are viewed as the same string.

2.6.1 Granddaddy de Bruijn Strings

We will often make use of the lexicographic least de Bruijn string to build sequences. For instance, of the 16 de Bruijn strings of order 4, 0000100110101111 is the lexicographic least. Such strings have been referred to as the *granddaddy* de Bruijn strings due to Knuth [90], or as *Ford* strings due to Ford's work in 1957 [67]. The first known algorithm to construct such de Bruijn strings was given by Martin in 1934 [107]. Martin's algorithm is as follows:

Martin's Algorithm:

1. Begin with the string $u = 1^{n-1}$.
2. While possible, append a bit, with 0 taking priority over 1, onto the end of u so that substrings of length n occur only once in u.
3. When step 2 is no longer possible[2], remove the prefix 1^{n-1} from u. The resulting string is the lexicographic least de Bruijn string of order n.

Martin's is a form of greedy algorithm and requires $\Omega(2^n)$ space, making it infeasible for large n. Another algorithm known as the FKM-algorithm after work by Fredricksen, Kessler and Maiorana [69, 70] requires only $O(n)$ space. We require the following two definitions to describe the FKM-algorithm.

Definition 2.6.2. A string $x \in \{0,1\}^n$ is a *necklace* if it is the lexicographical least string in the set of its rotations, i.e. of the set $\{x[j..n-1] \cdot x[0..j-1] : 0 \leq j < n\}$.

For example, the set of necklaces of $\{0,1\}^4$ is

$$\{0000, 0001, 0011, 0101, 0111, 1111\}.$$

Definition 2.6.3. The *aperiodic prefix* of a string $x \in \{0,1\}^n$ is the shortest prefix u of x such that there exists some $1 \leq j \leq n$ where $u^j = x$.

For example, the set of aperiodic prefixes of the necklaces of $\{0,1\}^4$ is

$$\{0, 0001, 0011, 01, 0111, 1\}. \tag{2.1}$$

The following is the FKM algorithm to form the lexicographic least de Bruijn string of order n:

[2]Martin shows this occurs when when $|u| = 2^n + n - 1$

FKM Algorithm:

Concatenate the aperiodic prefixes of the necklaces of $\{0,1\}^n$ in lexicographic order.

For example, concatenating the elements of the set 2.1 in lexicographic order gives us the de Bruijn string 000010011010111, i.e. the lexicographic least de Bruijn string of order 4.

Chapter 3

Finite-State Depth

Contents of this chapter were presented at SOFSEM 2020 in Cyprus during January 2020. doi: 10.1007/978-3-030-38919-2_16.

3.1 Introduction

One of the first notions of depth studied which attempted to overcome the uncomputablitiy of Kolmogorov complexity was a finite-state based notion by Doty and Moser [57]. Their notion was based on the minimal length of an input to a finite-state transducer of a certain size that results in the desired output. They proved that their notion satisfies the three fundamental properties of depth: They showed that FST-trivial sequences (those with a strong finite-state dimension of 0) and FST-incompressible sequences (those with a finite-state dimension of 1) are not deep in their notion (see Definition 3.2.6 for the full definition of dimension). They also proved a slow growth law where the transformations examined were those that could be computed by an information lossless finite-state transducer.

In their notion however, a sequence was finite-state deep if it satisfied some depth requirement for infinitely many of its prefixes. Bennett himself noted that for his notion, simply using an infinitely often requirement would result in every

computable sequence being deep [17]. In this chapter we attempt to overcome this by developing a notion of *almost everywhere finite-state depth*. Here, a sequence will be finite-state deep in our notion if it satisfies some depth requirement for all but finitely many of its prefixes.

We will show that there exists deep sequences in our notion. We will similarly show that FST-incompressible sequences are not deep and that a slow growth law holds in our notion. The question of whether FST-trivial sequences are classified as deep or not is explored also and it is discussed how this impacts the definition we choose for our depth.

We furthermore show that there exists a sequence which is deep in Moser and Doty's notion which is not deep in the new notion we present, thus differentiating the two.

3.2 Finite-State Transducers

We use the standard finite-state transducer model.

Definition 3.2.1. A *finite-state transducer (FST)* is a 4-tuple $T = (Q, q_0, \delta, \nu)$, where

- Q is a non-empty, finite set of *states*,
- $q_0 \in Q$ is the *initial state*,
- $\delta : Q \times \{0,1\} \to Q$ is the *transition function*,
- $\nu : Q \times \{0,1\} \to \{0,1\}^*$ is the *output function*.

For all $x \in \{0,1\}^*$ and $b \in \{0,1\}$, the *extended transition function* $\widehat{\delta} : \{0,1\}^* \to Q$ is defined by the recursion $\widehat{\delta}(\lambda) = q_0$ and $\widehat{\delta}(xb) = \delta(\widehat{\delta}(x), b)$. For $x \in \{0,1\}^*$, the output of T on x is the string $T(x)$ defined by the recursion

$T(\lambda) = \lambda$, and $T(xb) = T(x)\nu(\widehat{\delta}(x), b)$. We require the class of *information lossless* finite-state transducers to later demonstrate a slow growth law.

Definition 3.2.2. An FST T is *information lossless (IL)* if for all $x \in \{0,1\}^*$, the function $x \mapsto (T(x), \widehat{\delta}(x))$ is injective.

In other words, an FST T is IL if the output and final state of T on input x uniquely identify x. We call an FST that is IL an ILFST. By the identity FST, we mean the ILFST I_{FS} such that on every input x, $I_{\mathrm{FS}}(x) = x$. We write (IL)FST to denote the set of all (IL)FSTs. We note that occasionally we call ILFSTs *finite-state compressors* to emphasise when we view the ILFSTs as compressors as opposed to decompressors.

We require the concept of *(information lossless) finite-state computable* functions to demonstrate our slow growth also.

Definition 3.2.3. A function $f : \{0,1\}^\omega \to \{0,1\}^\omega$ is said to be *(information lossless) finite-state computable ((IL)FS computable)* if there is an (IL)FST T such that for all $S \in \{0,1\}^\omega$, $\lim_{n\to\infty} |T(S \upharpoonright n)| = \infty$ and for all $n \in \mathbb{N}$, $T(S \upharpoonright n) \sqsubseteq f(S)$.

Based on the above definition, if f is (IL)FS computable via the (IL)FST T, we say that $T(S) = f(S)$. We often use the following two results [82, 92] which demonstrate that any function computed by an ILFST can be inverted to be approximately computed by another ILFST.

Theorem 3.2.4 ([82, 92])**.** *For all $T \in$ ILFST, there exists $T^{-1} \in$ ILFST and a constant $c \in \mathbb{N}$ such that for all $x \in \{0,1\}^*$, $x \upharpoonright (|x| - c) \sqsubseteq T^{-1}(T(x)) \sqsubseteq x$.*

Corollary 3.2.5. *For all $T \in$ ILFST, there exists $T^{-1} \in$ ILFST such that for all $S \in \{0,1\}^\omega$, $T^{-1}(T(S)) = S$.*

3.2.1 k-Finite-State Complexity

For a class of transducers F, recall that in Section 2.4 of Chapter 2 we discussed the k-F complexity of strings and binary representations of F transducers. In this chapter we examine these notions with respect to the class of finite-state transducers, i.e. when F = FST. As such, the size of FSTs, the sets $\text{FST}^{\leq k}$, and the k-finite-state complexity $D^k_{\text{FS}}(x)$ of a string x are all defined as in Chapter 2.

In their original paper [57], Doty and Moser discuss the size of FSTs via 'some standard' *binary representation*. However, no specific discussion of a description is given. For the purposes of this chapter, we fix a binary representation of finite-state transducers σ as follows:

Let $T = (Q, q_0, \delta, \nu)$ be an FST. We define the function $\Delta : Q \times \{0,1\} \to Q \times \{0,1\}^*$, where

$$\Delta(q, b) = (\delta(q, b), \nu(q, b)) \tag{3.1}$$

which completely describes the state transitions and outputs of T. Calude, Salomaa and Roblot previously presented different methods to encode Δ in [31]. Further study of binary representations of FSTs can be found in [34]. We represent the first encoding scheme from [31] here (borrowing the notation they use) as it is used to define our own binary representation later.

For $n \in \mathbb{N}$, let $\text{bin}(n)$ denote the binary representation of n. For instance $\text{bin}(1) = 1$, $\text{bin}(2) = 10$, $\text{bin}(3) = 11$. Note that $\text{bin}(n)$ begins with a 1 for all n. $\text{string}(n)$ denotes the binary string built by removing the first 1 in $\text{bin}(n)$. So, $\text{bin}(n) = 1 \cdot \text{string}(n)$. Note that $|\text{string}(n)| = \lfloor \log(n) \rfloor$.

For $x = x_1 x_2 \dots x_l$, where $x_i \in \{0,1\}$ for $1 \leq i \leq l$, we define the following two strings:

1. $x^\dagger = x_1 0 x_2 0 \dots x_{l-1} 0 x_l 1$, and

2. $x^\diamond = \overline{(1x)^\dagger}$,

where $\bar{0} = 1$ and $\bar{1} = 0$.

Then if $Q = \{q_1, \ldots, q_m\}$, Δ is encoded by the string

$$\pi = \text{bin}(n_1)^{\ddagger} \cdot \text{string}(n_1')^{\diamond} \cdot \text{bin}(n_2)^{\ddagger} \cdots \text{bin}(n_{2m})^{\ddagger} \cdot \text{string}(n_{2m}')^{\diamond}, \tag{3.2}$$

where $\Delta(q_i, b) = (q_{(n_{2i-1+b} \bmod m)+1}, \text{string}(n'_{2i-1+b}))$, $1 \leq i \leq m$, and $b \in \{0, 1\}$. Here, $\text{bin}(n_t)^{\ddagger} = \lambda$ if the corresponding transition stays in the same state, that is $\delta(q_t, b) = q_t$. Otherwise $\text{bin}(n_t)^{\ddagger} = \text{bin}(n_t)^{\dagger}$.

While π in (3.2) gives a complete description of Δ, there is no indication of what the initial state of T is. One is left to assume that q_1 is the initial state. This leads to the question of whether changing the initial state of T to q_i, where $q_i \neq q_1$ drastically alters the length of the corresponding encoding of the new FST.

To overcome this, the binary representation $\sigma : D \to \text{FST}$ for FSTs we use is as follows. Let

$$\Delta_m = \{\pi \,|\, \pi \text{ is an encoding of } \Delta \text{ for an FST with } m \text{ states}\}$$

be the set of all possible encodings of Δ for all FSTs with m states. The domain D of σ is the set of strings

$$D = \bigcup_{m} \bigcup_{1 \leq i \leq m} \{d(\text{bin}(i))01y \,|\, y \in \Delta_m\}.$$

Then for $1 \leq i \leq m$ and $y \in \Delta_m$ we set

$$\sigma(d(\text{bin}(i))01y) = T \tag{3.3}$$

where T is the FST with $Q = \{q_1, \ldots q_m\}$ with initial state $q_0 = q_i$ and whose transition function Δ is described by y. Clearly σ is surjective and so is a binary

representation of all FSTs.

In later results we require a pointer to the initial state as for two transducers which are equivalent up to a relabelling of their states, this change of relabelling of states changes the encoding of their respective Δ. This pointer allows us to easily get a bound on the size of transducers with equivalent transition tables, but different initial states.

Consider the transducer T in Figure 3.1 with three states and with the assumption that on every transition the empty string is outputted. Simply swapping which state is labelled $\bigtriangledown$, $\square$ and $\bigcirc$ does affect the length of the encoding of Δ. For instance, if the state labelled with a $\bigtriangledown$ is the initial state, Δ is encoded in 26 bits. Otherwise, Δ is encoded in 30 bits. This is shown in Table 3.1.

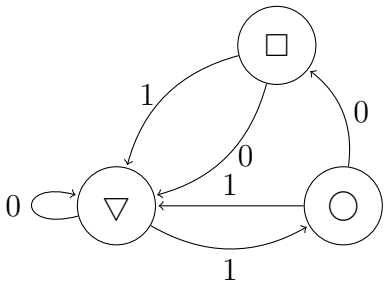

Figure 3.1: Diagram of T.

1	2	3	Δ	$\lvert\Delta\rvert$
$\bigtriangledown$	$\bigcirc$	$\square$	00100100101100110011001100	26
$\bigtriangledown$	$\square$	$\bigcirc$	00101100110011001001001100	26
$\bigcirc$	$\bigtriangledown$	$\square$	10110010010000110010010010100	30
$\bigcirc$	$\square$	$\bigtriangledown$	10010010110010110010110000110 0	30
$\square$	$\bigcirc$	$\bigtriangledown$	10010010010011001011000010010 0	30
$\square$	$\bigtriangledown$	$\bigcirc$	10010010010000101100110010010 0	30

Table 3.1: The encoding of T depending on which state is labelled $1, 2$, and 3.

Specifically our binary representation σ is used to prove Lemma 3.3.9, which in turn is needed to prove the existence of a deep sequence in Theorem 3.3.13. However, Theorem 3.3.4 demonstrates that if a sequence is deep when the size of transducers is viewed from the perspective of one binary representation, it is

deep when viewed from the perspective of any binary representation. Henceforth, we will drop the σ notation and instead write $|T|$ for $|T|_\sigma$, $\mathrm{FST}^{\leq k}$ for $\mathrm{FST}_\sigma^{\leq k}$ and $D_{\mathrm{FS}}^k(x)$ instead of $D_\sigma^k(x)$. All other definitions and results hold and can be proved regardless of the binary representation being used.

To measure the randomness of a sequence in the finite-state setting, we use the following notions which will be used to examine whether FST-trivial and FST-incompressible sequences are deep.

Definition 3.2.6. Let $S \in \{0,1\}^\omega$.

1. The *finite-state dimension* of S [49] is defined to be

$$\dim_{\mathrm{FS}}(S) = \lim_{k\to\infty} \liminf_{n\to\infty} \frac{D_{\mathrm{FS}}^k(S \upharpoonright n)}{n} = \inf_{T\in\mathrm{ILFST}} \liminf_{n\to\infty} \frac{|T(S \upharpoonright n)|}{n}. \quad (3.4)$$

2. The *finite-state strong dimension* of S [7] is defined to be

$$\mathrm{Dim}_{\mathrm{FS}}(S) = \lim_{k\to\infty} \limsup_{n\to\infty} \frac{D_{\mathrm{FS}}^k(S \upharpoonright n)}{n} = \inf_{T\in\mathrm{ILFST}} \limsup_{n\to\infty} \frac{|T(S \upharpoonright n)|}{n}. \quad (3.5)$$

Note that $\dim_{\mathrm{FS}}(S) = \rho_{\mathrm{ILFST}}(S)$ and $\mathrm{Dim}_{\mathrm{FS}}(S) = R_{\mathrm{ILFST}}(S)$ where $\rho_{\mathrm{ILFST}}(S)$ and $R_{\mathrm{ILFST}}(S)$ are the values from Definition 2.4.5 when the class of compressors being studied are ILFSTs.

Finite-state dimension (and strong finite-state dimension) satisfies many nice properties. For instance, consider how sequences can be viewed as the infinite binary expansion of real numbers (ignoring the integer part) in the interval $[0,1]$. When viewed as such, Doty, Lutz and Nandakumar have shown that for any real number $\alpha \in [0,1]$, its finite-state dimension (and strong finite-state dimension) remains unchanged under the operations of addition and multiplication with a nonzero rational number $\beta \in (0,1] \cap \mathbb{Q}$, as stated in the following theorem.

Theorem 3.2.7 ([55]). *Let $\alpha \in [0,1]$ and $\beta \in (0,1] \cup \mathbb{Q}$. The following two results hold:*

1. $\dim_{\mathrm{FS}}(\alpha) = \dim_{\mathrm{FS}}(\beta + \alpha) = \dim_{\mathrm{FS}}(\alpha \cdot \beta)$

2. $\mathrm{Dim}_{\mathrm{FS}}(\alpha) = \mathrm{Dim}_{\mathrm{FS}}(\beta + \alpha) = \mathrm{Dim}_{\mathrm{FS}}(\alpha \cdot \beta)$

The original definitions of finite-state and finite-state strong dimension presented in [7, 49] were based on finite-state gamblers and information lossless finite-state compressors. However, using the relationship between finite-state compressors and decompressors as shown in [56, 132], these definitions and the definition we use of dimension are equivalent. Intuitively, a sequence is FST-trivial (having finite-state-strong dimension of 0) if very few bits are needed to output long prefixes of the sequence by some FST. Similarly a sequence is FST-incompressible (having finite-state dimension of 1) if the number of bits required to output almost every prefix of the sequence is roughly equal to the length of the prefix being examined by every FST. Other equivalent definitions of finite-state and finite-state strong dimension can be found in terms of aligned and non-aligned block entropy rates [26, 95, 131, 157], a restricted class of super-additive functions [95] and finite-state log-loss predictors [76]. Note that the choice of binary representation of FSTs used has no effect on finite-state and finite-state strong dimension.

3.2.2 Normal Sequences and Finite-State Transducers

Normal sequences were briefly discussed in Section 2.5 of Chapter 2. Results by Schnorr and Stimm [128] and Dai, Lathrop, Lutz and Mayordomo [49] demonstrate that information lossless finite-state compressors (ILFSTs) cannot compress a sequence if and only if the sequence is normal (see [14] for a direct proof). This gives us the following result indicating that normal sequences are incompressible for FSTs.

Theorem 3.2.8. *Let $S \in \{0,1\}^\omega$. It holds that S is a normal sequence if and only if* $\dim_{\mathrm{FS}}(S) = 1$.

Note that as a corollary to Theorem 3.2.7, and as originally shown by Wall [146], when $\alpha \in [0,1]$ is a normal number (i.e. its corresponding sequence is normal), for every $\beta \in (0,1] \cap \mathbb{Q}$, it holds that $\beta + \alpha$ and $\beta \cdot \alpha$ are also normal numbers.

As an aside, it is possible to define a finite-state based descriptional complexity where normal sequences have minimal complexity. In [31], Calude, Salomaa and Roblot introduce the following complexity notion.

Definition 3.2.9. Let σ be a binary representation of FSTs and let $x \in \{0,1\}^*$. The *Calude-Salomaa-Roblot (CSR) complexity of* x with respect to σ, denoted by $CSR_\sigma(x)$, is given by

$$CSR_\sigma(x) = \min\{|y| + |T|_\sigma : T \in \mathrm{FST} \wedge T(y) = x\}. \tag{3.6}$$

In the above definition, σ may be replaced with any other binary representation of FSTs π, but the value of $CSR_\pi(x)$ and $CSR_\sigma(x)$ may differ greatly. The relationship between CSR_σ and $D^k_{\mathrm{FS}}(x)$ is demonstrated by the following: Consider two strings x and y such that $T \in \mathrm{FST}^{\leq k}$ with $T(y) = x$ and $D^k_{\mathrm{FS}}(x) = |y|$. If the same binary representation (in this case σ) is being used for both complexity notions it follows that $CSR_\sigma(x) \leq |y| + k$. $CSR_\sigma(x)$ differs from $D^k_{\mathrm{FS}}(x)$ in that the value of $CSR_\sigma(x)$ is free to look for a minimal description for x over all transducers instead of those whose size is bounded above by k. This freedom was used by Calude, Staiger and Stephan to show that regardless of the binary representation used, there exists normal sequences with minimal CSR complexity.

Theorem 3.2.10 ([34])**.** *Let σ be a binary representation of FSTs. Then there*

exists a normal sequence S such that

$$\lim_{n\to\infty} \frac{CSR_\sigma(S \upharpoonright n)}{n} = 0.$$

As a further aside, Kozachinskiy and Shen develop their own version of descriptional finite-state complexity in [95] which they call *Automatic Kolmogorov complexity*[1]. The difference with their approach is that they do not assign initial states to finite-state transducers, and they extend the domain of the transition and output functions to include λ as a valid input along with $\{0, 1\}$. This means that, if it is an option to take, a transducer can move between states and output a character without needing to read a character of its input.

To avoid the issue of every string therefore having a minimal descriptional length of 0 as a result of this, only transducers where each input only describes at most $O(1)$ number of strings are considered. Hence, the single state transducer which allows outputting either 0 or 1 without reading any characters of its input is not considered as then λ would be a description for every string.

3.3 Finite-State Depth

A sequence S is *finite-state deep* if, given any finite-state transducer, we can always build a more powerful finite-state transducer (i.e. via a combination of having more states to process the input and the ability to output longer strings) such that when we examine the k-finite-state complexity of prefixes of S on each transducer, their difference is always bounded below by the length of the prefix times a fixed constant. Intuitively, the larger transducer is more powerful and can spot patterns of the sequence that the smaller transducer cannot. As such,

[1]This is not to be confused with Shallit and Wang's *automatic complexity* discussed in Chapter 8.

the larger transducer requires less bits to describe the prefix. In [57], a notion[2] of depth based on FSTs was introduced called *infinitely often finite-state depth (i.o. FS-depth).*

Definition 3.3.1. $S \in \{0,1\}^\omega$ is *infinitely often finite-state deep (i.o. FS-deep)* if

$$(\exists \alpha > 0)(\forall k \in \mathbb{N})(\exists k' \in \mathbb{N})(\exists^\infty n \in \mathbb{N})\, D^k_{\mathrm{FS}}(S \upharpoonright n) - D^{k'}_{\mathrm{FS}}(S \upharpoonright n) \geq \alpha n.$$

We introduce an a.e. version of the original finite-state notion called *almost everywhere finite-state depth (a.e. FS-depth).*

Definition 3.3.2. $S \in \{0,1\}^\omega$ is *almost everywhere finite-state deep (a.e. FS-deep)* if both of the following hold:

1. $(\forall k \in \mathbb{N})(\exists \alpha > 0)(\exists k' \in \mathbb{N})(\forall^\infty n \in \mathbb{N})\, D^k_{\mathrm{FS}}(S \upharpoonright n) - D^{k'}_{\mathrm{FS}}(S \upharpoonright n) \geq \alpha n,$
2. $\mathrm{Dim}_{\mathrm{FS}}(S) \neq 0.$

We make note of the different order of the quantifiers in Definitions 3.3.1 and 3.3.2. We justify this order of quantifiers as in their original paper, Doty and Moser also introduced a second notion of i.o. FS-depth with the same order of quantifiers as in Definition 3.3.2. A sequence that is not a.e. FS-deep is called *almost everywhere finite-state shallow (a.e. FS-shallow).*

One of the limitations of our definition of a.e. FS-depth is condition 2 in Definition 3.3.2. If we dropped the requirement, some FST-trivial sequences would be considered deep which goes against the spirit of Bennett's original notion.

Lemma 3.3.3. *If condition* 2 *from Definition 3.3.2 was dropped, the sequence of all* 0*s would be considered a.e. FS-deep.*

[2] Actually two notions were introduced, which differ only by the order of quantifiers.

Proof. Let $k \in \mathbb{N}$. Let $p \in \mathbb{N}$ be greater than the maximum number of 0's any $T \in \mathrm{FST}^{\leq k}$ can output upon reading a single bit. Therefore we have that

$$D^k_{\mathrm{FS}}(0^n) \geq \lfloor \frac{n}{p} \rfloor \geq \frac{n}{p} - 1. \tag{3.7}$$

Next consider the FST T' such that on any input of the form $0^m 1 0^r$, T' uses 0^m to output $2pm$ 0's and then upon reading the 1, uses 0^r to output r 0's.

As every n can be written in the form $n = 2pm + r$ where $m, r \in \mathbb{N}$ and $0 \leq r < 2p$, we have that

$$D^{|T|}_{\mathrm{FS}}(0^n) \leq \lfloor \frac{n}{2p} \rfloor + r + 1 \leq \frac{n}{2p} + 2p + 1. \tag{3.8}$$

Let $0 < \alpha < 1$. Then for all but finitely many n it holds that

$$\begin{aligned} D^k_{\mathrm{FS}}(0^n) - D^{|T|}_{\mathrm{FS}}(0^n) &\geq (\frac{n}{p} - 1) - (\frac{n}{2p} + 2p + 1) && \text{(by (3.7) and (3.8))} \\ &= \frac{n}{2p} - 2p - 2 = \frac{1}{2p}(n - 4p^2) - 2 \\ &\geq n(\frac{1-\alpha}{2p}). && \text{(when } n \text{ is large)} \end{aligned}$$

As k is arbitrary, the sequence of all 0's would be considered a.e. FS-deep.

□

3.3.1 Basic Properties

The following lemma demonstrates that if a sequence S is a.e. FS-deep when the size of finite-state transducers are viewed with respect to one binary representation, then it is a.e. FS-deep regardless of what binary representation is used.

Lemma 3.3.4. *Let π be a binary representation of FSTs. Let S be an a.e. FS-deep sequence when the size of the FSTs are viewed with respect to the binary*

representation π. Then S is a.e. FS-deep when the size of the FSTs are viewed with respect to every binary representation.

Proof. Let S and π be as in the statement of the lemma. Let τ be a different binary representation of all FSTs.

Fix $k \in \mathbb{N}$. There exists a constant c such that $\mathrm{FST}_\tau^{\leq k} \subseteq \mathrm{FST}_\pi^{\leq k+c}$. Therefore for all $n \in \mathbb{N}$,

$$D_\pi^{k+c}(S \upharpoonright n) \leq D_\tau^k(S \upharpoonright n). \tag{3.9}$$

As S is a.e. FS-deep with respect to π, there exists constants α_k and $(k+c)'$ such that for almost every n

$$D_\pi^{k+c}(S \upharpoonright n) - D_\pi^{(k+c)'}(S \upharpoonright n) \geq \alpha_k n. \tag{3.10}$$

Let d be a constant such that $\mathrm{FST}_\pi^{(k+c)'} \subseteq \mathrm{FST}_\tau^{(k+c)'+d}$. Therefore for almost every n,

$$D_\tau^{(k+c)'+d}(S \upharpoonright n) \leq D_\pi^{(k+c)'}(S \upharpoonright n). \tag{3.11}$$

Therefore by Equation (3.10), for almost every n,

$$D_\tau^k(S \upharpoonright n) - D_\tau^{k'}(S \upharpoonright n) \geq D_\pi^{k+c}(S \upharpoonright n) - D_\pi^{(k+c)'}(S \upharpoonright n) \geq \alpha_k n \tag{3.12}$$

where $k' = (k+c)' + d$. As $\mathrm{Dim}_{\mathrm{FS}}(S) \neq 0$ regardless of the binary representation and as k is arbitrary, we have that S is also a.e FS-deep with respect to τ.

□

The following result shows that sequences that incompressible by finite-state transducers, i.e. normal sequences, cannot be a.e. FS-deep.

Theorem 3.3.5. *Let $S \in \{0,1\}^\omega$. If $\dim_{\mathrm{FS}}(S) = 1$, then S is not a.e. FS-deep.*

Proof. Let $S \in \{0,1\}^\omega$ be such that $\dim_{\mathrm{FS}}(S) = 1$. Let $k' \in \mathbb{N}$ and $\alpha > 0$. Then

$$(\forall^\infty n \in \mathbb{N})\, D^{k'}_{\mathrm{FS}}(S \upharpoonright n) > (1-\alpha)n. \tag{3.13}$$

Therefore, if k is such that $I_{\mathrm{FS}} \in \mathrm{FST}^{\leq k}$, it holds that

$$(\forall^\infty n \in \mathbb{N})\, D^{k}_{\mathrm{FS}}(S \upharpoonright n) - D^{k'}_{\mathrm{FS}}(S \upharpoonright n) < n - (1-\alpha) = \alpha n. \tag{3.14}$$

As α and k' are arbitrary, S is not a.e. FS-deep.

□

3.3.2 Slow Growth Law

In the following subsection we prove a *Slow Growth Law* for a.e. FS-depth. Recall that the intuition behind slow growth laws is that deep sequences cannot be constructed from simple processes, i.e. they must be computationally difficult to make. We demonstrate that a.e. FS-depth satisfies a slow growth law by demonstrating that no a.e. FS-deep sequence can be constructed from an a.e. FS-shallow sequence via an ILFS computable process. This is the same transformation used by Doty and Moser in their i.o. FS-depth notion [57].

To demonstrate the slow growth law we require the following lemma regarding how ILFSTs do not greatly alter the k-finite-state complexity of strings. It previously appeared in [57] in a slightly different format but we restate and reprove parts of it here.

Lemma 3.3.6 ([57])**.** *Let M be an* ILFST*.*

1. $(\forall k \in \mathbb{N})(\exists k' \in \mathbb{N})(\forall x \in \{0,1\}^*)\, D^{k'}_{\mathrm{FS}}(M(x)) \leq D^{k}_{\mathrm{FS}}(x)$.
2. $(\forall \varepsilon > 0)(\forall k \in \mathbb{N})(\exists k' \in \mathbb{N})(\forall^\infty x \in \{0,1\}^*)\, D^{k'}_{\mathrm{FS}}(x) \leq (1+\varepsilon) D^{k}_{\mathrm{FS}}(M(x)) + O(1)$.

Proof. The proof for part 1 is in [57].

For part 2, let ε, k, x and M be as stated in the lemma. Furthermore let $0 < \varepsilon' < \varepsilon$. By Theorem 3.2.4, there exists an ILFST M^{-1} and a constant $c \in \mathbb{N}$ such that for all $y \in \{0,1\}^*$, $y \upharpoonright (|y| - c) \sqsubseteq M^{-1}(M(y)) \sqsubseteq y$.

Let p be a k-minimal program for $M(x)$, i.e. $A(p) = M(x)$ for $A \in \mathrm{FST}^{\leq k}$ and $D^k_{\mathrm{FS}}(M(x)) = |p|$.

Let $b = \lceil \frac{2}{\varepsilon'} \rceil$. There exists non-negative integers n and r such that $|p| = nb + r$, where $0 \leq r < b$. Let p' be a new string such that p' begins with the first nb bits of p, with a 0 placed to separate every b bits starting at the beginning of the string, followed by a 1 and then the remaining r bits of p doubled. That is

$$p' = 0p_1 \dots p_b 0 p_{b+1} \dots p_{2b} 0 \dots p_{nb} 1 p_{nb+1} p_{nb+1} \dots p_{nb+r} p_{nb+r}.$$

Note therefore that

$$|p'| = n(b+1) + 2r + 1 = |p| + n + r + 1 < |p| + n + b + 1. \tag{3.15}$$

$nb \leq |p|$ means $n \leq \lceil \frac{|p|}{b} \rceil$ and so for $|p|$ large it holds that

$$\begin{aligned} |p'| &\leq |p| + \left\lceil \frac{|p|}{b} \right\rceil + b + 1 \leq |p| + 2\left\lceil \frac{|p|}{b} \right\rceil = |p| + 2\left\lceil \frac{|p|}{\lceil \frac{2}{\varepsilon'} \rceil} \right\rceil \\ &\leq |p| + 2(\frac{\varepsilon' |p|}{2} + 1) = |p|(1 + \varepsilon') + 2 \\ &\leq |p|(1 + \varepsilon). \end{aligned} \tag{3.16}$$

Next we build A' for x. Let $y = M^{-1}(M(x))$, i.e. $x = yz$ for some $|z| \leq c$. Let A' be the machine such that on input $p'01z$: A' uses p' to simulate $A(p)$ to retrieve $M(x)$. A' knows where p' ends due to the 01 separator. A' takes $M(x)$'s output and simulates it on M^{-1} to retrieve y. This is possible as the composition of ILFSTs can be simulated by an ILFST. After seeing the separator, A' acts as

the identity transducer and outputs z. Thus $A'(p'01z) = yz = x$. Thus

$$D_{\mathrm{FS}}^{|A'|}(x) \leq |p'| + 2 + |z| \leq |p|(1+\varepsilon) + 2 + c = D_{\mathrm{FS}}^{k}(M(x)) + O(1). \tag{3.17}$$

As $|A'|$ depends on the size of A, the size of M^{-1} and b (i.e. it does not vary with x), k' can be chosen such that $k' = |A'|$.

□

Consider two sequences S, $S' \in \{0,1\}^\omega$ such that S' is ILFS computable from S via the ILFST M. It may be the case that M on input of prefixes of S does not output every prefix of S'. For instance, it may only output the prefixes of even length. We use the following remark to relate the finite-state complexity of prefixes of S' with just those prefixes that are outputted by M on input S.

Remark 3.3.7. Let S, $S' \in \{0,1\}^\omega$ be such that S' is ILFS computable from S via the ILFST M. For each n, let m_n denote the largest integer such that $M(S \upharpoonright m_n) \sqsubseteq S' \upharpoonright n$. Then for all $0 < \varepsilon < 1$ we have that

$$(\forall k \in \mathbb{N})(\exists k' \in \mathbb{N})(\forall^\infty n \in \mathbb{N}) D_{\mathrm{FS}}^{k'}(S' \upharpoonright n) \leq (1+\varepsilon) D_{\mathrm{FS}}^{k}(M(S \upharpoonright m_n)) + O(1).$$

Proof. Let S, S', M, and the sequence of integers $\{m_n\}_{n\in\mathbb{N}}$ be as stated in the lemma. For each n, let $y_n \in \{0,1\}^*$ be the string which satisfies $M(S \upharpoonright m_n) \cdot y_n = S' \upharpoonright n$.

The proof then follows the same structure as part two of Lemma 3.3.6: By letting p be a k-description of $M(S \upharpoonright m_n)$, we can similarly use a padded version of p, along with a separator, and then the string y_n to retrieve $S' \upharpoonright n$. Since $|y_n|$ is bounded, for each chosen ε, the inequality will hold when $|p|$ is long enough, i.e. for long enough prefixes of $S' \upharpoonright n$.

□

Theorem 3.3.8 (Slow Growth Law)**.** *Let $S \in \{0,1\}^\omega$. Let $f : \{0,1\}^\omega \longrightarrow \{0,1\}^\omega$ be* ILFS *computable, and let $S' = f(S)$. If S' is a.e. FS-deep, then S is a.e. FS-deep.*

Proof. Let S, S', and f be as stated in the theorem and let M be an ILFST computing f.

For all n, let m_n denote the largest integer such that $M(S \upharpoonright m_n) \sqsubseteq S' \upharpoonright n$, and let $y_n \in \{0,1\}^*$ be such that $M(S \upharpoonright m_n) \cdot y_n = S' \upharpoonright n$. Note that $y_n = \lambda$ infinitely often. As M is IL, it cannot visit a state twice without outputting at least one bit, so there exists a $\beta > 0$ such that for all such n, $n \geq \beta m_n$.

Fix $l \in \mathbb{N}$. Let k be from Lemma 3.3.6 such that for all $x \in \{0,1\}^*$,

$$D^k_{\mathrm{FS}}(M(x)) \leq D^l_{\mathrm{FS}}(x). \tag{3.18}$$

As S' is a.e. FS-deep, there exists k' and $\alpha > 0$ such that for almost every n

$$D^k_{\mathrm{FS}}(S' \upharpoonright n) - D^{k'}_{\mathrm{FS}}(S' \upharpoonright n) \geq \alpha n. \tag{3.19}$$

Note that we can choose k' to satisfy (3.18) and be large enough so that $I_{\mathrm{FS}} \in \mathrm{FST}^{\leq k'}$. Hence, for all x and constants c, $D^{k'}_{\mathrm{FS}}(x)(1+c) \leq D^{k'}_{\mathrm{FS}}(x) + |x|c$.

Let $0 < \varepsilon < \alpha$. Let $l' \in \mathbb{N}$ be from Lemma 3.3.6 such that for almost every x,

$$D^{l'}_{\mathrm{FS}}(x) \leq D^{k'}_{\mathrm{FS}}(M(x))(1 + \frac{\varepsilon}{2}) + O(1). \tag{3.20}$$

Hence, for almost every m, there exists an n such that

$$\begin{aligned}
D^l_{\mathrm{FS}}(S \upharpoonright m) - D^{l'}_{\mathrm{FS}}(S \upharpoonright m) &\geq D^l_{\mathrm{FS}}(S \upharpoonright m) - D^{k'}_{\mathrm{FS}}(M(S \upharpoonright m))(1 + \frac{\varepsilon}{2}) - O(1) \\
&\geq D^k_{\mathrm{FS}}(M(S \upharpoonright m)) - D^{k'}_{\mathrm{FS}}(M(S \upharpoonright m))(1 + \varepsilon) \\
&= D^k_{\mathrm{FS}}(M(S \upharpoonright m_n)) - D^{k'}_{\mathrm{FS}}(M(S \upharpoonright m_n))(1 + \varepsilon) \\
&= D^k_{\mathrm{FS}}(S' \upharpoonright n) - D^{k'}_{\mathrm{FS}}(S' \upharpoonright n)(1 + \varepsilon) \\
&\geq \alpha n - \varepsilon(D^{k'}_{\mathrm{FS}}(S' \upharpoonright n)) \\
&\geq (\alpha - \varepsilon)n \\
&\geq (\alpha - \varepsilon)\beta m_n \geq (\alpha - \varepsilon)\beta m.
\end{aligned}$$

Thus as l was arbitrary, S satisfies condition 1 of Definition 3.3.2.

Next we must be show that $\mathrm{Dim}_{\mathrm{FS}}(S) \neq 0$. Recall that for every sequence T that

$$\mathrm{Dim}_{\mathrm{FS}}(T) = \lim_{k\to\infty} \limsup_{n\to\infty} \frac{D^k_{\mathrm{FS}}(T \upharpoonright n)}{n}.$$

First let $\delta > 0$. In the following, for each k, let k'' and k' be such that for almost all $n \in \mathbb{N}$

$$\begin{aligned}
D^{k''}_{\mathrm{FS}}(S' \upharpoonright n) &\leq (1 + \delta)D^{k'}_{\mathrm{FS}}(M(S \upharpoonright m_n)) + 2 + |y_n| \\
&= (1 + \delta)D^{k'}_{\mathrm{FS}}(M(S \upharpoonright m_n)) + O(1) \\
&\leq (1 + \delta)D^k_{\mathrm{FS}}(S \upharpoonright m_n) + O(1).
\end{aligned}$$

Such k'' and k' exist by Lemma 3.3.6 and Remark 3.3.7. We will use the following facts to show that $\mathrm{Dim}_{\mathrm{FS}}(S) \neq 0$: We require the following two facts to complete the proof: For any sequence

1. The limit superior of any subsequence is less than or equal to the limit

superior of the original sequence.

2. Every subsequence of a convergent sequence converges to the same limit.

As S' is a.e. FS-deep, it follows that $\mathrm{Dim}_{\mathrm{FS}}(S') > 0$. Therefore we have that

$$\begin{aligned}
0 &< \mathrm{Dim}_{\mathrm{FS}}(S') \\
&= \lim_{k\to\infty} \limsup_{n\to\infty} \frac{D^k_{\mathrm{FS}}(S' \upharpoonright n)}{n} \\
&= \lim_{k\to\infty} \limsup_{n\to\infty} \frac{D^{k''}_{\mathrm{FS}}(S' \upharpoonright n)}{n} && \text{(by Fact 2)} \\
&\leq \lim_{k\to\infty} \limsup_{n\to\infty} \frac{(1+\delta)D^{k'}_{\mathrm{FS}}(M(S \upharpoonright m_n) + O(1)}{n} \\
&\leq \lim_{k\to\infty} \limsup_{n\to\infty} \frac{(1+\delta)D^{k}_{\mathrm{FS}}(S \upharpoonright m_n)}{n} \\
&\leq \lim_{k\to\infty} \limsup_{n\to\infty} \frac{(1+\delta)D^{k}_{\mathrm{FS}}(S \upharpoonright m_n)}{m_n\beta} && \text{(as } n \geq m_n\beta\text{)} \\
&\leq \lim_{k\to\infty} \limsup_{m\to\infty} \frac{(1+\delta)D^{k}_{\mathrm{FS}}(S \upharpoonright m)}{m\beta} && \text{(by Fact 1 above)} \\
&= \frac{(1+\delta)}{\beta}\mathrm{Dim}_{\mathrm{FS}}(S).
\end{aligned}$$

Hence as $(1+\delta)/\beta$ is non-zero, neither is $\mathrm{Dim}_{\mathrm{FS}}(S)$. Thus S is a.e. FS-deep.

□

3.3.3 Existence of an a.e. FS-Deep Sequence

To prove the existence of an a.e. FS-deep sequence we need Lemmas 3.3.9 and 3.3.12 which examine the k-finite-state complexity of substrings within a string on FSTs of roughly the same size. Lemma 3.3.9's proof relies on viewing FSTS with respect to our fixed binary representation σ described in Section 3.2.1.

Suppose we are given an FST T and an input vw. If T outputs xy on reading vw and x on reading the prefix v, this means that v is a description of x and vw is a description of xy via T. We can alter T to create a new transducer T' such

that the states and transitions of T' and T are the same, with the only difference being that the start state of T' is the one T ends in after reading v. This means that w is a description of y via T'.

Lemma 3.3.9. *For our fixed binary representation σ (from 3.3), when $k \geq 4$ it holds that*

$$(\forall n \in \mathbb{N})(\forall x, y, z \in \{0,1\}^*)\, D^k_{\mathrm{FS}}(xy^n z) \geq D^{3k}_{\mathrm{FS}}(x) + nD^{3k}_{\mathrm{FS}}(y) + D^{3k}_{\mathrm{FS}}(z).$$

Proof. Let k, n, x, y and z be as stated. Let $T \in \mathrm{FST}^{\leq k}$ and $p_x, p_{y,i}, \ldots p_{y,n}, p_z \in \{0,1\}^*$ be such that $D^k_{\mathrm{FS}}(xy^n z) = |p_x p_{y,1} \ldots p_{y,n} p_z|$ with $T(p_x p_{y,1} \ldots p_{y,n} p_z) = xy^n z$, $T(p_x p_{y,1} \ldots p_{y,j}) = xy^j$ for $1 \leq j \leq n$, and $T(p_x) = x$.

For all $w \in \{0,1\}^*$, let T_w be the FST such that T_w's states, transitions, and outputs are the same as T's with the only difference being that the start state of T_w is the state that T on input w ends in. That is, T_w's initial state is the state $\delta_T(q_0, w)$ where q_0 is the initial state of T. Hence we have that $T_{p_x}(p_{y,1}) = y$, $T_{p_x p_{y,1} \ldots p_{y,n}}(p_z) = z$ and for $2 \leq j \leq n$, $T_{p_x p_{y,1} \ldots p_{y,j-1}}(p_{y,j}) = y$.

Next we put a bound on the binary description length of T_w. Recall by our choice of σ, for an FST M, $\sigma(d(\mathrm{bin}(n))01\pi) = M$ where $d(\mathrm{bin}(n))$ is a pointer to M's start state q_n, and π describes the function Δ of M's transitions and outputs. We write Δ_M for the Δ function of M.

As $|T| \leq k$, T has at most k states. Therefore the pointer to T_w's start state takes at most $2|\mathrm{bin}(k)| = 2(\lfloor \log k \rfloor + 1)$ bits to encode. Similarly, the encoding of Δ_T can be used to encode Δ_{T_w} and so the number of bits required to encode Δ_{T_w} is bounded above by k bits also. Hence we have that whenever $k \geq 4$

$$|T_w| \leq 2(\lfloor \log k \rfloor + 1) + 2 + k \leq 3k. \tag{3.21}$$

Thus for $k \geq 4$ we have that $D^{3k}_{\mathrm{FS}}(x) \leq |p_x|$, $D^{3k}_{\mathrm{FS}}(z) \leq |p_z|$ and $D^{3k}_{\mathrm{FS}}(y) \leq |p_y|$

where $|p_y| = \min\{|p_{y,j}| : 1 \leq j \leq n\}$.

Hence

$$D^k_{\mathrm{FS}}(xy^nz) \geq |p_x| + n|p_y| + |p_z| \geq D^{3k}_{\mathrm{FS}}(x) + nD^{3k}_{\mathrm{FS}}(y) + D^{3k}_{\mathrm{FS}}(z)$$

as desired.

□

Remark 3.3.10. Note that for all $x \in \{0,1\}^*$, $D^k_{\mathrm{FS}}(x) \leq D^{k+1}_{\mathrm{FS}}(x)$. Hence while Lemma 3.3.9's result is only for $k \geq 4$, it gives us that for all x, y, z,

$$D^1_{\mathrm{FS}}(xy^nz) \geq D^2_{\mathrm{FS}}(xy^nz) \geq D^3_{\mathrm{FS}}(xy^nz) \geq D^{12}_{\mathrm{FS}}(x) + nD^{12}_{\mathrm{FS}}(y) + D^{12}_{\mathrm{FS}}(z).$$

Remark 3.3.11. Lemma 3.3.9 can be generalised such that for our fixed binary representation σ, we can break the input into any number of substrings to get a similar result. That is for any string $x = x_1 \dots x_n$,

$$(\forall^\infty k \in \mathbb{N})\, D^k_{\mathrm{FS}}(x_1 \dots x_n) \geq \sum_{i=1}^{n} D^{3k}_{\mathrm{FS}}(x_i).$$

The following lemma states that for almost every pair of strings x and y, given a description of x and a description of y, a transducer T can be built such that upon reading a padded version of the description of x, a flag, and the description for y, T can output the string xy.

Lemma 3.3.12. $(\forall \varepsilon > 0)(\forall k \in \mathbb{N})(\exists k' \in \mathbb{N})(\forall^\infty x \in \{0,1\}^*)(\forall y \in \{0,1\}^*)$

$$D^{k'}_{\mathrm{FS}}(xy) \leq (1+\varepsilon)D^k_{\mathrm{FS}}(x) + D^k_{\mathrm{FS}}(y) + 2.$$

Proof. Let ε, x, y and k be as stated in the lemma. Consider $p, q \in \{0,1\}^*$ such that $D^k_{\mathrm{FS}}(x) = |p|$ and $D^k_{\mathrm{FS}}(y) = |q|$, and suppose $A, B \in \mathrm{FST}^{\leq k}$ where $A(p) = x$ and $B(q) = y$.

Let $b = \lceil \frac{2}{\varepsilon} \rceil$. Then there exists integers n an r such that $|p| = nb + r$, where $0 \leq r < b$. Let p' be a new string such that p' begins with the first nb bits of p, with a 0 placed to separate every b bits starting at the beginning of the string. This is followed by a 1 and the remaining r bits of p doubled, i.e.

$$p' = 0p_1 \dots p_b 0 p_{b+1} \dots p_{2b} 0 \dots p_{nb} 1 p_{nb+1} p_{nb+1} \dots p_{nb+r} p_{nb+r}.$$

Then by the same argument as in Lemma 3.3.6, whenever $|p|$ is large enough we can arrive to the same result as in Equation (3.16), i.e. it holds that $|p'| \leq |p|(1+\varepsilon)$. Another way of saying this holds only for large $|p|$ is when $D^k_{\mathrm{FS}}(x)$ is large.

Next let $M \in \mathrm{FST}^{\leq k'}$ (where k' is a number whose value is dependent only on k and b) be the FST such that on input $p'10q$: M uses p' to output $A(p) = x$. M can spot the beginning bits of p from the blocks of size b by the 0s. When M sees the block beginning with 1 it knows that the remaining bits will be the final bits of p doubled. Upon reading 10, M uses the remaining bits to output $B(q) = y$. Therefore, for almost all x and all y it holds that

$$D^{k'}_{\mathrm{FS}}(xy) \leq |p'| + |q| + 2 \leq (1+\varepsilon) D^k_{\mathrm{FS}}(x) + D^k_{\mathrm{FS}}(y) + 2.$$

□

We now present the main result of this chapter by constructing an a.e. FS-deep sequence. The sequence is constructed in consecutive blocks where each block is devoted to some pair k, k'. On such a block, transducers of size k do poorly, while some larger transducer of size k' does very well. The key difference from the proof in [57] where Doty and Moser prove the existence of an i.o. FS-deep sequence is that blocks devoted to the same pair (k, k') repeat every constant number of blocks. This method of repeating blocks assigned to a certain pair at

a predetermined frequency has also been used to construct deep sequences in different notions of depth [62, 77]. This ensures an a.e. FS-deep sequence, as opposed to merely an i.o. FS-deep sequence.

Theorem 3.3.13. *There exists an a.e. FS-deep sequence.*

Proof. We begin by partitioning the non-negative integers into disjoint consecutive intervals $I_0, I_1, \ldots$ where interval I_j has size 2^j. For instance, $I_0 = \{0\}$, $I_1 = \{1, 2\}$ and $I_3 = \{3, 4, 5, 6\}$. For each I_j, set $m_j = \min(I_j)$ and $M_j = \max(I_j)$. Note that for all j, $m_{j+1} = M_j + 1$. For all integers $k \geq 0$, k is devoted to every interval I_j where j is of the form $2^k - 1 + t(2^{k+1})$ for $t \geq 0$. So $k = 0$ will first be assigned to I_0 and every 2^{nd} interval after that. $k = 1$ is first assigned to I_1 and every 4^{th} interval after that, and so on. If k is devoted to I_j, I_{j_+} denotes the next interval k is devoted to.

S is constructed in stages $S_0 S_1 S_2 \ldots$, where $S[m_j..M_j] = S_j$. Note that $|S_j| = |I_j|$. Each stage S_j of S is constructed as follows:

If I_j is devoted to 0, set $S_j = 0^{|I_j|}$. Otherwise if I_j is devoted to some non-zero k, suppose $j = 2^k - 1 + t(2^{k+1})$ for some t. Let r_k be a string of length 2^{2^k-1} that is $3k$-FS random in the sense that

$$D^{3k}_{\text{FS}}(r_k) \geq |r_k| - 4k. \tag{3.22}$$

Such a string exists as $|\text{FST}^{\leq 3k}| \cdot 2^{|r_k|-4k} < 2^{|r_k|}$ for $k \geq 1$. In this case construct S_j by concatenating $|I_j|/|r_k|$ copies of r_k. That is

$$S_j = r_k^{\frac{|I_j|}{|r_k|}} = r_k^{2^{t(2^{k+1})}}. \tag{3.23}$$

Fix some $k \geq 4$ (this allow us to apply Lemma 3.3.9) and consider some arbitrarily long prefix $S \upharpoonright n$ of S. Let $j = \max\{i : I_i \text{ is assigned to } k \text{ and } M_i \leq n - 1\}$. That is, j denotes the index of the largest interval devoted to k such that

S_j is a substring of $S \upharpoonright n$ but the substring denoted by S_{j+} which is tied to interval I_{j+} is not. We split $S \upharpoonright n$ into three substrings x, y, z such that $x = S[0..M_{j-1}]$, $y = S_j$, and $z = S[m_{j+1}..n-1]$.

We find a lower bound for the k-finite-state complexity of xyz as follows:

$$\begin{aligned} D^k_{\mathrm{FS}}(xyz) &\geq D^{3k}_{\mathrm{FS}}(x) + \frac{|I_j|}{|r_k|} D^{3k}_{\mathrm{FS}}(r_k) + D^{3k}_{\mathrm{FS}}(z) && \text{(by Lemma 3.3.9)} \\ &\geq D^{3k}_{\mathrm{FS}}(x) + |I_j|(1 - \frac{4k}{|r_k|}) + D^{3k}_{\mathrm{FS}}(z) && \text{(by (3.22))} \\ &\geq D^{3k}_{\mathrm{FS}}(x) + |I_j| c_1 + D^{3k}_{\mathrm{FS}}(z), && (3.24) \end{aligned}$$

where $c_1 = (2^{11} - 1)/2^{11}$ as $1 - \frac{4k}{|r_k|}$ is minimum for $k = 4$.

For each $r \in \{0,1\}^*$, consider the single state FST T_r such that for each bit of its input read, T_r stays in the same state and outputs r. That is, for all $x \in \{0,1\}^*$, $T_r(x) = r^{|x|}$. Let $\widehat{k}$ be large enough such that $\widehat{k} > 3k$ and both the identity transducer I_{FS} and T_{r_k} are contained in $\mathrm{FST}^{\leq \widehat{k}}$. This enables us to get the following upper bounds for the $\widehat{k}$-finite-state complexity of x and y of

$$D^{\widehat{k}}_{\mathrm{FS}}(x) \leq |x| = 2^j - 1 < 2^j = |I_j| \text{ and } D^{\widehat{k}}_{\mathrm{FS}}(y) \leq \frac{|I_j|}{|r_k|}. \tag{3.25}$$

Next set $\varepsilon = 5/10923$. We set δ to have a value such that $\delta + \delta^2/4 = \varepsilon$. By Lemma 3.3.12, whenever $D^{\widehat{k}}_{\mathrm{FS}}(x)$ and $D^{\widehat{k}}_{\mathrm{FS}}(y)$ are large enough (i.e. for long enough prefixes of S) we have that there exists k'', $k' \geq 3k$ such that

$$\begin{aligned} D^{k''}_{\mathrm{FS}}(xyz) &\leq (1 + \frac{\delta}{2}) D^{k'}_{\mathrm{FS}}(xy) + D^{k'}_{\mathrm{FS}}(z) + 2 && \text{(by Lemma 3.3.12)} \\ &\leq (1 + \frac{\delta}{2})^2 D^{\widehat{k}}_{\mathrm{FS}}(x) + (1 + \frac{\delta}{2}) D^{\widehat{k}}_{\mathrm{FS}}(y) + D^{k'}_{\mathrm{FS}}(z) + 4 && \text{(by Lemma 3.3.12)} \\ &\leq (1 + \varepsilon)(D^{\widehat{k}}_{\mathrm{FS}}(x) + D^{\widehat{k}}_{\mathrm{FS}}(y)) + D^{k'}_{\mathrm{FS}}(z) + 4 && \text{(by choice of } \delta) \\ &\leq D^{\widehat{k}}_{\mathrm{FS}}(x) + |I_j|(\varepsilon + \frac{1+\varepsilon}{|r_k|}) + D^{k'}_{\mathrm{FS}}(z) + 4 && \text{(by (3.25))} \\ &= D^{\widehat{k}}_{\mathrm{FS}}(x) + |I_j| c_2 + D^{k'}_{\mathrm{FS}}(z) + 4 && (3.26) \end{aligned}$$

where $c_2 = 2^{-11}$ as $1/|r_k|$ is maximum for $k = 4$.

Hence we have for almost every m,

$$\begin{aligned}
D^k_{\mathrm{FS}}(S \upharpoonright n) - D^{k''}_{\mathrm{FS}}(S \upharpoonright n) &= D^k_{\mathrm{FS}}(xyz) - D^{k''}_{\mathrm{FS}}(xyz) \\
&\geq (D^{3k}_{\mathrm{FS}}(x) - D^{\widehat{k}}_{\mathrm{FS}}(x)) + (D^{3k}_{\mathrm{FS}}(z) - D^{k'}_{\mathrm{FS}}(z)) \\
&\quad + |I_j|(c_1 - c_2) - 4 && \text{(by (3.24) and (3.26))} \\
&= |I_j|(\frac{2^{10}-1}{2^{10}}) - 4 \\
&\geq |I_j|(\frac{2^9-1}{2^9}) && \text{(for } j \text{ large)} \\
&= \frac{(2^9-1)|I_{j+}|}{2^9(2^{2^{k+1}})} && \text{(as } |I_{j+}| = 2^{2^{k+1}}|I_j|) \\
&> \frac{n}{2}(\frac{2^9-1}{2^{2^{k+1}+9}}) && \text{(as } n < M_{j+} < 2|I_{j+}|) \\
&= n\alpha_k && (3.27)
\end{aligned}$$

where $\alpha_k = (2^9-1)/2^{2^{k+1}+10}$. Condition 1 of Definition 3.3.2 for a.e. FS-depth is then met as for $0 \leq k \leq 3$, Equation (3.27) is satisfied by taking the appropriate k'' when $k = 4$ and setting α_k to be α_4.

Finally we must show that $\mathrm{Dim}_{\mathrm{FS}}(S) \neq 0$. Let $k \geq 4$. We consider the k-finite-state complexity of prefixes of the form $S[0..M_j]$ of S where I_j is an interval devoted to k. Hence we have that

$$\begin{aligned}
D^k_{\mathrm{FS}}(S[0..M_j]) &\geq \frac{|I_j|}{|r_k|}(D^{3k}_{\mathrm{FS}}(r_k)) && \text{(by Lemma 3.3.9)} \\
&\geq |I_j|(1 - \frac{4k}{|r_k|}) && \text{(by (3.22))} \\
&\geq |I_j|(\frac{2^{11}-1}{2^{11}}) && \text{(as } 4k/|r_k| \text{ is minimum for } k = 4) \\
&\geq \frac{2^{11}-1}{2^{12}}|S[0..M_j]|. && \text{(as } M_j < 2|I_j|)
\end{aligned}$$

Hence we have that $\mathrm{Dim}_{\mathrm{FS}}(S) \geq (2^{11}-1)/2^{12} > 0$. Therefore S is a.e. FS-deep.

□

3.3.4 Separation from i.o. FS-depth

The following result demonstrates a difference between our a.e. FS-depth and Doty and Moser's original i.o. FS-depth notion. We do this by building a sequence which is i.o. FS-deep but not a.e. FS-deep, thus demonstrating that being i.o. FS-deep is a weaker requirement. The sequence is constructed similarly to the sequence from Theorem 3.3.13, however, every second block is now a random string. This prevents us achieving a.e. FS-depth.

Theorem 3.3.14. *There exists a sequence which is i.o. FS-deep but not a.e. FS-deep.*

Proof. We begin by partitioning the non-negative integers into disjoint consecutive intervals $I_1, I_2, \ldots$ where $|I_1| = 2$ and $|I_j| = 2^{|I_1|+\cdots+|I_{j-1}|}$ for $j \geq 2$. For instance, $I_1 = \{0, 1\}$, $I_2 = \{2, 3, 4, 5\}$, and $I_3 = \{6, 7, \ldots, 69\}$. For each I_j, set $m_j = \min(I_j)$ and $M_j = \max(I_j)$. Note that for all j, $m_{j+1} = M_j + 1$.

S is constructed in stages $S_1S_2S_3\ldots$ where S_j denotes the substring $S[m_j..M_j]$. For all j, we henceforth use the notation $\overline{S_j}$ to denote the prefix $S_1 \cdots S_j$ of S. Note that $|S_j| = |I_j|$ and when $j > 1$ we have that $\log(|S_j|) = |\overline{S_{j-1}}|$. Each stage S_j is constructed as follows:

For every interval I_j where j is odd, we set S_j to be a string with maximal plain Kolmogorov complexity in the sense that $K(S_j) \geq |S_j|$. If j is even, I_j is devoted to some FST description bound length k. Specifically for each k, k is devoted to every interval I_j where j is of the form $j = 2^k + t(2^{k+1})$, for $t \geq 0$. So $k = 1$ is first devoted to I_2 and every 4^{th} interval after that and $k = 2$ is first devoted to I_4 and every 8^{th} interval after that, and so on. For each k, let r_k be a

string of length $|I_{2^k}|$ such that r_k is $3k$-FS random in the sense that

$$D^{3k}_{\mathrm{FS}}(r_k) \geq |r_k| - 4k. \tag{3.28}$$

Such a string exists as $|\mathrm{FST}^{\leq 3k}| \cdot 2^{|r_k|-4k} < 2^{|r_k|}$ for $k \geq 1$. Then if I_j is devoted to k we set

$$S_j = r_k^{\frac{|I_j|}{|r_k|}} = r_k^{2^{t(2^{k+1})}}. \tag{3.29}$$

First we show S is i.o. FS-deep:

Fix some $k \geq 4$ (this allows us to apply Lemma 3.3.9). We consider prefixes of the form $\overline{S_j}$ of S where interval I_j is devoted to k. We first find a lower bound for the k-finite-state complexity of $\overline{S_j}$. We have that

$$\begin{aligned} D^{k}_{\mathrm{FS}}(\overline{S_j}) &\geq D^{3k}_{\mathrm{FS}}(\overline{S_{j-1}}) + \frac{|S_j|}{|r_k|} D^{3k}_{\mathrm{FS}}(r_k) && \text{(by Lemma 3.3.9)} \\ &\geq \frac{|S_j|}{|r_k|}(|r_k| - 4k). && \text{(by (3.28))} \end{aligned}$$

Let T_{r_k} be the single state FST as described in Theorem 3.3.13 which on every input bit outputs r_k. Let $\widehat{k}$ be large enough so that T_{r_k} and the identity transducer are contained in $\mathrm{FST}^{\leq \widehat{k}}$. This enables us to get upper bounds for the $\widehat{k}$-finite-state complexity of $\overline{S_{j-1}}$ and S_j of

$$D^{\widehat{k}}_{\mathrm{FS}}(\overline{S_{j-1}}) \leq |\overline{S_{j-1}}| \text{ and } D^{\widehat{k}}_{\mathrm{FS}}(S_j) \leq \frac{|S_j|}{|r_k|}. \tag{3.30}$$

By Lemma 3.3.12, whenever $D^{\widehat{k}}_{\mathrm{FS}}(\overline{S_{j-1}})$ and $D^{\widehat{k}}_{\mathrm{FS}}(S_j)$ are large (i.e. for long

enough prefixes of S) we have that there exists a k' such that

$$\begin{aligned} D^{k'}_{\mathrm{FS}}(\overline{S_j}) &\leq 2D^{\widehat{k}}_{\mathrm{FS}}(\overline{S_{j-1}}) + D^{\widehat{k}}_{\mathrm{FS}}(S_j) + 2 && \text{(by Lemma 3.3.12)} \\ &\leq 2|\overline{S_{j-1}}| + \frac{|S_j|}{|r_k|} + 2 && \text{(by (3.30))} \\ &= 2(\log(|S_j|) + 1) + \frac{|S_j|}{|r_k|}. && (3.31) \end{aligned}$$

Let $0 < \varepsilon < 1$. Using that $\lim_{k\to\infty}(4k+1)/|r_k| = 0$, for sufficiently long prefixes of the form $\overline{S_j}$ where k is devoted to j, for k sufficiently large it holds that

$$\begin{aligned} D^{k}_{\mathrm{FS}}(\overline{S_j}) - D^{k'}_{\mathrm{FS}}(\overline{S_j}) &\geq \frac{|S_j|}{|r_k|}(|r_k| - 4k - 1) \\ &\quad - 2(\log(|S_j|) + 1) && \text{(by (3.30) and (3.31))} \\ &\geq |S_j|(1 - \frac{\varepsilon}{2}) && \text{(for } j \text{ and } k \text{ sufficiently large)} \\ &= (|\overline{S_j}| - \log|S_j|)(1 - \frac{\varepsilon}{2}) \\ &\geq |\overline{S_j}|(1-\varepsilon). && (3.32) \end{aligned}$$

Similarly as Equation (3.32) only holds for large k, for all $i \leq k$ where k is large, when k' is chosen as above we have that $D^{i}_{\mathrm{FS}}(\overline{S_j}) - D^{k'}_{\mathrm{FS}}(\overline{S_j}) \geq |\overline{S_j}|(1-\varepsilon)$ when j is devoted to k. Hence S is i.o. FS-deep.

Next we show S is not a.e. FS-deep, i.e.

$$(\exists k \in \mathbb{N})(\forall \alpha > 0)(\forall k' \in \mathbb{N})(\exists^{\infty} n \in \mathbb{N})\, D^{k}_{\mathrm{FS}}(S \upharpoonright n) - D^{k'}_{\mathrm{FS}}(S \upharpoonright n) < \alpha n.$$

Throughout the remainder of the proof we assume j is odd. Recall that for all odd j, $K(S_j) \geq |S_j|$.

Fix some k'. Let $a \geq 3$ be large enough such that for some $T \in \mathrm{FST}^{\leq ak'}$, there exists a $y \in \{0,1\}^*$ where $|y| = D^{ak'}_{\mathrm{FS}}(S_j)$ and $T(y) = S_j$. Furthermore, consider the machine M that on input of strings of the form $d(\sigma)01x$, where σ

is an encoding of an FST, M runs x on said FST and returns the same output. Then from an encoding σ_T of T, we have that $M(d(\sigma_T)01y) = S_j$. Hence for all k' we have

$$|S_j| \leq K(S_j) \leq 2ak' + 2 + D_{\mathrm{FS}}^{ak'}(S_j) + O(1) = D_{\mathrm{FS}}^{ak'}(S_j) + O(1). \tag{3.33}$$

Let $0 < \beta < 1$. Hence by Equation (3.33), whenever j is sufficiently large it holds that

$$D_{\mathrm{FS}}^{ak'}(S_j) > |S_j|(1 - \frac{\beta}{2}). \tag{3.34}$$

By Lemma 3.3.9, whenever j is sufficiently large we therefore have that

$$\begin{aligned} D_{\mathrm{FS}}^{k'}(\overline{S_j}) &\geq D_{\mathrm{FS}}^{3k'}(S_j) \geq D_{\mathrm{FS}}^{ak'}(S_j) > |S_j|(1 - \frac{\beta}{2}) && \text{(by (3.34))} \\ &= (|\overline{S_j}| - \log(|S_j|))(1 - \frac{\beta}{2}) \\ &> |\overline{S_j}|(1 - \beta). && (3.35) \end{aligned}$$

Let k be such that the identity transducer is in $\mathrm{FST}^{\leq k}$. Hence by Equation (3.35), for infinitely many odd j it holds that

$$D_{\mathrm{FS}}^{k}(\overline{S_j}) - D_{\mathrm{FS}}^{k'}(\overline{S_j}) < |\overline{S_j}| - |\overline{S_j}|(1 - \beta) = |\overline{S_j}|\beta. \tag{3.36}$$

Then as β and k' were chosen arbitrarily, it holds that S is not a.e. FS-deep.

□

3.4 Summary

In this chapter we introduced a new notion of depth at a low complexity level called almost everywhere finite-state depth. This notion expanded upon a previous finite-state notion of Doty and Moser [57]. We demonstrated that our notion

satisfied some of the fundamental properties of depth. These included that finite-state incompressible sequences are not a.e. FS-deep in Theorem 3.3.5 and that a slow growth law is satisfied in Theorem 3.3.8. We proved the existence of an a.e. FS-deep sequence in Theorem 3.3.13 and showed how our notion was different from Doty and Moser's original notion in Theorem 3.3.14.

We will revisit i.o. FS-depth when we examine other notions in later chapters.

Chapter 4

Pushdown Depth

4.1 Introduction

In this chapter we develop a new notion of depth based on information lossless pushdown compressors (ILPDCs). ILPDCs are more powerful that lossless finite-state compressors in that they have access to an additional data type known as a *stack* to aid in compression. Using the stack, the ILPDC stores characters in its memory, however the compressor is only able to access this memory in a *last in, first out* way.

In the spirit of Bennett's depth which looks at Kolmogorov Complexity against its weaker counterpart of time bounded Kolmogorov complexity, we will define pushdown depth in this chapter by comparing pushdown compressors (PDCs) against unary-stack pushdown compressors (UPDCs). A UPDC is the same as an ordinary PDC with the key difference in this chapter being that a UPDC can only push the symbol 0 onto its stack. In contrast a PDC can push both the symbols 0 and 1.

We will show that there exists deep sequences in our pushdown notion. We will similarly show that ILUPDC-trivial and ILPDC-incompressible sequences are not pushdown deep. We will also show that a slow growth law holds in our

notion.

We tie this chapter back to Chapter 3 also by demonstrating the existence of a sequence which is pushdown deep but not i.o. FS-deep and vice versa. This separates the two notions.

4.2 Pushdown Compressors

The model of pushdown compressors (PDC) we use to define pushdown depth can be found in [109] where PDCs were referred to as *bounded pushdown compressors*. We use this model as it allows for feasible run times by bounding the number of times a PDC can pop a bit from its stack without reading an input bit. This prevents the compressor spending an arbitrarily long time altering its stack without reading its input. This model of pushdown compression also has the nice property that it is equivalent to a notion of pushdown-dimension based on bounded pushdown gamblers [1]. Similar models where there is no bound on the number of times a bit can be popped off from the stack can be found in [2, 58].

The following contains details of the model used. It is taken from [109].

Definition 4.2.1. A *pushdown compressor (PDC)* is a 7-tuple

$$C = (Q, \Gamma, \delta, \nu, q_0, z_0, c)$$

where

1. Q is a non-empty, finite set of *states*,
2. $\Gamma = \{0, 1, z_0\}$ is the finite *stack alphabet*,
3. $\delta : Q \times \{0, 1, \lambda\} \times \Gamma \to Q \times \Gamma^*$ is the *transition function*,
4. $\nu : Q \times \{0, 1, \lambda\} \times \Gamma \to \{0, 1\}^*$ is the *output function*,

5. $q_0 \in Q$ is the *initial state*,

6. $z_0 \in \Gamma$ is the special *bottom of stack symbol*,

7. $c \in \mathbb{N}$ is an upper bound on the number of λ-transitions per input bit.

We write δ_Q and $\delta_{\Gamma*}$ to represent the projections of the function δ. For $z \in \Gamma^+$ the stack of C, z is ordered such that $z[0]$ is the topmost symbol of the stack and $z[|z|-1] = z_0$. δ is restricted to prevent z_0 being popped from the bottom of the stack. That is, for every $q \in Q$, $b \in \{0, 1, \lambda\}$, either $\delta(q, b, z_0) = \perp$, or $\delta(q, b, z_0) = (q', vz_0)$ where $q' \in Q$ and $v \in \Gamma^*$.

Note that δ accepts λ as a valid input symbol. This means that C has the option to pop the top symbol from its stack and move to another state without reading an input bit. This type of transition is call a λ-*transition*. In this scenario $\delta(q, \lambda, a) = (q', \lambda)$. To enforce determinism, we ensure that one of the following hold for all $q \in Q$ and $a \in \Gamma$:

- $\delta(q, \lambda, a) = \perp$, or
- $\delta(q, b, a) = \perp$ for all $b \in \{0, 1\}$.

This means that the compressor does not have a choice to read either 0 or 1 characters. To prevent an arbitrary number of λ-transitions occurring at any one time, we restrict δ such that at most c λ-transitions can be performed in succession without reading an input bit.

The transition function is extended to $\delta' : Q \times \{0, 1, \lambda\} \times \Gamma^+ \to Q \times \Gamma^*$ and is defined recursively as follows. For $q \in Q$, $v \in \Gamma^*$, $a \in \Gamma$ and $b \in \{0, 1, \lambda\}$

$$\delta'(q, b, av) = \begin{cases} (\delta_Q(q, b, a), \delta_{\Gamma^*}(q, b, a)v), & \text{if } \delta(q, b, a) \neq \perp; \\ \perp & \text{otherwise.} \end{cases}$$

For readability, we abuse notation and write δ instead of δ'. The transition function is extended further to $\delta'' : Q \times \{0,1\}^* \times \Gamma^+ \to Q \times \Gamma^*$ as follows. For $q \in Q$, $v \in \Gamma^+$, $w \in \{0,1\}^*$ and $b \in \{0,1,\lambda\}$

$$\delta''(q,\lambda,v) = \begin{cases} \delta''(\delta_Q(q,\lambda,v),\lambda,\delta_{\Gamma^*}(q,\lambda,v)), & \text{if } \delta(q,\lambda,v) \neq \perp; \\ (q,v) & \text{otherwise,} \end{cases}$$

$$\delta''(q,wb,v) = \begin{cases} \delta''(\delta_Q(\delta''_Q(q,w,v),b,\delta''_{\Gamma^*}(q,w,v)),\lambda,\delta_{\Gamma^*}(\delta''_Q(q,w,v),b,\delta''_{\Gamma^*}(q,w,v)), \\ \quad \text{if } \delta''(q,w,v) \neq \perp \text{ and } \delta(\delta''_Q(q,w,v),b,\delta''_{\Gamma^*}(q,w,v)) \neq \perp; \\ \perp, \text{ otherwise.} \end{cases}$$

In an abuse of notation we write δ for δ'' and $\delta(w)$ for $\delta(q_0,w,z_0)$.

We define the *output* from state $q \in Q$ on input $w \in \{0,1\}^*$ with stack contents $v \in \Gamma^*$ by the recursion $\nu(q,\lambda,v) = \lambda$ and

$$\nu(q,wb,v) = \nu(q,w,v) \cdot \nu(\delta_Q(q,w,v),b,\delta_{\Gamma^*}(q,w,v)).$$

The *output* of C on input $w \in \{0,1\}^*$ is denoted by the string $C(w) = \nu(q_0,w,z_0)$.

To define our notion of depth, we examine the class of *information lossless pushdown compressors*.

Definition 4.2.2. A PDC C is *information lossless (IL)* if for all $x \in \{0,1\}^*$, the function $x \mapsto (C(x),\delta_Q(x))$ is injective.

In other words, a PDC C is IL if the output and final state of C on input x uniquely identify x. We call a PDC that is IL an ILPDC. We write (IL)PDC to denote the set of all (IL)PDCs. By the identity PDC, we mean the ILPDC I_{PD} where on every input x, $I_{\mathrm{PD}}(x) = x$.

As part of our definition of pushdown depth, we examine ILPDCs whose stack is limited to containing only the symbol 0 also.

4.2.1 Unary-stack Pushdown Compressors

UPDCs are similar to counter compressors as seen in [11]. The difference here is that for a UPDC, only a single 0 can be popped from the stack during a single transition, while for a counter transducer, an arbitrary number of 0s can be popped from its stack on a single transition, i.e. its counter can be deducted by an arbitrary amount. However, the UPDC has the ability to pop off 0s from its stack without reading a symbol via λ-transitions while the counter compressor cannot. Thus, if a counter compressor decrements its counter by the value of k on a single transition, a UPDC can do the same by performing $k-1$ λ-transitions in a row before reading performing the transition of the counter compressor and popping off the final 0.

Definition 4.2.3. A *unary-stack pushdown compressor (UPDC)* is a 7-tuple

$$C = (Q, \Gamma, \delta, \nu, q_0, z_0, c)$$

where Q, δ, ν, q_0, z_0 and c are all defined the same as for a PDC in Definition 4.2.1, while the *stack alphabet* Γ is the set $\{0, z_0\}$.

Definition 4.2.4. A UPDC C is *information lossless (IL)* if for all $x \in \{0,1\}^*$, the function $x \mapsto (C(x), \delta_Q(x))$ is injective. A UPDC which is IL is referred to as an ILUPDC.

We make the following observation regarding ILUPDCs. Let $C \in$ ILUPDC and suppose it has been given the input yx. After reading the prefix y, if C's stack height is large enough such that it never empties on reading the suffix x, the actual height of the stack doesn't matter. That is, any reading of x with an

arbitrarily large stack which is far enough away from being empty will all have a similar behaviour if starting in the same state. This is because if the stack does not empty, it has little impact on the processing of x. We describe this below.

Remark 4.2.5. Let $C \in$ ILUPDC and suppose C can perform at most c λ-transitions in a row. Consider running C on an input of the form yx and let q be the state C ends in after reading y. If C's stack has a height above $(c+1)|x|$ after reading y, then C's stack can never be fully emptied upon reading x. Hence, for $k, k' \geq (c+1)|x|$ with $k \neq k'$ then $C(q, x, 0^k z_0) = C(q, x, 0^{k'} z_0)$, i.e. C will output the same string regardless of whether the height is k or k'. Thus, prior to reading x, only knowing whether the stack's height is below $(c+1)|x|$ will have any importance.

To examine the complexity of a sequence S from the perspective of pushdown compressors, we are interested in the best case and worst case compression ratios of S via pushdown compressors. For a compressor $C \in$ ILPDC (similarly for $C \in$ ILUPDC), we write $\rho_C(S)$ to denote the *best-case compression ratio* of S via C and $R_C(S)$ to denote the *worst-case compression ratio* of C via S where $\rho_C(S)$ and $R_C(S)$ are defined as in Definition 2.4.4.

We similarly examine the compression ratio of a sequence S over the family of all ILPDCs. In this case we write $\rho_{\mathrm{PD}}(S)$ for the the *best-case* compression ratio of S over all ILPDCs and $R_{\mathrm{PD}}(S)$ for the *worst-case compression ratios* of S over all ILPDCs. Here, $\rho_{\mathrm{PD}}(S)$ and $R_{\mathrm{PD}}(S)$ are defined as in Definition 2.4.5 with the class F being ILPDCs. Note we write $\rho_{\mathrm{PD}}(S)$ and $R_{\mathrm{PD}}(S)$ instead of $\rho_{\mathrm{ILPDC}}(S)$ and $R_{\mathrm{ILPDC}}(S)$ purely for neatness of notation. $\rho_{\mathrm{UPD}}(S)$ and $R_{\mathrm{UPD}}(S)$ are similarly defined when F is taken to be the class ILUPDC. Again we write $\rho_{\mathrm{UPD}}(S)$ and $R_{\mathrm{UPD}}(S)$ instead of $\rho_{\mathrm{ILUPDC}}(S)$ and $R_{\mathrm{ILUPDC}}(S)$ purely for notational reasons.

4.3 Pushdown Depth and its Properties

We now present our notion of pushdown depth (PD-depth). It is an almost everywhere notion. Our current definition examines the difference between the performance between all ILUPDCs and an ILPDC C'. Intuitively, a sequence S is pushdown deep if S contains some structure which ILUPDCs cannot exploit during compression due to their stack restriction while C' can.

Definition 4.3.1. Let $S \in \{0,1\}^\omega$. S is *pushdown deep (PD-deep)* if $(\exists \alpha > 0)(\forall C \in \text{ILUPDC})(\exists C' \in \text{ILPDC})(\forall^\infty n \in \mathbb{N})$

$$|C(S \upharpoonright n)| - |C'(S \upharpoonright n)| \geq \alpha n.$$

The following theorem shows that PD-depth satisfies two of the fundamental depth properties in that both ILUPDC-trivial sequences (in the sense that $R_{\text{UPD}}(S) = 0$) and ILPDC-incompressible sequences (in the sense that $\rho_{\text{PD}}(S) = 1$) are not PD-deep. This is analogous to computable and ML-random sequences being shallow in Bennett's depth.

Theorem 4.3.2. *Let $S \in \{0,1\}^\omega$.*

1. *If $\rho_{\text{PD}}(S) = 1$, then S is not* PD*-deep.*
2. *If $R_{\text{UPD}}(S) = 0$, then S is not* PD*-deep.*

Proof. Let $S \in \{0,1\}^\omega$ be such that $\rho_{\text{PD}}(S) = 1$. Therefore for every $\alpha > 0$ and every $C \in \text{ILPDC}$, for almost every n

$$|C(S \upharpoonright n)| > n(1-\alpha). \tag{4.1}$$

Then for almost every n

$$|I_{\text{PD}}(S \upharpoonright n)| - |C(S \upharpoonright n)| < n - n(1-\alpha) = \alpha n. \tag{4.2}$$

As α is arbitrary and $I_{\mathrm{PD}} \in \mathrm{ILUPDC}$, S is therefore not PD-deep.

Next suppose $S \in \{0,1\}^\omega$ is such that $R_{\mathrm{UPD}}(S) = 0$. Let $C \in \mathrm{ILUPDC}$ be such that $\limsup_{n\to\infty} |C(S \upharpoonright n)|/n = 0$. Hence for every $\beta > 0$ and almost every n,

$$|C(S \upharpoonright n)| < \beta n. \tag{4.3}$$

Therefore for every $C' \in \mathrm{ILPDC}$, it holds that for almost every n

$$|C(S \upharpoonright n)| - |C'(S \upharpoonright n)| \leq |C(S \upharpoonright n)| < \beta n. \tag{4.4}$$

As β is arbitrary, S is not PD-deep.

□

4.3.1 Slow Growth Law

Before we prove a slow growth law for pushdown depth, we first demonstrate that the composition of any ILPDC (or ILUPDC) C with any ILFST T can be simulated by another ILPDC (or ILUPDC) N which is allowed to perform more λ-transitions than C.

Lemma 4.3.3. *Given $C \in$ ILPDC (similarly $C \in$ ILUPDC) and $T \in$ ILFST, we can build an* ILPDC *(similarly an* ILUPDC*) N, such that $\forall x \in \{0,1\}^*$, $N(x) = C(T(x))$.*

Proof. Let $T = (Q_T, q_{0,T}, \delta_T, \nu_T)$ and $C = (Q_C, \Gamma_C, \delta_C, \nu_C, q_{0,C}, z_0, c)$ be the ILFST and the ILPDC respectively as stated in the lemma. Let $d = \max\{|T(q,b)| : q \in Q_T, b \in \{0,1\}\}$ denote the longest output possible from a transition in T. We build the PDC $N = (Q_N, \Gamma_C, \delta_N, \nu_N, q_{0,N}, z_0, cd)$, where

- $Q_N = Q_C \times Q_T \times S$, where $S = \{0,1\}^{\leq cd}$,
- $q_{0,N} = (q_{0,C}, q_{0,T}, \lambda)$.

N works as follows: Before reading a bit, N uses λ-transitions to pop the topmost cd bits of its stack, or until the stack only contains z_0, and remembers them in its states. That is, while $|y| < cd$ and $a \neq z_0$,

$$\delta_N((q_C, q_T, y), \lambda, a) = ((q_C, q_T, ya), \lambda).$$

On such states,

$$\nu_N((q_C, q_T, y), \lambda, a) = \lambda.$$

Then for $b \in \{0, 1\}$, if $a = z_0$ or $|y| = cd$, N moves to the state representing how C would move on input $\nu_T(q_T, b)$, how T would move on input b, and to the state representing not having the topmost stack bits in memory. N's stack then updates to be the same as C's would be as if it had read $\nu_T(q_T, b)$. That is,

$$\delta((q_C, q_T, y), b, a) = ((\delta_{C,Q}(q_C, \nu_T(q_T, b), ya), \delta_{T,Q}(q_T, b), \lambda), xa)$$

where for some $w \in \{0, 1\}^*$ either

1. $x = wy$, if C would have pushed w onto its stack reading $\nu_T(q_T, b)$,
2. $x = wy[i \dots |y| - 1]$, if C would have popped off the top i symbols and then pushed w onto its stack reading $\nu_T(q_T, b)$,
3. $x = y[i \dots |y| - 1]$, if C would have popped off the top i symbols from its stack and pushed nothing on when reading $\nu_T(q_T, b)$.

As there are only a finite number of possibilities, these can all be coded into the states and transitions. On such states,

$$\nu_N((q_C, q_T, y), b, a) = \nu_C(q_C, \nu_T(q_T, b), ya).$$

N is an ILPDC as from knowledge of the output and q_C, we can recover $T(x)$

as C is IL, and from q_T and $T(x)$ we can recover x as T is IL.

Note that if $C \in$ ILUPDC, the proof remains the same except $S = \{0\}^{\leq cd}$ and w mentioned above is an element of $\{0\}^*$.

□

The following result shows that pushdown depth satisfies a slow growth law.

Theorem 4.3.4 (Slow Growth Law)**.** *Let $S \in \{0,1\}^\omega$, let $g : \{0,1\}^\omega \to \{0,1\}^\omega$ be ILFS computable and let $S' = g(S)$. If S' is PD-deep then S is PD-deep.*

Proof. Let S, S', f, and g be as in the statement of the lemma and let T be an ILFST computing g.

For all m, let m_n denote the largest integer such that

$$T(S \upharpoonright m) = T(S \upharpoonright m_n) = S' \upharpoonright n.$$

As T is IL, it cannot visit the same state twice without outputting at least one bit, so there exists a $\beta > 0$ such that for all n, $n \geq \beta m_n$. Furthermore recall from Theorem 3.2.4 that there exists an ILFST T^{-1} and a constant a such that for all $x \in \{0,1\}^*$, $x \upharpoonright (|x| - a) \sqsubseteq T^{-1}(T(x)) \sqsubseteq x$.

Let $C \in$ ILUPDC. Let N be the ILUPDC given by Lemma 4.3.3 such that $N(x) = C(T^{-1}(x))$ for all x. Note then that for some n,

$$\begin{aligned} |C(S \upharpoonright m)| &\geq |C(T^{-1}(T(S \upharpoonright m))| = |N(T(S \upharpoonright m))| \\ &= |N(T(S \upharpoonright m_n))| = |N(S' \upharpoonright n)|. \end{aligned} \tag{4.5}$$

As S' is deep, there exists $\alpha > 0$ and an ILPDC N' such that for almost every n,

$$|N(S' \upharpoonright n)| - |N'(S' \upharpoonright n)| \geq \alpha n. \tag{4.6}$$

Next let C' be the ILPDC given by Lemma 4.3.3 such that on input x, $C'(x) =$

$N'(T(x))$. Hence for some n,

$$|C'(S \upharpoonright m)| = |N'(T(S \upharpoonright m))| = |N'(T(S \upharpoonright m_n))| = |N'(S' \upharpoonright n)|. \tag{4.7}$$

Therefore for almost every $m \in \mathbb{N}$, there exists some n such that

$$\begin{aligned} |C(S \upharpoonright m)| - |C'(S \upharpoonright m)| &\geq |N(S' \upharpoonright n)| - |N'(S' \upharpoonright n)| && \text{(by (4.5) and (4.7))} \\ &\geq \alpha n && \text{(by (4.6))} \\ &\geq \alpha\beta m_n \geq \alpha\beta m. && (4.8) \end{aligned}$$

Hence S is PD-deep.

□

4.4 Separation from Finite-State Depth

Prior to comparing pushdown depth with Doty and Moser's i.o. finite-state depth, we require the following definition which defines the *depth-level* of a sequence. Simply put, the depth-level of a deep sequence is the α term in Definitions 3.3.1 and 4.3.1. Note that we exclusively refer to i.o. FS-depth here and drop the i.o. notation.

Definition 4.4.1. Let $S \in \{0,1\}^\omega$. Let $\alpha > 0$.

1. We say that FS-*depth*$(S) \geq \alpha$ if

$$(\forall k \in \mathbb{N})(\exists k' \in \mathbb{N})(\exists^\infty n \in \mathbb{N})\, D^k_{\mathrm{FS}}(S \upharpoonright n) - D^{k'}_{\mathrm{FS}}(S \upharpoonright n) \geq \alpha n.$$

   Otherwise we say FS-*depth*$(S) < \alpha$.

2. We say that PD-*depth*$(S) \geq \alpha$ if

$$(\forall C \in \mathrm{ILUPDC})(\exists C' \in \mathrm{ILPDC})(\forall^\infty n \in \mathbb{N})\, |C(S \upharpoonright n)| - |C'(S \upharpoonright n)| \geq \alpha n.$$

   Otherwise we say PD-*depth*$(S) < \alpha$.

The following result demonstrates the existence of a sequence which has a large FS-depth level but not even a small PD-depth level. This sequence is composed of chunks of random strings which grow exponentially. Some of these chunks are composed of repetitions of random strings which small FSTs cannot identify while larger FSTs can, resulting in finite-state depth. Specifically, since the chunks grow exponentially in size, this means that for infinitely many prefixes the finite-state depth of the sequence we construct gets arbitrarily close to 1. Other chunks x are such that $K(x) \geq |x|$ preventing the sequence being PD-deep. Specifically, these random chunks result in pushdown compressors achieving little compression on infinitely many of the prefixes of the sequence. This means that the sequence's pushdown depth gets arbitrarily close to 0. The construction takes advantage of the fact that one is an i.o. notion (Moser and Doty's notion) while the other is an a.e. notion.

Theorem 4.4.2. *There exists a sequence S such that for all $0 < \alpha < 1$,* FS-*depth(S)* $> (1 - \alpha)$ *and* PD-*depth(S)* $< \alpha$.

Proof. Let $0 < \alpha < 1$. We claim that the sequence S constructed in Theorem 3.3.14 satisfies this theorem. We give a brief recap of the construction of S:

Begin by partitioning the non-negative integers into disjoint consecutive intervals $I_1, I_2, \ldots$ where $|I_1| = 2$ and $|I_j| = 2^{|I_1|+\cdots+|I_{j-1}|}$ for $j \geq 2$. For instance, $I_1 = \{0, 1\}$, $I_2 = \{2, 3, 4, 5\}$, and $I_3 = \{6, 7, \ldots, 69\}$. For each I_j, set $m_j = \min(I_j)$ and $M_j = \max(I_j)$. Note that for all j, $m_{j+1} = M_j + 1$. S is constructed in stages $S_1 S_2 S_3 \ldots$, where S_j denotes the substring $S[m_j..M_j]$. For all j, we henceforth use the notation $\overline{S_j}$ to denote the prefix $S_1 \cdots S_j$ of S. Note that $|S_j| = |I_j|$ and when $j > 1$ we have that $\log(|S_j|) = |\overline{S_{j-1}}|$. Each stage S_j is constructed as follows:

For every interval I_j where j is odd, we set S_j to be a string with maximal plain Kolmogorov complexity in the sense that $K(S_j) \geq |S_j|$. If j is even, I_j is devoted to some FST description bound length k. Specifically for each k, k is devoted to every interval I_j where j is of the form $j = 2^k + t(2^{k+1})$, for $t \geq 0$. So $k = 1$ is first devoted to I_2 and every 4^{th} interval after that and $k = 2$ is first devoted to I_4 and every 8^{th} interval after that, and so on. For each k, let r_k be a string of length $|I_{2^k}|$ which is $3k$-FS random. Then if I_j is devoted to k we set

$$S_j = r_k^{\frac{|I_j|}{|r_k|}}.$$

If we similarly follow the argument in the proof of Theorem 3.3.14 up to Equation (3.32), we therefore have that for all $0 < \alpha < 1$, FS-depth$(S) \geq 1 - \alpha$. This is because the ε term in Equation 3.32 was arbitrarily chosen (so set $\varepsilon = \alpha$) such that $0 < \varepsilon < 1$.

Next we show that PD-depth$(S) < \alpha$: Throughout the remainder of the proof we assume that j is odd.

Let C be any ILPDC. Consider the tuple

$$(\overline{S_{j-1}}, q_s, q_e, \widehat{C}, \bar{\nu_C}(S_j))$$

where q_s is the state C when begins reading S_j, q_e is the state C ends up in after reading S_j, $\widehat{C}$ is an encoding of C in some representation of PDCs and $\bar{\nu_C}(S_j)$ is the suffix of $C(\overline{S_j})$ outputted when reading S_j. Then given this tuple, one can recover S_j as C is information lossless.

Using the fact that tuples of the form $(x_1, x_2, \ldots, x_n)$ can be encoded by the string

$$1^{\lceil \log n_1 \rceil}0n_1x_1 1^{\lceil \log n_2 \rceil}0n_2x_2 \ldots 1^{\lceil \log n_{n-1} \rceil}0n_{n-1}x_{n-1}x_n, \tag{4.9}$$

where $n_i = |x_i|$ in binary, we have that

$$\begin{aligned} |S_j| \leq K(S_j) &\leq |\bar{\nu_C}(S_j)| + 2\log(|\overline{S_{j-1}}|) + |\overline{S_{j-1}}| + O(|\widehat{C}|) + O(1) \quad (\text{as } j \text{ is odd}) \\ &= |\bar{\nu_C}(S_j)| + 2\log(\log(|S_j|)) + \log(|S_j|) + O(|\widehat{C}|) + O(1). \end{aligned} \tag{4.10}$$

Hence for j large we have that

$$|\bar{\nu_C}(S_j)| \geq |S_j| - 2\log(\log(|S_j|)) - \log(|S_j|) - O(|\widehat{C}|) - O(1) > |S_j|(1 - \frac{\alpha}{2}). \tag{4.11}$$

Therefore, for j large we have that

$$\begin{aligned} |C(\overline{S_j})| \geq |\bar{\nu_C}(S_j)| &> |S_j|(1 - \frac{\alpha}{2}) && (\text{by (4.11)}) \\ &= (|\overline{S_j}| - \log(|\overline{S_j}|))(1 - \frac{\alpha}{2}) > |\overline{S_j}|(1 - \alpha). && (4.12) \end{aligned}$$

Hence, for infinitely many prefixes $\overline{S_j}$ of S it holds that

$$|I_{\mathrm{PD}}(\overline{S_j})| - |C(\overline{S_j})| < |\overline{S_j}| - |\overline{S_j}|(1 - \alpha) = \alpha|\overline{S_j}| \tag{4.13}$$

As C was chosen arbitrarily, it holds that PD-depth(S) $< \alpha$.

□

The next result demonstrates the existence of a sequence which achieves a PD-depth of roughly 1/2 while at the same time while having a small finite-state depth level. This sequence is split into chunks of strings of the form RFR^{-1} where F is a flag and R is a string not containing F with large plain Kolmogorov complexity relative to its length. A large ILPDC C is built to push R onto its stack, and then when it sees the flag F, uses its stack to compress R^{-1}. These R are such that an ILUPDC cannot use its stack to compress R, resulting in no compression. For the finite-state transducers, the sequence appears almost random and so little depth is achieved. The sequence from Theorem 3 of [109] satisfies the result as shown in the following theorem.

Theorem 4.4.3. *For all $0 < \beta < 1/2$, there exists a sequence S such that PD-depth$(S) \geq 1/2 - \beta$, and FS-depth$(S) < \beta$.*

Proof. First we shall give the construction of the sequence from Theorem 3 of [109]:

Let $0 < \beta < 1/2$, and let $k > 8$ be a positive integer such that $\beta \geq 8/k$. For each n, let $t_n = k^{\lceil \frac{\log n}{\log k} \rceil}$. Note that for all n,

$$n \leq t_n \leq kn. \tag{4.14}$$

Consider the set T_j which contains all strings of length j that do not contain 1^k as a substring. As T_j contains strings of the form $x_1 0 x_2 0 x_3 0 \cdots$ where each x_t is a string of length $k-1$, we have that $|T_j| \geq 2^{j(1-\frac{1}{k})}$. For each j, let $R_j \in T_{kt_j}$ have maximal plain Kolmogorov complexity in the sense that

$$K(R_j) \geq |R_j|(1 - \frac{1}{k}). \tag{4.15}$$

Such an R_j exists as $|T_{|R_j|}| > 2^{|R_j|(1-\frac{1}{k})} - 1$. Note that $kj \leq |R_j| \leq k^2 j$.

The sequence S which we claim satisfies the theorem is constructed as follows: For each j, set

$$S_j = R_j 1^k R_j^{-1}.$$

First we examine how well any ILUPDC compresses occurrences of R_j zones in S. Let $C \in$ ILUPDC. Consider the tuple

$$(\widehat{C}, q_s, q_e, z, \nu_C(q_s, R_j, z))$$

where $\widehat{C}$ is an encoding of C, q_s is the state that C begins reading R_j in, q_e is the state C ends up in after reading R_j, z is the stack contents of C as it begins reading R_j in q_s (i.e. $z = 0^p z_0$ for some p), and the output $\nu_C(q_s, R_j, z)$ of C on R_j. By Remark 4.2.5, C's stack is only important if $|z| < (c+1)|R_j|$, as if $|z|$ is larger, C will output the same regardless of $|z|$'s true value. Hence, set

$$z' = \begin{cases} |z| & \text{if } |z| < (c+1)|R_j| \\ (c+1)|R_j| & \text{if } |z| \geq (c+1)|R_j|. \end{cases} \tag{4.16}$$

As C is lossless, having knowledge of the tuple $(\widehat{C}, q_s, q_e, z', \nu_C(q_s, R_j, z))$ means we can recover R_j. If we encode the tuple $(\widehat{C}, q_s, q_e, z', \nu_C(q_s, R_j, z))$ the same way as in (4.9), and noting that z' contributes roughly $O(\log |R_j|)$ bits to the encoding, we have we have by Equation (4.15)

$$|R_j|(1 - \frac{1}{k}) \leq K(R_j) \leq |\nu_C(q_s, R_j, z)| + O(\log |R_j|) + O(|\widehat{C}|) + O(1). \tag{4.17}$$

Therefore, for j large we have

$$|\nu_C(q_s, R_j, z)| \geq |R_j|(1 - \frac{1}{k}) - O(\log |R_j|) > |R_j|(1 - \frac{2}{k}). \tag{4.18}$$

This is similarly true for R_j^{-1} zones also as $K(R_j) \leq K(R_j^{-1}) + O(1)$. Hence for j large we see that C outputs at least

$$\begin{aligned} |C(\overline{S_j})| - |C(\overline{S_{j-1}})| &\geq 2|R_j|(1 - \frac{2}{k}) \\ &= (|S_j| - k)(1 - \frac{2}{k}) \\ &\geq |S_j|(1 - \frac{3}{k}) \end{aligned} \tag{4.19}$$

bits when reading S_j.

Next we examine how well C compresses S on arbitrary prefixes. Consider the prefix $S \upharpoonright n$ and let j be such that $\overline{S_j}$ is a prefix of $S \upharpoonright n$ but $\overline{S_{j+1}}$ is not. Thus $S \upharpoonright n = \overline{S_j} \cdot y$ for some $y \sqsubset S_{j+1}$. Suppose Equation (4.19) holds for all $i \geq j'$. Hence we have that

$$\begin{aligned} |C(S \upharpoonright n)| &\geq |C(\overline{S_j})| \geq |C(\overline{S_j})| - |C(\overline{S_{j'-1}})| \\ &\geq |S_{j'} \dots S_j|(1 - \frac{3}{k}) - O(1) && \text{(by (4.19))} \\ &= (n - |y| - |\overline{S_{j'-1}}|)(1 - \frac{3}{k}) - O(1) \\ &\geq (n - |y|)(1 - \frac{4}{k}). \end{aligned} \tag{4.20}$$

Noting that $n = \Omega(j^2)$ and $|y| = O(j)$, by Equation (4.20) we have that

$$|C(S \upharpoonright n)| \geq n(1 - \frac{5}{k}). \tag{4.21}$$

As C was arbitrary, we therefore have that

$$\rho_{\text{UPD}}(S) > 1 - \frac{6}{k}. \tag{4.22}$$

Next we build an ILPDC C' that is able to compress prefixes of S. In [109], it was shown that $R_{\text{PD}}(S) \leq 1/2$. We provide the details of this proof here for

completeness.

For the ILPDC C', informally, C' outputs its input for some prefix $S_1 \dots S_{p-1}$. Then, for all $j \geq p$, C' compresses S_j as follows: On S_j, C' outputs its input on $R_j 1^k$ while trying to identify the 1^k flag. Once the flag is found, C' pops the flag from its stack and then begins to read an R_j^{-1} zone. On R_j^{-1}, C' counts modulo v to output a zero every v bits, and uses its stack to ensure that the input is indeed R_j^{-1}. If this fails, C' outputs an error flag, enters an error state and from then on outputs its input. Furthermore, v is cleverly chosen such that for all but finitely many j, v divides evenly into $|R_j|$. Specifically we set $v = k^a$ for some $a \in \mathbb{N}$. A complete description of C' is provided at the end of this proof for completeness.

Next we will compute the compression ratio of C' on S. We let p be such that for all $j \geq p$, v divides evenly into $|R_j|$. C' will output its input on $\overline{S_{p-1}}$ and begin compressing on the succeeding zones. Also, note that the compression ratio of C' on S is largest on prefixes ending with a flag 1^k. Hence, consider some prefix $\overline{S_{j-1}} R_j 1^k$ of S. We have that for j sufficiently large

$$
\begin{aligned}
\frac{|C(\overline{S_{j-1}} R_j 1^k)|}{|\overline{S_{j-1}} R_j 1^k|} &\leq \frac{|\overline{S_{p-1}}| + \sum_{i=p}^{j} (|R_i| + k + \frac{|R_i|}{v}) - \frac{|R_j|}{k}}{|\overline{S_{j-1}} R_j 1^k|} \\
&\leq \frac{|\overline{S_{p-1}}|}{|\overline{S_{j-1}}|} + \frac{(1+\frac{1}{v}) \sum_{i=1}^{j} k t_i + jk - \frac{k t_j}{v}}{|\overline{S_{j-1}}|} \\
&\leq \frac{1}{6v} + \frac{(1+\frac{1}{v}) \sum_{i=1}^{j} k t_i + jk - \frac{k t_j}{v}}{2k \sum_{i=1}^{j-1} t_i} && \text{(for } j \text{ large)} \\
&\leq \frac{1}{6v} + \frac{(1+\frac{1}{v}) \sum_{i=1}^{j} t_i + j - \frac{t_j}{v}}{2 \sum_{i=1}^{j-1} t_i} \\
&\leq \frac{1}{6v} + \frac{(1+\frac{1}{v}) \sum_{i=1}^{j-1} t_i}{2 \sum_{i=1}^{j-1} t_i} + \frac{t_j}{2 \sum_{i=1}^{j-1} t_i} + \frac{j}{2 \sum_{i=1}^{j-1} t_i} \\
&\leq \frac{1}{6v} + \frac{1}{2} + \frac{1}{2v} + \frac{1}{2}\Big(\frac{jk}{(j-1)(j)/2}\Big) + \frac{1}{2}\Big(\frac{j}{(j-1)(j)/2}\Big) \\
&\leq \frac{1}{6v} + \frac{1}{2} + \frac{1}{2v} + \frac{1}{6v} + \frac{1}{6v} && \text{(for } j \text{ large)} \\
&= \frac{1}{2} + \frac{1}{v}. && (4.23)
\end{aligned}
$$

As v can be chosen to be arbitrarily large, we therefore have that

$$R_{\mathrm{PD}}(S) \leq \frac{1}{2}. \tag{4.24}$$

Hence, for almost every n, by Equations (4.22) and (4.24) it follows that for all $C \in \mathrm{ILUPDC}$,

$$|C(S \upharpoonright n)| - |C'(S \upharpoonright n)| \geq (1 - \frac{6}{k} - \frac{1}{k})n - (\frac{1}{2} + \frac{1}{k})n \tag{4.25}$$

$$= (\frac{1}{2} - \frac{8}{k}). \tag{4.26}$$

Choosing k large such that $\frac{8}{k} \leq \beta$ gives us our desired result of PD-depth$(S) \geq \frac{1}{2} - \beta$.

Next we examine the finite-state depth of S. From Equation (4.22) and Definition 3.2.6, it follows that $\dim_{\mathrm{FS}}(S) > 1 - \frac{6}{k}$. Hence, for all l it follows that for all but finitely many n that

$$D^l_{\mathrm{FS}}(S \upharpoonright n) \geq (1 - \frac{7}{k})n. \tag{4.27}$$

Therefore, for l' such that $I_{\mathrm{FS}} \in \mathrm{F}^{\leq l'}$ we have that for all l and almost every n

$$D^{l'}_{\mathrm{FS}}(S \upharpoonright n) - D^l_{\mathrm{FS}}(S \upharpoonright n) \leq n - (1 - \frac{7}{k})n < \frac{8}{k} \cdot n. \tag{4.28}$$

That is, FS-depth$(S) < \beta$ as desired.

For completeness we now present a full description of the ILPDC C': Let Q be the following set of states:

1. the start state q^s_0,
2. the counting states $q^s_1, \ldots q^s_m$ and q_0 that count up to $m = |\overline{S_{p-1}}|$,
3. the flag checking states $q^{f_1}_1, \ldots, q^{f_1}_k$ and $q^{f_0}_1, \ldots, q^{f_0}_k$,

4. the pop flag states $q_0^F, \ldots, q_k^F$,

5. the compress states $q_1^c, \ldots, q_{v+1}^c$,

6. the error state q_e.

We now describe the transition function of C'. At first, C' counts from q_0^s to q_m^s to ensure that for later R_j zones, v divides evenly into $|R_j|$. That is, for $0 \leq i \leq m-1$,

$$\delta(q_i^s, x, y) = (q_{i+1}^s, y)$$

and

$$\delta(q_m^s, \lambda, y) = (q_0, y).$$

Once this counting has taken place, an R_j zone begins. Here, the input is pushed onto the stack and C' tries to identify the flag 1^k by examining group of k symbols. We set

$$\delta(q_0, x, y) = \begin{cases} (q_1^{f_1}, xy) & \text{if } x = 1 \\ (q_1^{f_0}, xy) & \text{if } x \neq 1 \end{cases}$$

and for $1 \leq i \leq k-1$,

$$\delta(q_i^{f_0}, x, y) = (q_{i+1}^{f_0}, xy)$$

and

$$\delta(q_i^{f_1}, x, y) = \begin{cases} (q_{i+1}^{f_1}, xy) & \text{if } x = 1 \\ (q_{i+1}^{f_0}, xy) & \text{if } x \neq 1. \end{cases}$$

If the flag 1^k is not detected after k symbols, the test begins again. That is

$$\delta(q_k^{f_0}, \lambda, y) = (q_0, y).$$

If the flag is detected, the pop flag state is entered. $\delta(q_k^{f_1}, \lambda, y) = (q_0^F, y)$. The flag is then removed from the stack, that is, for $0 \leq i \leq k-1$

$$\delta(q_i^F, \lambda, y) = (q_{i+1}^F, \lambda)$$

and

$$\delta(q_k^F, \lambda, y) = (q_1^c, y).$$

C' then checks using the stack, that the next part of the input it reads is R_j^{-1}, counting modulo v. If the checking fails, the error state is entered. That is for $1 \leq i \leq v$,

$$\delta(q_i^c, x, y) = \begin{cases} (q_{i+1}^c, \lambda) & \text{if } x = y \\ (q_e, y) & \text{if } x \neq y \text{ and } y \neq z_0 \\ (q_1^{f_1}, xz_0) & \text{if } x = 1 \text{ and } y = z_0 \\ (q_1^{f_0}, xz_0) & \text{if } x \neq 1 \text{ and } y = z_0. \end{cases}$$

Once v symbols are checked, the checking starts again. That is

$$\delta(q_{v+1}^c, \lambda, y) = (q_1^c, y).$$

The error state is the loop

$$\delta(q_e, x, y) = (q_e, y).$$

We now describe the output function of C'. Firstly, on the counting states, C' outputs its input. That is, for $0 \leq i \leq m-1$

$$\nu(q_i^s, x, y) = x.$$

On the flag checking states C' outputs its input. That is, for $1 \leq i \leq k-1$

$$\nu(q_i^{f_0}, x, y) = \nu(q_i^{f_1}, x, y) = x.$$

C' outputs nothing while in the flag popping states $q_0^F, \ldots, q_k^F$ and on the compress states $q_1^c, \ldots, q_{v+1}^c$ except in the case when v symbols have just been checked. That is,

$$\nu(q_v^c, x, y) = 0 \text{ if } x = y.$$

When an error is seen, a flag is outputted. That is for $1 \leq i \leq v$

$$\nu(q_i^c, x, y) = 1^{3m+i}0x \text{ if } x \neq y \text{ and } y \neq z_0.$$

C' outputs its input while in the error state. That is,

$$\nu(q_e, x, y) = x.$$

Lastly we verify that C' is in fact IL. If the final state is not an error state, then all R_j zones and 1^k flags are output as in the input. If the final state is q_i^c then the number t of zeros after the last flag in the output along with q_i^c determines that the last R_j^{-1} zone read is $tv + i - 1$ bits long. If the final state is q_e, then the output is of the form

$$aR_j1^k0^t1^{3m+i}0b$$

for $a, b \in \{0,1\}^*$. The input is uniquely determined to be the input corresponding to the output $aR_j1^k0^t$ with final state q_1^c followed by

$$R_j^{-1}[tv..tv + (i-1) - 1].$$

As 1^{3m} does occur anywhere as a substring of S post the prefix $\overline{S_{p-1}}$, its first occurrence post $\overline{S_{p-1}}$ as part of an output must correspond to an error flag.

□

4.5 Summary

In this chapter we introduced a new notion of depth at a low complexity based on lossless pushdown compressors called pushdown depth. This notion expanded upon the previous finite-state notion of Chapter 3 as pushdown compressors are simply finite-state transducers with an added stack. We demonstrated that our notion satisfies some of the fundamental properties of depth. This included that ILPDC-incompressible and ILUPDC-trivial sequences are not pushdown deep in Theorem 4.3.2 and that a slow growth law is satisfied in Theorem 4.3.4.

Our pushdown depth was based on the difference between ILUPDCs and ILPDCs. We proved the existence of a PD-deep sequence in Theorem 4.4.3. We also compared our pushdown depth with i.o. FS-depth from Chapter 3. In Theorem 4.4.2 we showed there exists i.o. FS-deep sequences which are not pushdown deep and in Theorem 4.4.3 we showed the existence of sequences which are PD-deep, and if they are i.o. FS-deep, they must have low FS-depth levels.

Pushdown depth will be revisited in Chapters 5 and 6.

Chapter 5

Lempel-Ziv Depth

5.1 Introduction

In 1976, Lempel and Ziv presented a new approach to measure the complexity of finite strings based on the number of substrings that a string would be parsed into based on certain constraints [98]. They noted that their parsing complexity may have use in compression and subsequently developed two lossless compression algorithms in 1977 and 1978, commonly referred to as Lempel-Ziv 77 (LZ77) and Lempel-Ziv 78 (LZ78) respectively [156, 157]. For a parsed string, both algorithms achieve compression by using a pointer to indicate previous instances of a phrase in the parsing. LZ77 uses a sliding window to keep track of past phrases while LZ78 builds a dictionary.

Both algorithms have been shown to be very powerful for certain families of inputs. For instance, for input sequences formed by finite alphabets generated by stationary, ergodic sources, both algorithms are optimal, i.e. their compression rate approaches the entropy of the input when the size of the sliding window and dictionary are allowed to get infinitely large [131, 152, 157]. Both algorithms are called *universal* as they do not need to know anything about the statistics of the source to compress, unlike say Huffman encoding [81]. Their success has led to a

number of variations of the algorithms being developed and studied, such as in [94, 147, 150, 151].

In this chapter we develop a notion of depth based on the LZ78 compression algorithm. Our notion examines the difference in compression between ILFSTs (referred to as finite-state compressors in this chapter) and the LZ78 algorithm. We propose that LZ78 is a good choice to compare against ILFSTs as it asymptotically reaches the lower bound of compression attained by any finite-state compressor [131, 157].

We show that our notion of Lempel-Ziv depth satisfies some of the expected fundamental properties of depth. We also compare it with finite-state depth discussed in Chapter 3 and pushdown depth of Chapter 4. We do this by demonstrating the existence of sequences which are Lempel-Ziv deep which are neither a.e. nor i.o. finite-state deep. We then show there exists a sequence which is Lempel-Ziv deep but not pushdown deep. We also demonstrate the existence of a sequence with a PD-depth level of roughly 1/2 and a low Lempel-Ziv depth level.

5.2 The Lempel-Ziv 78 Algorithm

The Lempel-Ziv algorithm LZ78 (henceforth denoted by LZ unless clearly stated otherwise) [157] is a lossless dictionary based compression algorithm. Given an input $x \in \{0,1\}^*$, LZ parses x into phrases $x = x_1 x_2 \dots x_n$ such that each phrase x_i is unique in the parsing, except for possibly the last phrase. Furthermore, for each phrase x_i, every prefix of x_i also appears previously as a phrase in the parsing. That is, if $y \sqsubset x_i$, then $y = x_j$ for some $j < i$. Each phrase is stored in LZ's dictionary. LZ encodes x by encoding each phrase as a pointer to its dictionary containing the longest proper prefix of the phrase along with the final bit of the phrase. Specifically for each phrase x_i, $x_i = x_{l(i)} b_i$ for $l(i) < i$ and

$b_i \in \{0,1\}$. Then for $x = x_1 x_2 \dots x_n$

$$\mathrm{LZ}(x) = c_{l(1)} b_1 c_{l(2)} b_2 \dots c_{l(n)} b_n,$$

where c_i is an encoding of the pointer to the i^{th} element of LZ's dictionary, and $x_0 = \lambda$. We restrict LZ's input to binary strings. We note that LZ being asymptotically optimal means that in the worst case scenario, strings of length n compress to $n + o(n)$ bits.

For instance $y_1 = 0100000101000011000$ would be parsed as

$$0, 1, 00, 000, 10, 100, 001, 1000.$$

Best case scenario for LZ is if the input string is parsed in such a way such that for all i, phrase x_i is a prefix of x_{i+1} (except for maybe the last phrase). For instance $y_2 = 00101001000100101000$ parses as

$$0, 01, 010, 0100, 01001, 0100.$$

Note in this case that each phrase x_i has length i (except for maybe the last phrase). Worst case scenario for LZ if its input is a concatenation of all binary strings in order of length. For instance $y_3 = 0100011011000001010$ parses as

$$0, 1, 00, 01, 10, 11, 000, 001, 010.$$

We have that $|y_1| = |y_2| = |y_3| = 19$ yet y_1 is parsed into 8 phrases, y_2 into 6 phrases and y_3 into 9 phrases.

For strings $w = xy$, we let $\mathrm{LZ}(y|x)$ denote the output of LZ on y after it has already parsed x. For strings of the form $w = xy^n$, we use Lemma 1 from [109] to get an upper bound for $|\mathrm{LZ}(y^n|x)|$.

Lemma 5.2.1 ([109]). *Let $n \in \mathbb{N}$, and $x, y \in \{0,1\}^*$ where $y \neq \lambda$. Let $w = xy^n$. Suppose on its computation of the string w that* LZ*'s dictionary contained $d \geq 0$ phrases after reading x. Then we have that*

$$|\mathrm{LZ}(y^n|x)| \leq \sqrt{2(|y|+1)|y^n|}\log\left(d + \sqrt{2(|y|+1)|y^n|}\right).$$

We occasionally make reference to the best-case compression ratio of LZ on sequences. These are as defined in Definition 2.4.5 where the class F contains only the LZ algorithm.

Definition 5.2.2. The LZ-*upper* and LZ-*lower* compression ratios of S are respectively given by

$$\rho_{\mathrm{LZ}}(S) = \liminf_{n\to\infty} \frac{|\mathrm{LZ}(S \upharpoonright n)|}{n}, \text{ and } R_{\mathrm{LZ}}(S) = \limsup_{n\to\infty} \frac{|\mathrm{LZ}(S \upharpoonright n)|}{n}.$$

5.3 Lempel-Ziv Depth and its Properties

We now present our notion of *Lempel-Ziv depth (*LZ*-depth)*. Our current definition examines the difference between the performance of ILFSTs (i.e. finite-state compressors) and the LZ algorithm. Intuitively a sequence is Lempel-Ziv deep if given any ILFST, the compression difference between the ILFST and the LZ algorithm is bounded below by a constant times the length of the prefix examined.

Definition 5.3.1. A sequence S is *(almost everywhere) Lempel-Ziv deep ((a.e.) LZ-deep)* if

$$(\exists \alpha > 0)(\forall C \in \mathrm{ILFST})(\forall^\infty n \in \mathbb{N}), |C(S \upharpoonright n)| - |\mathrm{LZ}(S \upharpoonright n)| \geq \alpha n.$$

We say a sequence is *infinitely often (i.o.) LZ-deep* if the $(\forall^\infty n \in \mathbb{N})$ term in the above definition is replaced with $(\exists^\infty n \in \mathbb{N})$. We usually use the i.o. notation

to denote i.o. LZ-depth but do not use the a.e. notation to denote a.e. LZ-depth. Therefore, one can assume LZ-depth always refers to the a.e. notion.

We note that a Lempel-Ziv notion of depth could similarly be defined based on the Lempel-Ziv 77 algorithm. For instance, in Chapter 7 we will later encounter sequences that are Lempel-Ziv 77 deep if the sliding window was allowed to have infinite length. Furthermore, instead of a single LZ-style algorithm, LZ-depth could be defined over a family of LZ-style algorithms. We proceed content with our current definition based on LZ78.

Prior to comparing LZ-depth with other notions, we require the following definition which describes the *depth-level* of a sequence. This is similar to Definition 4.4.1.

Definition 5.3.2. Let $S \in \{0,1\}^\omega$. Let $\alpha > 0$. We say that LZ-*depth*$(S) \geq \alpha$ if

$$(\forall C \in \text{ILFST})(\forall^\infty n \in \mathbb{N})\, |C(S \upharpoonright n)| - |\text{LZ}(S \upharpoonright n)| \geq \alpha n.$$

Otherwise we say that LZ-*depth*$(S) < \alpha$.

We now show that LZ-depth satisfies the property of depth which says that trivial sequences and random sequences are not deep. Here, trivial means sequences which are FST-trivial (i.e. have a finite-state strong dimension of 0 (recall Definition 3.2.6)) and random means LZ-incompressible.

Theorem 5.3.3. *Let $S \in \{0,1\}^\omega$.*

1. *If $\rho_{\text{LZ}}(S) = 1$, then S is not* LZ-*deep.*

2. *If $\text{Dim}_{\text{FS}}(S) = 0$, then S is not* LZ-*deep.*

Proof. The proof follows the same structure as Theorem 4.3.2.

Let $S \in \{0,1\}^\omega$ be such that $\rho_{\text{LZ}}(S) = 1$. Therefore for every $\alpha > 0$, for almost every n

$$|\text{LZ}(S \upharpoonright n)| > n(1-\alpha). \tag{5.1}$$

Then for almost every n

$$|I_{\mathrm{FS}}(S \upharpoonright n)| - |\mathrm{LZ}(S \upharpoonright n)| < n - n(1-\alpha) = \alpha n. \tag{5.2}$$

As α is arbitrary and $I_{\mathrm{FS}} \in \mathrm{ILFST}$, S is not LZ-deep.

Next suppose $S \in \{0,1\}^\omega$ is such that $\mathrm{Dim}_{\mathrm{FS}}(S) = 0$. Hence there exists some $C \in \mathrm{ILFST}$ such that for every $\beta > 0$ and almost every n,

$$|C(S \upharpoonright n)| < \beta n. \tag{5.3}$$

Therefore it holds that for almost every n

$$|C(S \upharpoonright n)| - |\mathrm{LZ}(S \upharpoonright n)| \leq |C(S \upharpoonright n)| < \beta n. \tag{5.4}$$

As β is arbitrary, S is not LZ-deep.

□

5.4 Separation from Finite-State Depth

In the following section we compare LZ-depth with FS-depth of Chapter 3. In our first result we demonstrate the existence of a LZ-deep sequence which is neither deep in either our notion of a.e. FS-depth nor in Doty and Moser's i.o FS-depth. It relies on a result by Lathrop and Strauss which demonstrates the existence of a normal sequence S such that $R_{\mathrm{LZ}}(S) \neq 1$, i.e. a normal sequence Lempel-Ziv can compress [97].

Theorem 5.4.1. *There exists a normal* LZ*-deep sequence.*

Proof. Let S be the normal sequence from Theorem 4.3 of [97] such that $R_{\mathrm{LZ}}(S) = \varepsilon < 1$. We claim this sequence satisfies the theorem.

Let $\delta > 0$ be such that $\varepsilon + \delta < 1$. Therefore, for almost every n it holds that

$$|\mathrm{LZ}(S \upharpoonright n)| \leq (\varepsilon + \frac{\delta}{2})n. \tag{5.5}$$

As S is normal, $\dim_{\mathrm{FS}}(S) = 1$ by Theorem 3.2.8. Hence for all $C \in \mathrm{ILFST}$ and almost every n we have

$$|C(S \upharpoonright n)| \geq (1 - \frac{\delta}{2})n. \tag{5.6}$$

Therefore for almost every n

$$|C(S \upharpoonright n)| - |\mathrm{LZ}(S \upharpoonright n)| \geq (1 - \frac{\delta}{2})n - (\varepsilon + \frac{\delta}{2})n = (1 - \varepsilon - \delta)n. \tag{5.7}$$

Hence as C was arbitrary, S is LZ-deep.

□

Corollary 5.4.2. *There exists a sequence which is LZ-deep but neither a.e. nor i.o. FS-deep.*

Proof. Let S be the normal LZ-deep sequence from Theorem 5.4.1. As S is normal, $\dim_{\mathrm{FS}}(S) = 1$. Hence S is not a.e. FS-deep by Theorem 3.3.5. This is similarly true for i.o. FS-depth.

□

Next we demonstrate that the sequence which satisfies Theorem 4.4.2 is an i.o. FS-deep sequence that is not LZ-deep. The long sections of random strings in S prevent LZ-depth as was the case with PD-depth.

Theorem 5.4.3. *There exists a sequence S which is i.o. FS-deep but not LZ-deep.*

Proof. Let $0 < \beta < 1$. Let S be the sequence satisfying Theorem 4.4.2. We claim this sequence satisfies the theorem. As shown in Theorem 4.4.2, i.o. FS-depth$(S) \geq \beta$, i.e. S is i.o. FS-deep.

All that remains to show is that S is not LZ-deep. Recall that S is broken into substrings $S = S_1 S_2 S_3 \ldots$ where $|S_1| = 2$ and for all j, $|S_j| = 2^{|S_1 \ldots S_{j-1}|}$, i.e. for $j > 1$, $\log |S_j| = |S_1 \ldots S_{j-1}|$. Recall also that for j odd, S_j is a string of maximal plain Kolmogorov complexity in the sense that $K(S_j) \geq |S_j|$. We assume j is always odd in the rest of the theorem. We write $\overline{S_j}$ to denote the prefix $S_1 \ldots S_j$.

For any prefix of the form $\overline{S_j}$, as LZ is lossless, $\overline{S_j}$ can be recovered from the string

$$d(\overline{S_{j-1}}) \cdot 01 \cdot \mathrm{LZ}(S_j | \overline{S_{j-1}}).$$

Therefore for j odd,

$$|S_j| \leq K(S_j) \leq 2|\overline{S_{j-1}}| + 2 + |\mathrm{LZ}(S_j | \overline{S_{j-1}})| + O(1), \tag{5.8}$$

and so for j large

$$\begin{aligned} |\mathrm{LZ}(S_j | \overline{S_{j-1}})| &\geq |S_j| - 2|\overline{S_{j-1}}| - O(1) \\ &= |S_j| - 2\log(|S_j|) - O(1) > |S_j|(1 - \frac{\beta}{2}). \end{aligned} \tag{5.9}$$

Therefore for infinitely many prefixes of the form $\overline{S_j}$, we have

$$\begin{aligned} |\mathrm{LZ}(\overline{S_j})| &\geq |\mathrm{LZ}(S_j | \overline{S_{j-1}})| > |S_j|(1 - \frac{\beta}{2}) && \text{(by (5.9))} \\ &= (|\overline{S_j}| - \log(|S_j|))(1 - \frac{\beta}{2}) \\ &> |\overline{S_j}|(1 - \frac{\beta}{2})(1 - \frac{\beta}{2}) > |\overline{S_j}|(1 - \beta). && (5.10) \end{aligned}$$

Hence for infinitely many prefixes of S we have that

$$|I_{\mathrm{FS}}(\overline{S_j})| - |\mathrm{LZ}(\overline{S_j})| < |\overline{S_j}| - |\overline{S_j}|(1-\beta) = |\overline{S_j}|\beta. \tag{5.11}$$

As β was arbitrary, it follows that S is not LZ-deep.

□

The following result follows from the previous theorem and shows that it is possible to build i.o. LZ-deep sequences which are not a.e. LZ-deep.

Lemma 5.4.4. *There exists a sequence S which is i.o. LZ-deep and i.o. FS-deep but not a.e. LZ-deep.*

Proof. Let S be the sequence satisfying Theorems 4.4.2 and 5.4.3 which is i.o. FS-deep but not a.e. LZ-deep. All that remains to show is that S is i.o. LZ-deep.

Recall to construct S, we split the non-negative integers into intervals $I_1, I_2, \ldots$ such that $|I_1| = 2$ and $|I_j| = 2^{|I_1|+\cdots|I_{j-1}|}$ for all $j > 1$. Also recall that for all $k \geq 1$, k was devoted to intervals I_j where j had the form $j = 2^k + t(2^{k+1})$, for $t \geq 0$. Recall also that S was built in stages $S = S_1 S_2 \ldots$ such that if k was devoted to interval I_j then $S_j = r_k^{|I_j|/|r_k|}$ where r_k was a string of length $|I_{2^k}|$ that was $3k$-finite-state random in the sense that

$$D_{\mathrm{FS}}^{3k}(r_k) \geq |r_k| - 4k. \tag{5.12}$$

We first examine how well any ILFST compresses prefixes of S. Let $C \in$ ILFST with states $Q_C = \{q_1, \ldots, q_p\}$. We assume all states of C are reachable from its start state. For all $1 \leq i \leq p$, we let C_i denote the FST with the same states, transitions and outputs as C but with start state q_i, i.e. for all $x \in \{0,1\}^*$, $C_i(x) = \nu_C(q_i, x)$. As C is an ILFST, so is C_i. Recall from our encodings of FSTs that therefore $C_i \in \mathrm{FST}^{\leq 3|C|}$, where $|C|$ is the length of the encoding for C.

Next let d be such that $I_{\mathrm{FS}} \in \mathrm{FST}^{\leq d}$. As C_i is an ILFST, by Lemma 3.3.6 there exists d' such that for all i and x

$$D^{d'}_{\mathrm{FS}}(C_i(x)) \leq D^{d}_{\mathrm{FS}}(x). \tag{5.13}$$

Let $0 < \varepsilon < 1$. By Lemma 3.3.6, there exists an l such that for all i and almost every x,

$$D^{l}_{\mathrm{FS}}(x) \leq (1 + \frac{\varepsilon}{3}) D^{d'}_{\mathrm{FS}}(C_i(x)) + O(1). \tag{5.14}$$

Of our set of random strings $\{r_k\}_{k\geq 1}$, let $l' \geq l$ be such that $r_{l'}$ satisfies both Equation (5.14) and $|r_{l'}| - 4l' \geq (1 - \varepsilon/3)|r_{l'}|$. Such an l' must exist as $\{r_k\}_{k\in\mathbb{N}}$ is a set of strings of increasing length.

Hence we have that for all i

$$\begin{aligned}
|r_{l'}|(1 - \frac{\varepsilon}{3}) \leq |r_{l'}| - 4l' \leq D^{3l'}_{\mathrm{FS}}(r_{l'}) \leq D^{l}_{\mathrm{FS}}(r_{l'}) \qquad & \text{(as } r_{l'} \text{ is } 3l'\text{-FS random)} \\
\leq (1 + \frac{\varepsilon}{3}) D^{d'}_{\mathrm{FS}}(C_i(r_{l'})) + O(1) \qquad & \text{(by (5.14))} \\
\leq D^{d'}_{\mathrm{FS}}(C_i(r_{l'})) + \frac{\varepsilon}{3} D^{d}_{\mathrm{FS}}(r_{l'}) + O(1) \qquad & \text{(by (5.13))} \\
\leq |C_i(r_{l'})| + \frac{\varepsilon}{3}|r_{l'}| + O(1). \qquad & (5.15)
\end{aligned}$$

Hence by Equation (5.15) for all i, when l' is chosen large

$$|C_i(r_{l'})| \geq |r_{l'}|(1 - \frac{\varepsilon}{3}) - \frac{\varepsilon}{3}|r_{l'}| - O(1) > |r_{l'}|(1 - \varepsilon). \tag{5.16}$$

That is, for all i

$$|C(q_i, r_{l'})| > |r_{l'}|(1 - \varepsilon). \tag{5.17}$$

We now calculate a lower bound of compression of C for prefixes of S of the

form $\overline{S_j}$ where j is devoted to $r_{l'}$. For almost every such j we have that

$$\begin{aligned}
|C(\overline{S_j})| &\geq |C(\overline{S_j})| - |C(\overline{S_{j-1}})| \\
&> \frac{|S_j|}{|r_{l'}|}(|r_{l'}|(1-\varepsilon)) && \text{(by (5.17))} \\
&= |S_j|(1-\varepsilon) = (|\overline{S_j}| - |\overline{S_{j-1}}|)(1-\varepsilon) && (5.18) \\
&= (|\overline{S_j}| - \log(|S_j|))(1-\varepsilon) \\
&> |\overline{S_j}|(1-\beta) && (5.19)
\end{aligned}$$

where $\varepsilon < \beta < 1$.

We next examine how well LZ compresses any prefix $\overline{S_j}$ of S where j is devoted to l'. Note after reading $\overline{S_{j-1}}$, LZ's dictionary will contain at most $|\overline{S_{j-1}}|$ entries, i.e. it has size bounded above by $\log(|S_j|)$. Setting $a_{l'} = |r_{l'}| + 1$, by Lemma 5.2.1 we have that

$$\begin{aligned}
|\mathrm{LZ}(S_j|\overline{S_{j-1}})| &\leq \sqrt{2a_{l'}|S_j|}\log(|\overline{S_{j-1}}| + \sqrt{2a_{l'}|S_j|} \\
&= \sqrt{2a_{l'}|S_j|}\log(\log|S_j| + \sqrt{2a_{l'}|S_j|}) \\
&= O(\sqrt{|S_j|}\log|S_j|). && (5.20)
\end{aligned}$$

Hence we have that for j large devoted to l' that

$$\begin{aligned}
|\mathrm{LZ}(\overline{S_j})| &= |\mathrm{LZ}(\overline{S_{j-1}})| + |\mathrm{LZ}(S_j|\overline{S_{j-1}})| \\
&\leq |\overline{S_{j-1}}| + o(|\overline{S_{j-1}}|) + O(\sqrt{|S_j|}\log|S_j|) && \text{(by (5.20))} \\
&= \log(|S_j|) + o(\log(|S_j|)) + O(\sqrt{|S_j|}\log|S_j|) \\
&= O(\sqrt{|S_j|}\log|S_j|). && (5.21)
\end{aligned}$$

Hence, as infinitely many intervals are devoted to l', for infinitely many pre-

fixes we have

$$|C(\overline{S_j})| - |\mathrm{LZ}(\overline{S_j})| \geq |\overline{S_j}|(1-\beta) - O(\sqrt{|S_j|}\log|S_j|) \quad \text{(by (5.19) and (5.21))}$$
$$> |\overline{S_j}|(1-\alpha) \tag{5.22}$$

where $\beta < \alpha < 1$. Hence as C was an arbitrary ILFST, it holds that S is i.o. LZ-deep.

□

5.5 Separation from Pushdown Depth

The following results demonstrate the difference between LZ-depth with PD-depth. We first show that the sequence S from Theorem 1 of [109] is an example of a LZ-deep sequence which is not PD-deep. S contains repeated non-consecutive random substrings which Lempel-Ziv can exploit to compress. However, the random strings are long enough that a pushdown compressor cannot compress the sequence.

Theorem 5.5.1. *There exists a sequence S which is not PD-deep but is LZ-deep.*

Proof. Let S be the sequence from Theorem 1 of [109] which satisfies $R_{\mathrm{LZ}}(S) = 0$ but $\rho_{\mathrm{PD}}(S) = 1$. We claim S satisfies the theorem statement. We omit a description of the construction of S in this proof. As $\rho_{\mathrm{PD}}(S) = 1$, S is not PD-deep by Theorem 4.3.2.

Let $0 < \alpha < 1$. As $\rho_{\mathrm{PD}}(S) = 1$, it also holds that $\dim_{\mathrm{FS}}(S) = 1$. Hence for all $\widehat{C} \in$ ILFST it holds that

$$|\widehat{C}(S \upharpoonright n)| > (1 - \frac{\alpha}{2})n. \tag{5.23}$$

As $R_{\mathrm{LZ}}(S) = 0$, it holds that for almost all n that

$$|\mathrm{LZ}(S \upharpoonright n)| \leq \frac{\alpha}{2} n. \tag{5.24}$$

Hence for almost all n we have that

$$|\widehat{C}(S \upharpoonright n)| - |\mathrm{LZ}(S \upharpoonright n)| \geq (1 - \frac{\alpha}{2})n - \frac{\alpha}{2}n = (1 - \alpha)n. \tag{5.25}$$

As $\widehat{C}$ was arbitrary, it holds that S is LZ-deep.

□

Next we demonstrate the existence of a sequence that roughly has a PD-depth level of 1/2 while having a very small LZ-depth level. This sequence was first presented in Theorem 5 of [109] and was built by enumerating strings in such a way so that a pushdown compressor can use its stack to compress, but an ILUPDC cannot use their stacks due to their unary nature. LZ cannot compress the sequence either as it is similar to a Champernowne sequence. LZ performs poorly on such sequences as discussed briefly previously.

Theorem 5.5.2. *For all $0 < \beta < 1/2$, there exists a sequence S such that PD-depth$(S) \geq (1/2 - \beta)$ but LZ-depth$(S) < \beta$.*

Proof. Let $0 < \beta < 1/2$. We claim that the sequence from Theorem 5 of [109] satisfies the theorem statement. We first give a brief description of this sequence. Let ε be such that $0 < \varepsilon < \beta$, and let k and v be non-negative integers to be determined later.

For any $n \in \mathbb{N}$, let T_n denote the set of strings of length n that do not contain the substring 1^j in x for all $j \geq k$. As T_n contains the set of strings whose every k^{th} bit is 0, it follows that $|T_n| \geq 2^{(\frac{k-1}{k})n}$. Note that for every $x \in T_n$, there exists

$y \in T_{n-1}$ and $b \in \{0,1\}$ such that $x = yb$. Hence

$$|T_n| < 2|T_{n-1}|. \tag{5.26}$$

Let $A_n = \{a_{1_n}, \ldots, a_{u_n}\}$ be the set of palindromes in T_n. As fixing the first $\lceil \frac{n}{2} \rceil$ bits determines a palindrome, $|A_n| \leq 2^{\lceil \frac{n}{2} \rceil}$. The remaining strings in $T_n - A_n$ are split into $v+1$ pairs of sets $X_{n,i} = \{x_{n,i,1}, \ldots, x_{n,i,t_n^i}\}$ and $Y_{n,i} = \{y_{n,i,1}, \ldots, y_{n,i,t_n^i}\}$ where $t_n^i = \lfloor \frac{|T_n - A_n|}{2v} \rfloor$ if $i \neq v+1$ and

$$t_n^{v+1} = \frac{1}{2}(|T_n - A_n| - 2\sum_{i=1}^{v} |X_{n,i}|),$$

$(x_{n,i,j})^{-1} = y_{n,i,j}$ for every $1 \leq j \leq t_n^i$ and $1 \leq i \leq v+1$ both $x_{n,i,1}$ and y_{n,i,t_n} start with 0 (excluding the case where both $X_{n,v+1}$ and $Y_{n,v+1}$ are the empty sets). Note that for convenience we write X_i, Y_i for $X_{n,i}, Y_{n,i}$ respectively.

The sequence S which satisfies the theorem is constructed in stages as follows: Let $f(k) = 2k$ and $f(n+1) = f(n) + v + 2$. Note that $n < f(n) < n^2$ for large n. For $n \leq k-1$, S_n is a concatenation of all strings of length n, i.e. $S_n = 0^n \cdot 0^{n-1}1 \cdots 1^{n-1}0 \cdot 1^n$. For $n \geq k$,

$$S_n = a_{1_n} \ldots a_{u_n} 1^{f(n)} z_{n,1} z_{n,2} \ldots z_{n,v} z_{n,v+1}$$

where

$$z_{n,i} = x_{n,i,1} x_{n,i,2} \ldots x_{n,i,t_n^i - 1} x_{n,i,t_n^i} 1^{f(n)+i} y_{n,i,t_n^i} y_{n,i,t_n^i - 1} \ldots y_{n,i,2} y_{n,i,1},$$

with the possibility that $z_{n,v+1} = 1^{f(n)+v+1}$ only. That is, S_n is a concatenation of all strings in A_n followed by a flag of $f(n)$ ones, followed by a concatenation of all strings in the X_i zones and Y_i zones separated by flags of increasing length

such that each Y_i zone is the X_i zone written in reverse. Let

$$S = S_1 S_2 \dots S_{k-1} 1^k 1^{k+1} \dots 1^{2k-1} S_k S_{k+1} \dots$$

i.e. the concatenation of all S_j zones with some extra flags between S_{k-1} and S_k.

Then from [109], for ε small, choosing k and v appropriately large we have that

$$\rho_{LZ}(S) \geq 1 - \varepsilon, \text{ and } R_{\mathrm{PD}}(S) \leq 1/2. \tag{5.27}$$

Next we consider how any $C \in$ ILUPDC performs on S. Let $\overline{S_j}$ denote the prefix

$$S_1 \dots S_{k-1} 1^k \dots 1^{2k-1} S_k \dots S_j$$

of S for all $j \geq k$.

Suppose C is reading zone S_n and can perform at most c λ-transitions in a row. We examine the proportion of strings in T_n that give a large contribution to the output. For simplicity, we write $C(p, x, s) = (q, v)$ to represent that when C is in state p with stack contents $s = 0^a z_0$ for some $a \geq 0$, on input x C outputs v and ends in state q, i.e. $C(p, x, s) = (\widehat{\delta_Q}(p, x, s), \widehat{\nu}(p, x, s))$.

For each $x \in T_n$, let

$$h_x = \min\{|v| : \exists p, q \in Q, \exists s \in \{0^a z_0 : a \geq 0\}, C(p, x, s) = (q, v)\}$$

be the minimum possible addition of the output that could result from reading x. We restrict ourselves to reachable combinations of pairs of states and choices for s. Let

$$B_n = \{x \in T_n : h_x \geq \frac{(k-2)n}{k}\}$$

be the set of strings that give a large contribution to the output.

Next consider $x' \in T_n - B_n$. There is a computation of x' that results in C outputting at most $\frac{(k-2)n}{k}$ bits. As C is lossless, x' can be associated uniquely to a start state $p_{x'}$, stack contents $s_{x'}$, end state $q_{x'}$ and output $v_{x'}$ where $|v_{x'}| < \frac{(k-2)n}{k}$ such that $C(p_{x'}, x', s_{x'}) = (q_{x'}, v_{x'})$. Recall from our previous discussion that on reading x of length n, if C has a stack of height bigger than $(c+1)n$, the stack will have no impact on the compression of x. That is, if $k \neq k'$ but $k, k' \geq (c+1)n$, then $C(p_{x'}, x', 0^k z_0) = C(p_{x'}, x', 0^{k'} z_0)$.

Hence we can build a map g such that $g(x') = (p_{x'}, s'_{x'}, v_{x'}, q_{x'})$ where $0 \leq s'_{x'} < (c+1)|x|$. As this map g is injective, we can bound $|T_n - B_n|$ as follows:

$$\begin{aligned} |T_n - B_n| &\leq |Q|^2 \cdot (c+1)n \cdot 2^{<\frac{(k-2)n}{k}} \\ &< |Q|^2 \cdot (c+1)n \cdot 2^{\frac{(k-2)n}{k}}. \end{aligned} \tag{5.28}$$

For $0 < \delta < 1/6$ whose value is determined later, as $|T_n| \geq 2^{\frac{(k-1)n}{k}}$, we have that for n large

$$\begin{aligned} |B_n| &= |T_n| - |T_n - B_n| \\ &> |T_n| - |Q|^2 \cdot (c+1)n \cdot 2^{\frac{(k-2)n}{k}} && \text{(by (5.28))} \\ &> |T_n|(1-\delta). && (5.29) \end{aligned}$$

Similarly, as the flags are only comprised of $O(n^2)$ bits in each S_n zone, we have for n large that

$$|T_n|n > |S_n|(1-\delta). \tag{5.30}$$

Then for n large (say for all $n \geq i$ such that (5.29) and (5.30) hold),

$$
\begin{aligned}
|C(\overline{S_n})| &> \frac{k-2}{k}\sum_{j=i}^{m} j|B_j| \\
&> \frac{k-2}{k}(1-\delta)\sum_{j=i}^{n} j|T_j| && \text{(by (5.29))} \\
&> \frac{k-2}{k}(1-2\delta)\sum_{j=i}^{n} |S_j| && \text{(by (5.30))} \\
&= \frac{k-2}{k}(1-2\delta)(|\overline{S_n}| - |\overline{S_{i-1}}|) \\
&> \frac{k-2}{k}(1-3\delta)|\overline{S_n}|. && (5.31)
\end{aligned}
$$

The compression ratio of S on $C \in \text{ILUPDC}$ is least on prefixes of the form $\overline{S_n}\cdot x_{n+1}$, where potentially x_{n+1} is a concatenation of all the strings in $T_{n+1} - B_{n+1}$, i.e. the compressible strings of T_{n+1}. Let x_{n+1} be a such a potential prefix of S_{n+1}. Then if $F_{n+1} = \sum_{i=0}^{v}(f(n+1)+i)$, the length of the flags in S_{n+1}, we can bound the length of $|x_{n+1}|$ as follows. For n large we have

$$
\begin{aligned}
|x_{n+1}| &< |T_{n+1} - B_{n+1}|(n+1) + F_{n+1} \\
&< (|T_{n+1}| - |B_{n+1}|)(n+1) + (v+2)n^2 \\
&< \delta|T_{n+1}|(n+1) + \delta|T_n|(n+1) && \text{(by (5.29))} \\
&< 2\delta|T_n|(n+1) + \delta|T_n|(n+1) && \text{(by (5.26))} \\
&= 3\delta|T_n|(n+1) \\
&< 3\delta|S_1 \ldots S_n|. && (5.32)
\end{aligned}
$$

Hence for n large

$$\begin{aligned}
|C(\overline{S_n}x_{n+1})| &\geq (\frac{k-2}{k})(1-3\delta)(|\overline{S_n}x_{n+1}| - |x_{n+1}|) \\
&> (\frac{k-2}{k})(1-3\delta)(|\overline{S_n}x_{n+1}| - 3\delta|\overline{S_n}|) \qquad \text{(by (5.31))} \\
&> (\frac{k-2}{k})(1-6\delta)|\overline{S_n}x_{n+1}| \\
&> \frac{k-3}{k}|\overline{S_n}x_{n+1}| \qquad (5.33)
\end{aligned}$$

when δ is chosen sufficiently small, i.e. $\delta < \frac{1}{6(k-2)}$. Hence

$$\rho_{\mathrm{UPD}}(S) \geq \frac{k-3}{k}. \qquad (5.34)$$

Thus for all $0 < \varepsilon' < \frac{k-3}{k}$, for almost every n,

$$|C(S \upharpoonright n)| \geq (\frac{k-3}{k} - \varepsilon')n.$$

Next let $\hat{C} \in \mathrm{ILPDC}$ be such that $\hat{C}$ achieves $R_{\mathrm{PD}}(S) \leq 1/2$. Then for all $\varepsilon' > 0$ and almost every n it holds that

$$|C(S \upharpoonright n)| \leq (\frac{1}{2} + \varepsilon').$$

Hence, choosing $\varepsilon' = 1/2k$, for almost every n and every $C \in \mathrm{ILUPDC}$ we have

$$\begin{aligned}
|C(S \upharpoonright n)| - |\hat{C}(S \upharpoonright n)| &\geq (\frac{k-3}{k} - \varepsilon')n - (\frac{1}{2} + \varepsilon')n \\
&= (\frac{1}{2} - \frac{3}{k} - \frac{1}{k})n = (\frac{1}{2} - \frac{4}{k})n. \qquad (5.35)
\end{aligned}$$

That is, PD-depth$(S) > \frac{1}{2} - \frac{4}{k}$. Hence, choosing k large appropriately at the start such that $\frac{4}{k} < \beta$ we have that PD-depth$(S) > \frac{1}{2} - \beta$.

Next we examine LZ-depth. Recall $\rho_{LZ}(S) \geq 1 - \varepsilon$. Thus for c such that

$\varepsilon + c < \beta$ (recall $\varepsilon < \beta$), for almost every n it holds that

$$|LZ(S \upharpoonright n)| > (1 - \varepsilon - c)n. \tag{5.36}$$

Hence as $I_{\mathrm{FS}} \in \mathrm{ILFST}$, we have that for almost every n

$$|I_{\mathrm{FS}}(S \upharpoonright n)| - |LZ(S \upharpoonright n)| < n - (1 - \varepsilon - c)n = (\varepsilon + c)n < \beta n. \tag{5.37}$$

Hence we have that LZ-depth$(S) < \beta$.

In conclusion, for all $0 < \beta < \frac{1}{2}$, choosing ε such that $\varepsilon < \beta$ and k such that $\frac{4}{k} < \beta$, a sequence S can be built which satisfies the requirements of the theorem.

□

5.6 Summary

In this chapter we introduced a new notion of depth based on the LZ78 compression algorithm which examined the difference between ILFSTs and the LZ78 compression algorithm.

We showed LZ-depth satisfied some of the fundamental properties of depth in Theorem 5.3.3, i.e. FST-trivial and LZ-incompressible sequences were not deep. The question of a slow growth type law remains open. We demonstrated that LZ-depth differs from both a.e. and i.o. FS-depth in Theorem 5.4.1 by showing the existence of a LZ-deep normal sequence. We also separated LZ-depth from PD-depth by first demonstrating the existence of a sequence which is LZ-deep but not PD-deep in Theorem 5.5.1. We continued by showing the existence of a sequence which is PD-deep, and if it is LZ-deep, it must have a low depth level as given in Theorem 5.5.2.

We will revisit LZ-depth in Chapter 6.

Chapter 6

Pebble Depth

6.1 Introduction

In Chapters 3, 4 and 5 we examined depth notions where the transducers and compression algorithms we used were limited in the following sense: For FS-depth where complexity was based on the minimal input needed to output a certain string, finite-state transducers are limited in that that relationship between the output and input is linear. For PD-depth, pushdown transducers with a bounded number of λ-transitions between the reading of each bit are similarly linear. Furthermore finite-state, pushdown transducers and the Lempel-Ziv 78 algorithm are limited to reading their input in one direction. They are unable to backtrack to reread a previously seen part of their input.

In this chapter we examine a notion of depth based on transducers which do not suffer from the above restrictions which we call *pebble depth (PB-depth)*. Specifically we examine the minimal descriptional complexity of a class of two-way transducers known as *pebble transducers* which have a polynomial size output compared to their input [65].

For $k \in \mathbb{N}$, a *k-pebble automaton* is a two-way finite-state automaton with the additional capacity to mark k squares of its tape with its *pebbles*. In particular, a

0-pebble automaton is the same as the classical understanding of a two-way finite-state automaton[1]. Both 0-pebble automata [121, 133] and 1-pebble automata [22] are known to be equivalent to one-way finite-state automata, i.e. they decide exactly the regular languages.

When two or more pebbles are used, we henceforth will exclusively examine pebble automata whose pebbles follow a stack-like discipline. By this we mean that the pebbles are ordered and a pebble's position on the tape can only be altered (lifted from or dropped onto a square of the input tape) if all lower ranked pebbles are currently on the tape and all higher ranked pebbles are not currently on the tape. All papers cited with regards to pebble automata and transducers, unless stated otherwise, also use this restriction. This restricted class was examined as acceptors by Globerman and Harel who also showed that these restricted pebble automata also only decide the regular languages [74]. Transforming a k-pebble automaton into a two-way finite-state automaton, and the blow-up in terms by the number of states, has been studied by Geffert and Ištoňová in [72].

Another variation of pebble automata (explored on trees) can be seen in [64] where the automata's pebbles are split into two groups of *visible* and *invisible* pebbles. Visible pebbles act as pebbles of a normal pebble automaton in that a transducer can see all the visible pebbles currently on the tape square it is reading. For invisible pebbles on the other hand, the automaton is only able to see the topmost invisible pebble on the current square it is reading. That is, on any tape square, the pebble automaton can see at most one invisible pebble while it can see all visible pebbles. This variation of pebble automata is not explored in this chapter.

Pebble automata were first examined as transducers for trees by Milo et al.

[1]It should be noted that some authors refer to the head of the automaton as a pebble and so a classical two-way finite-state automaton is considered a 1-pebble automaton to those.

in [110]. These tree transducers were later restricted to monadic trees, i.e. for strings, by Engelfriet and Maneth in [65]. While Globerman and Harel's result show that as acceptors, pebble automata are equivalent to finite-state automata [74], pebble transducers are much more powerful in the following sense: While two-way finite-state transducers have output of size $O(n)$ on inputs of size n, a k-pebble transducer has output of size $O(n^{k+1})$. As such, the class of functions computed by pebble transducers has been referred to as the class of *polyregular* functions and have been shown to have several equivalent characterizations [23, 24]. With regards to this class of polyregular functions, a recent result by Lhote demonstrates that a polyregular function has output of size $O(n^{k+1})$ if and only if the function can be performed by a k-pebble transducer [102].

Other variations of k-pebble transducers include *k-marble transducers* [60], where if pebble i is placed on the tape, the reading head of the transducer cannot move right past the square containing pebble i until it is lifted, and pebble $i+1$ can only be placed on a square to the left of the square containing pebble i. Another variation are the *comparison-free k-pebble transducers* [117], also known as *k-blind transducers* [59], which are similar to, but further restrict the idea of invisible pebbles mentioned previously from [64]. Recently, Douéneau-Tabot has compared the expressive power of ordinary k-pebble, k-marble and k-blind transducers [59] that produce unary outputs. Neither k-marble nor k-blind transducers are studied in this chapter.

The extra power pebble transducers have over finite-state transducers are their ability to compute the following three basic functions:

1. Due to their two-way nature, pebble transducers can compute the function $\mathrm{rev}(x) = x^{-1}$ which prints the reverse of the input string.

2. A k-pebble transducer is able to compute the function $\mathrm{pow}_k(x) = x^{|x|^k}$ which prints x $|x|^k$ times resulting in a string of length $|x|^{k+1}$. It achieves

this by moving its head left to right printing x and using its k pebbles to count to $|x|^k$ by moving one pebble to the right for each printing of x performed.

3. A 1-pebble transducer is able to compute the function pref(x) which prints every prefix of x in order of length. That is, pref(x) $= x_0x_0x_1x_0x_1x_2 \dots x$, where $x = x_0 \dots x_n$. It achieves this by scanning over the input tape and moving its pebble one square to the right to keep track of each prefix printed.

We define our notion of pebble depth (PB-depth) based on the difference between the minimal descriptional complexity of finite-state transducers and pebble transducers. We demonstrate how our notion of PB-depth satisfies some of the basic properties of depth, i.e. FST-trivial sequences and PB-incompressible sequences are not deep and that a slow growth law holds. We furthermore compare PB-depth with the other depth notions previously examined. We first show that unlike FS-depth (both a.e. and i.o.) where normal sequences are not deep, there exists normal PB-deep sequences. We demonstrate a difference with LZ-depth by showing the existence of a sequence which has a PB-depth level of approximately 1/2 and, if it is LZ-deep, has a low level of LZ-depth. This sequence is an example of a non-normal pebble deep sequence. We furthermore show that there exists sequences which are both PB-deep and PD-deep.

6.2 Pebble Transducers

A k-pebble transducer is two-way finite-state transducer which also has k-pebbles labelled $1, \dots, k$. Initially the transducer has no pebbles on its input tape, however during its computation the transducer can drop pebbles onto and pick up pebbles from squares of the tape. At each stage of computation the transducer knows which pebbles are on the square under its head and it can choose to drop

a new pebble on that square, lift the topmost pebble from that square, or move to a different square. However, the way in which the transducer can drop and lift pebbles is restricted to act in a stack like fashion.

We use the pebble transducers with this restriction as these transducers have very nice properties. For instance, in Theorem 2 of [65] it was shown that this class of deterministic pebble transducers is closed under composition. We use this property in the proof of Theorem 6.3.6. These transducers also have other nice properties such as that all non-deterministic pebble transducers which compute a partial function can in fact be computed by a deterministic pebble transducer [63].

Definition 6.2.1. A *pebble transducer (PB)* is a 6-tuple $T = (Q, q_0, F, k, \delta, \nu)$ where

1. Q is a non-empty, finite set of *states*,
2. $q_0 \in Q$ is the *start state*,
3. $F \subseteq Q$ is the set of *final states*,
4. k is the number of *pebbles* allowed to be placed on the tape,
5. $\delta : (Q-F)\times\{0,1,\dashv,\vdash\}\times\{0,1\}^k \to Q\times\{+1,-1,\text{push},\text{pop}\}$ is the *transition function*,
6. $\nu : (Q-F)\times\{0,1,\dashv,\vdash\}\times\{0,1\}^k \to \{0,1\}^*$ is the *output function*.

A PB with k pebbles is referred to as a k-pebble transducer. On input $x \in \{0,1\}^*$, the input tape contains $\dashv x \vdash$, where $\dashv$ and $\vdash$ are the left and right end markers of the tape respectively. The tape squares are numbered $0, 1, \ldots, |x|, |x|+1$. A *configuration* of T is a 4-tuple (q, i, σ, w) where $q \in Q$ is the current state of T, $0 \leq i \leq |x|+1$ is the current position of the head, $\sigma \in \{\perp, 0, \ldots, |x|+1\}^k$ is a tuple indicating the location of the pebbles and $w \in \{0,1\}^*$ is what T has

outputted so far. That is, $\sigma[m-1] = j$ means that pebble m is on square j, and $\sigma[m-1] = \perp$ means pebble m is not currently on the tape. Hence by the stack nature of T, if only l pebbles are currently placed on the tape then $\sigma[l] = \cdots = \sigma[k-1] = \perp$.

If T is in configuration (q, i, σ, w) with a being the symbol on square i of the tape, then T's transition and output functions δ and ν take the input (q, a, b) where for $0 \leq j \leq k-1$, $b[j] = 1 \iff \sigma[j] = i$.

If there are l pebbles on T's tape, $\delta(q, a, b) = (q', d)$ means that T moves from state q to state q' and performs action d, where $+1$ means move one square right, -1 means move one square left, push means place pebble $l+1$ onto the current square and pop means remove pebble l from the current square. These four types transitions are undefined if performing d results in an impossible action, i.e. if $d = +1$ when $a = \vdash$, $d = -1$ when $a = \dashv$, $d =$ push when all pebbles are currently on the tape, and $d =$ pop when pebble l is not on the current square of the configuration respectively. Following a transition, T enters the *successor* configuration $(q', i', \sigma', w \cdot \nu(q, a, b))$, where q', i' and σ' reflect the new state, head position and place of the pebbles based on the result of $\delta(q, a, b)$. Specifically $q' = \delta_Q(q, a, b)$, $i' \in \{i-1, i, 1+1\}$ depending on whether the instruction was to move left, push or pop, or move right respectively and

$$\sigma' = \begin{cases} \sigma & \text{if the instruction was } +1 \text{ or } -1 \\ \sigma[0..l-1] i \perp^{k-l-1} & \text{if the instruction was to push pebble } l+1 \\ \sigma[0..l-2] \perp^{k-l+1} & \text{if the instruction was to pop pebble } l. \end{cases} \tag{6.1}$$

We say that T on input x outputs w, i.e. $T(x) = w$, if starting in configuration $(q_0, 0, \perp^k, \lambda)$, there is a finite sequence of successor configurations of T ending with (q, i, σ, w), where $q \in F$. We require the use of final states to define the output as we do not wish to consider cases where T finds itself in a loop and outputs an

infinite sequence, i.e. $T(x) = zy^\omega$, for some $z, y \in \{0,1\}^*$ where $|y| \geq 1$. We write PB to denote the set of deterministic pebble transducers. We use $\text{PB}_k \subset \text{PB}$ to denote set of deterministic k-pebble transducers. Note then that each $M \in \text{PB}$ computes a partial function from $\{0,1\}^*$ to $\{0,1\}^*$.

Using constructions of [72], Engelfriet showed that the set of functions computed by PBs is closed under composition [63]. This property is used in the proof of a slow growth law in Theorem 6.3.6.

Theorem 6.2.2 ([63])**.** *Let $r, m \geq 0$. Let $R \in PB_r$ and $M \in PB_m$. Then there exists $T \in PB_{rm+r+m}$ such that for all $x \in \{0,1\}^*$, $T(x) = R(M(x))$.*

6.2.1 k-Pebble Complexity

For a class of transducers F, recall in Section 2.4 of Chapter 2 we discussed the k-F complexity of strings and binary representations of F transducers. In this chapter we examine these notions with respect to the class of pebble transducers, i.e. when F = PB. As such, the size of PBs, the sets $\text{PB}^{\leq k}$, and the k-pebble complexity $D^k_{\text{PB}}(x)$ of a string x are all defined as in Chapter 2. We note that the set $\text{PB}^{\leq k}$, the set of pebble transducers with a binary representation of length k, should not be confused with the set of k pebble transducers PB_k. Unlike in Chapter 3 where we gave a specific binary representation of FSTs, we do not give a binary representation of PBs here as our proofs do not depend on a specific representation. The proof of Theorem 3.3.4 can be adapted to hold for PBs too, i.e. pebble depth does not depend on the binary representation chosen.

Before we discuss pebble depth, we explore an analogous result to Lemma 3.3.6 (part 1) for k-PB complexity. It states that if given a description y for a string x, then y is also a description for $T(x)$ where $T \in \text{PB}$.

Lemma 6.2.3. *Let $T \in \mathrm{PB}$. Then*

$$(\forall k \in \mathbb{N})(\exists k' \in \mathbb{N})(\forall x \in \{0,1\}^*)\, D^{k'}_{\mathrm{PB}}(T(x)) \leq D^{k}_{\mathrm{PB}}(x).$$

Proof. Let x, k and t be as stated. Let p be a k-minimal program for x, i.e. $A(p) = x$ where $A \in \mathrm{PB}^{\leq k}$ and $\mathrm{PB}^{\leq k}(x) = |p|$.

Suppose A has a pebbles and T has t pebbles. Then by Theorem 6.2.2 there exists $B \in \mathrm{PB}_{ta+a+n}$ such that for all $y \in \{0,1\}^*$, $B(y) = T(A(y))$. In particular $B(p)$ outputs $T(x)$ and hence p is also a description of $T(x)$. Therefore setting $k' = |B|$ we have that $D^{k'}_{\mathrm{PB}}(T(x)) \leq D^{k}_{\mathrm{PB}}(x)$.

□

In particular, the above lemma says that if we have a description y for the string x, y is also a description of $x^{|x|^n}$ for all n, x^{-1}, and $\mathrm{pref}(x)$.

Corollary 6.2.4. *Let $k, n \in \mathbb{N}$. Let $x, y \in \{0,1\}^*$ such that $D^{k}_{\mathrm{PB}}(x) = |y|$. Then there exists $k' \in \mathbb{N}$ such that $D^{k'}_{\mathrm{PB}}(x^{|x|^n}) \leq |y|$.*

Proof. Let n be as stated in the lemma. Note that there exists a pebble transducer $T \in \mathrm{PB}_n$ such that for all $z \in \{0,1\}^*$, $T(z) = z^{|z|^n}$. The result then follows from Lemma 6.2.3.

□

Corollary 6.2.5. *Let $k \in \mathbb{N}$. Let $x, y \in \{0,1\}^*$ such that $D^{k}_{\mathrm{PB}}(x) = |y|$. Then there exists $k' \in \mathbb{N}$ such that $D^{k'}_{\mathrm{PB}}(x^{-1}) \leq |y|$.*

Proof. Note that there exists $T \in \mathrm{PB}_0$ such that for all $z \in \{0,1\}^*$, $T(z) = z^{-1}$. The result then follows from Lemma 6.2.3.

□

Corollary 6.2.6. *Let $k \in \mathbb{N}$. Let $x, y \in \{0,1\}^*$ such that $D^{k}_{\mathrm{PB}}(x) = |y|$. Then there exists $k' \in \mathbb{N}$ such that $D^{k'}_{\mathrm{PB}}(\mathit{pref}(x)) \leq |y|$.*

Proof. Note that there exists $T \in \mathrm{PB}_1$ such that for all $z \in \{0,1\}^*$, $T(z) = \mathrm{pref}(z)$. The result then follows from Lemma 6.2.3.

□

The following lemma is analogous to Lemma 3.3.6 (part 2). It states that if y is a description for $M(x)$ where $M \in \mathrm{ILFST}$, then y is also a description of x.

Lemma 6.2.7. *Let $M \in \mathrm{ILFST}$. Then*

$$(\forall k \in \mathbb{N})(\exists k' \in \mathbb{N})(\forall x \in \mathbb{N})\, D^{k'}_{\mathrm{PB}}(x) \leq D^{k}_{\mathrm{PB}}(M(x)).$$

Proof. Let M, k and x be as stated in the lemma. By Theorem 3.2.4, there exists an ILFST M^{-1} and constant b such that for all $z \in \{0,1\}^*$, $z \upharpoonright |z| - b \sqsubseteq M^{-1}(M(x)) \sqsubseteq z$.

Note that both M and M^{-1} can be simulated by 0-pebble transducers that print nothing on first reading $\dashv$, then read their input bit by bit moving right performing the same actions as M and M^{-1} respectively and where upon reading $\vdash$, they output nothing and enter their final state. For simplicity of notation we call these equivalent 0-pebble transducers M and M^{-1} also.

Let p be a k-PB minimal program for $M(x)$, i.e. $A(p) = M(x)$ for $A \in \mathrm{PB}^{\leq k}$, and $D^{k}_{\mathrm{PB}}(M(x)) = |p|$. We construct A' and p' for x. Let $y = M^{-1}(M(x))$, i.e. there exists some $z \in \{0,1\}^{\leq b}$ such that $yz = x$. Let A' be the PB which on input p simulates $A(p)$ to get $M(x)$, and sticks the output into M^{-1} and adds z at the end of M^{-1}'s output, i.e. when it enters the final state of M^{-1}, regardless of what is under its reading head, it prints z and enters its own final state. Note that by Theorem 6.2.2, A' will have the same number of pebbles as A. Thus $D^{|A'|}_{\mathrm{PB}}(x) \leq D^{k}_{\mathrm{PB}}(M(x))$. As $|A'|$ depends only on k, the size of M and $|z| \leq b$, we set k' to be the smallest integer that takes all the possibilities for z into account. That is $D^{k'}_{\mathrm{PB}}(x) \leq D^{k}_{\mathrm{PB}}(M(x))$.

□

The following theorem is an analogous result to Lemma 3.3.12 which states that if p is a pebble description of x and q is a pebble description of y, a padded version of p followed by a flag, followed by q is a pebble description of the string xy.

Lemma 6.2.8. $(\forall \varepsilon > 0)(\forall k \in \mathbb{N})(\exists k' \in \mathbb{N})(\forall^{\infty} x \in \{0,1\}^*)(\forall y \in \{0,1\}^*)$

$$D^{k'}_{\mathrm{PB}}(xy) \leq (1+\varepsilon) D^{k}_{\mathrm{PB}}(x) + D^{k}_{\mathrm{PB}}(y) + 2.$$

Proof. The proof is similar to Lemma 3.3.12's proof.

Let ε, x, y and k be as stated in the lemma. Consider $p, q \in \{0,1\}^*$ such that $D^{k}_{\mathrm{PB}}(x) = |p|$ and $D^{k}_{\mathrm{PB}}(y) = |q|$. Let $A, B \in \mathrm{PB}^{\leq k}$ where $A(p) = x$ and $B(q) = y$.

Let $b = \lceil \frac{2}{\varepsilon} \rceil$. Then there exists integers n and r such that $|p| = nb + r$, where $0 \leq r < b$. Let p' be a new string such that p' begins with the first nb bits of p, with a 0 placed to separate every b bits starting at the beginning of the string. This is followed by a 1 and the remaining r bits of p doubled. So

$$p' = 0p_1 \dots p_b 0 p_{b+1} \dots p_{2b} 0 \dots p_{nb} 1 p_{nb+1} p_{nb+1} \dots p_{nb+r} p_{nb+r},$$

and

$$|p'| = n(b+1) + 2r + 1 = |p| + n + r + 1 \leq |p| + n + b + 1.$$

Then by the same argument as in Lemma 3.3.6, whenever $|p|$ is large enough we can arrive to the same result as in Equation 3.17, i.e. it holds that $|p'| \leq |p|(1+\varepsilon)$.

Suppose A has i pebbles and B has j pebbles. Let T be a pebble-transducer with $m = \max\{i, j\} + 1$ pebbles such that on input $p'01q$: T first examines the section of its tape containing $\dashv p'01$ and uses i of its pebbles to simulate $A(p)$. To do this, T treats the 01 following p' as if it was $\vdash$ on A's tape. Also, to ensure

T does not use more than i pebbles, T keeps track in its states of how many pebbles it has placed on the $\dashv p'01$ part of the tape. When T enters a final state of A, T knows it has outputted x. Upon this, T scans $\dashv p'01$ moving back and forth popping off all of the pebbles from its tape and moves its head right until it reaches the end of $p'01$ when no pebbles remain. On the final 1, T drops a pebble (henceforth denoted by $1^\bullet$) and enters the initial state of B. T then exclusively examines the section of tape containing $1^\bullet q \vdash$ to simulate $B(q)$. To do this, T treats $1^\bullet$ as $\dashv$ on B's tape. To ensure T does not use more than j pebbles, T keeps track in its states of how many pebbles it has placed on the $1^\bullet q \vdash$ part of the tape, excluding the pebble already dropped on $1^\bullet$. When T enters a final state of B, T enters its own final state having just outputted y.

Thus for $k' = |T|$, whenever $|p|$ is large enough we have that,

$$D^{k'}_{\mathrm{PB}}(xy) \leq |p'| + |q| + 2 = (1+\varepsilon)D^{k}_{\mathrm{PB}}(x) + D^{k}_{\mathrm{PB}}(y) + 2. \tag{6.2}$$

□

6.3 Pebble Depth

In this section we present our notion of *pebble depth (PB-depth)*, show it satisfies the three basic fundamental properties of depth and identify a normal PB-deep sequence.

We define PB-depth by examining the difference in k-FS and k'-PB complexity on prefixes on sequences. This keeps to the spirit of Bennett's original notion of comparing Kolmogorov complexity against its restricted time-bounded version. Here, FSTs can be viewed as a restricted subset of 0-pebble transducers which can only move in one direction.

Definition 6.3.1. A sequence S is *pebble deep (PB-deep)* if

$$(\exists \alpha > 0)(\forall k \in \mathbb{N})(\exists k' \in \mathbb{N})(\forall^\infty n \in \mathbb{N}) D^k_{\mathrm{FS}}(S \upharpoonright n) - D^{k'}_{\mathrm{PB}}(S \upharpoonright n) \geq \alpha n.$$

Definition 6.3.2. Let $S \in \{0,1\}^\omega$. We say that PB-*depth*$(S) \geq \alpha$ if

$$(\forall k \in \mathbb{N})(\exists k' \in \mathbb{N})(\forall^\infty n \in \mathbb{N})\, D^k_{\mathrm{FS}}(S \upharpoonright n) - D^{k'}_{\mathrm{PB}}(S \upharpoonright n) \geq \alpha n.$$

Otherwise we say PB-*depth*$(S) < \alpha$.

Unlike in Chapters 4 and 5 where compressors were used to define depth, we will take the minimal descriptional approach as opposed to using pebble compressors because of their two-way nature, issues arise when trying to define what compressibility by two-way compressors means. We similarly did not compare pebble transducers against pebble transducers to define depth, unlike with FSTs in Chapter 3, to keep with the spirit of Bennett's original notion. Other difficulties arise if this approach is taken also, one of which is finding an analogous result to Lemma 3.3.9. Specifically, unlike in the finite-state setting where from a description of the string xy we could get a description of both x and y separately by switching the start state of the transducer used to output xy, this trick does not work in the pebble setting due to their two-way nature and due to the pebbles. Section 6.4 of this chapter discusses how we arrived at this definition of depth in more detail.

6.3.1 Fundamental Properties

Before we show that pebble depth satisfies the fundamental properties of depth, we first show the existence of PB-incompressible sequences. PB-trivial sequences are evident in our later proofs. To demonstrate this, we first need the following result which relates H and K, the prefix-free and plain version of Kolmogorov

complexity. It can be found as Corollary 2.4.2 in [118].

Lemma 6.3.3. *For all $x \in \{0,1\}^*$ it holds that*

$$H(x) \leq K(x) + 2\log(K(x)) + O(1) \leq K(x) + 2\log(|x|) + O(1).$$

Lemma 6.3.4. *If $S \in \{0,1\}^\omega$ is ML-random, then $\rho_{\mathrm{PB}}(S) = 1$.*

Proof. Let $S \in \{0,1\}^\omega$ be ML-random. Hence for all n there exists some $c \in \mathbb{N}$ such that $H(S \upharpoonright n) > n - c$.

Fix $k \in \mathbb{N}$ and consider the prefix $S \upharpoonright n$ of S. Let $T \in \mathrm{PB}^{\leq k}$ and $y \in \{0,1\}^*$ be such that $T(y) = S \upharpoonright n$ and $D^k_{\mathrm{PB}}(S \upharpoonright n) = |y|$. Let M be the machine such that on inputs of the form $d(\sigma)01x$, where σ is a description of a pebble transducer via our encoding and x is in the domain of the pebble transducer that σ describes, M uses $d(\sigma)$ to retrieve the pebble transducer and then simulates the transducer on x. Otherwise, M loops.

Hence we have that

$$K(S \upharpoonright n) \leq 2|\sigma| + 2 + |y| + O(1) \leq 2k + 2 + D^k_{\mathrm{PB}}(S \upharpoonright n) + O(1).$$

By Lemma 6.3.3 it follows that

$$H(S \upharpoonright n) \leq 2k+2+D^k_{\mathrm{PB}}(S \upharpoonright n)+2\log(n)+O(1) = D^k_{\mathrm{PB}}(S \upharpoonright n)+2\log(n)+O(1).$$

Therefore

$$D^k_{\mathrm{PB}}(S \upharpoonright n) > n - c - 2\log(n) - O(1) = n - 2\log(n) - O(1).$$

Note if no y as above exists, this still holds as in such cases $D^k_{\mathrm{PB}}(S \upharpoonright n) = \infty$.

Therefore, for all k we have that

$$1 = \liminf_{n\to\infty} \frac{n - 2\log(n) - O(1)}{n} \leq \liminf_{n\to\infty} \frac{D^k_{\mathrm{PB}}(S \upharpoonright n)}{n} \leq 1.$$

As k was arbitrary, it follows that $\rho_{\mathrm{PB}}(S) = 1$.

□

The following demonstrates that sequences which are FST-trivial and PB-incompressible (in the sense of Definition 2.4.3) are not PB-deep. This is analogous to Bennett's fundamental properties of computable and ML-random sequences being shallow.

Theorem 6.3.5. *Let $S \in \{0,1\}^\omega$. If $R_{\mathrm{FS}}(S) = 0$ or $\rho_{\mathrm{PB}}(S) = 1$ then S is not PB-deep.*

Proof. Suppose that $R_{\mathrm{FS}}(S) = 0$. Let $\alpha > 0$. Let k be such that for almost every n

$$D^k_{\mathrm{FS}}(S \upharpoonright n) \leq \alpha n. \tag{6.3}$$

Then for all $k' \in \mathbb{N}$, for almost every n we have that

$$D^k_{\mathrm{FS}}(S \upharpoonright n) - D^{k'}_{\mathrm{PB}}(S \upharpoonright n) \leq D^k_{\mathrm{FS}}(S \upharpoonright n) < \alpha n. \tag{6.4}$$

As α was chosen arbitrarily, S is not PB-deep.

Next suppose that $\rho_{\mathrm{PB}}(S) = 1$. Therefore for all $\alpha > 0$ and $k \in \mathbb{N}$, for almost every n it holds that

$$D^k_{\mathrm{PB}}(S \upharpoonright n) \geq (1 - \varepsilon). \tag{6.5}$$

Therefore we have for k' such that $I_{\mathrm{FS}} \in \mathrm{FST}^{\leq k'}$,

$$D^{k'}_{\mathrm{FS}}(S \upharpoonright n) - D^k_{\mathrm{PB}}(S \upharpoonright n) \leq n - (1 - \alpha)n = \alpha n. \tag{6.6}$$

As α was chosen arbitrarily, S is not PB-deep

□

We now demonstrate that PB-depth also satisfies a slow growth law. In the following theorem we show that if a deep sequence S' is the output of some ILFS computable mapping, the original sequence S used to compute S' must also have been deep. This is analogous to Bennett's slow growth law as it demonstrates that the fast process of computation by an ILFST cannot transform a non-deep sequence into a deep sequence.

Theorem 6.3.6 (Slow Growth Law)**.** *Let S be a sequence. Let $f : \{0,1\}^\omega \longrightarrow \{0,1\}^\omega$ be* ILFS *computable, and let $S' = f(S)$. If S' is PB-deep, then S is PB-deep.*

Proof. The proof is similar to Theorem 3.3.8's. Let S, S', f be as stated, and M be an ILFST computing f.

For all n such that $M(S \upharpoonright m) = S' \upharpoonright n$ for some m, let m_n denote the largest integer such that $M(S \upharpoonright m_n) = S' \upharpoonright n$. As M is IL, it cannot visit the same state twice without outputting at least one bit, so there exist a $\beta > 0$ such that for all n, $n \geq \beta m_n$.

Fix $l \in \mathbb{N}$. Let k be from Lemma 3.3.6 such that for all $x \in \{0,1\}^*$

$$D^k_{\mathrm{FS}}(M(x)) \leq D^l_{\mathrm{FS}}(x). \tag{6.7}$$

As S' is PB-deep, there exists k' and $\alpha > 0$ such that for almost every n

$$D^k_{\mathrm{FS}}(S' \upharpoonright n) - D^{k'}_{\mathrm{PB}}(S \upharpoonright n) \geq \alpha n. \tag{6.8}$$

Similarly let l' be from Lemma 6.2.7 such that for all x

$$D^{l'}_{\mathrm{PB}}(x) \leq D^{k'}_{\mathrm{PB}}(M(x)). \tag{6.9}$$

Hence for almost every m we have that

$$\begin{aligned}
D^{l}_{\mathrm{FS}}(S \upharpoonright m) - D^{l'}_{\mathrm{PB}}(S \upharpoonright m) &\geq D^{k}_{\mathrm{FS}}(M(S \upharpoonright m) - D^{l'}_{\mathrm{PB}}(S \upharpoonright m) && \text{(by (6.7))} \\
&\geq D^{k}_{\mathrm{FS}}(M(S \upharpoonright m) - D^{k'}_{\mathrm{PB}}(M(S \upharpoonright m)) && \text{(by (6.9))} \\
&= D^{k}_{\mathrm{FS}}(M(S \upharpoonright m_n) - D^{k'}_{\mathrm{PB}}(M(S \upharpoonright m_n)) && \text{(for some } n\text{)} \\
&= D^{k}_{\mathrm{FS}}(S' \upharpoonright n) - D^{k'}_{\mathrm{PB}}(S' \upharpoonright n) \\
&\geq \alpha n && \text{(by (6.8))} \\
&\geq \alpha\beta m_n \geq \alpha\beta m.
\end{aligned}$$

Hence S is PB-deep.

□

Next we show that if a transformation via a pebble transducer is performed instead of via an ILFST, the existence of a sequence S such that $\rho_{\mathrm{PB}}(S) = 1$ would break the alternative slow growth law. First we need the following definition of a pebble-computable function.

Definition 6.3.7. A function $f : \{0,1\}^\omega \to \{0,1\}^\omega$ is said to be pebble computable if there is exists some $T \in \mathrm{PB}$ such that for all $S \in \{0,1\}^\omega$, $\lim_{n\to\infty} |T(S \upharpoonright n)| = \infty$ and for all $n \in \mathbb{N}$, $T(S \upharpoonright n) \sqsubseteq f(S)$.

We now present this result which breaks the alternative slow growth law.

Lemma 6.3.8. *Let $S \in \{0,1\}^\omega$ be such that $\rho_{PB}(S) = 1$. There exists a pebble computable function f such that for $S' = f(S)$, S' is PB-deep while S is not PB-deep.*

Proof. Let $S \in \{0,1\}^\omega$ be such that $\rho_{\mathrm{PB}}(S) = 1$. Recall the function pref on strings and note that pref can extended to be a pebble computable function on sequences. We will show that $S' = \mathrm{pref}(S)$ is PB-deep even though S is not PB-deep by Theorem 6.3.5.

Let $0 < \varepsilon < 1$. Note that as $\rho_{\mathrm{PB}}(S) = 1$, it follows that $\dim_{\mathrm{FS}}(S) = 1$ also. Hence, for all k and almost every n it holds that

$$D^k_{\mathrm{FS}}(S \upharpoonright n) > n(1 - \frac{\varepsilon}{3}). \tag{6.10}$$

Consider an arbitrary prefix $S' \upharpoonright n$ of S'. Let j be such that

$$\frac{j(j+1)}{2} \leq n < \frac{(j+1)(j+2)}{2}.$$

$S' \upharpoonright n$ can be written in the form $x_1 x_2 \dots x_j y$ where $x_i = S[0..i-1]$ and $y \sqsubset x_{j+1}$.

Suppose j^* is such that Equation (6.10) holds for all $j \geq j^*$. Then by Remark 3.3.11 it follows that for almost all k and large enough n,

$$\begin{aligned}
D^k_{\mathrm{FS}}(S' \upharpoonright n) &\geq \sum_{i=1}^{j} D^{3k}_{\mathrm{FS}}(x_i) + D^{3k}_{\mathrm{FS}}(y) \\
&> \sum_{i=j^*}^{j} D^{3k}_{\mathrm{FS}}(x_i) \\
&> \sum_{i=j^*}^{j} |x_i|(1 - \frac{\varepsilon}{3}) && \text{(by (6.10))} \\
&= (n - O(1) - |y|)(1 - \frac{\varepsilon}{3}) \\
&> n(1 - \frac{2\varepsilon}{3}) && (6.11)
\end{aligned}$$

as $|y| = O(\sqrt{n})$.

Similarly, let T be the pebble transducer such that on inputs of the form $d(x)01z$, T uses $d(x)$ and a pebble to print $\mathrm{pref}(x)$ and then uses z to print z. Hence, $T(d(x_j)01y) = S' \upharpoonright n$. Thus for n large,

$$D^{|T|}_{\mathrm{PB}}(S' \upharpoonright n) \leq 2j + 2 + |y| < 3(j+1) = O(\sqrt{n}). \tag{6.12}$$

Hence, for almost every k and for n large it holds that

$$D^k_{\mathrm{FS}}(S' \upharpoonright n) - D^{|T|}_{\mathrm{PB}}(S' \upharpoonright n) > n(1 - \frac{2\varepsilon}{3}) - \frac{n\varepsilon}{3} = n(1 - \varepsilon). \tag{6.13}$$

As Equation (6.13) in fact holds for every k as $D^i_{\mathrm{FS}}(x) \geq D^{i+1}_{\mathrm{FS}}(x)$ for all strings x, S' is in fact PB-deep. Thus the alternative slow growth law breaks.

For completeness, we provide the following construction for T: As T is a 1-pebble transducer, the pebble placement part of the transition and output function will have value 0 or 1 indicating whether or not the pebble is present on the current square of the input tape. Note that some transitions are omitted if they are not seen.

Let $T = (Q, q_0, \{q_f\}, 1, \delta, \nu)$ be as follows. T has the following set of states:

1. q_s the start state.
2. q_p is the state T enters when it needs to move its pebble.
3. q^b is the state which records the first bit for $b \in \{0, 1\}$ when examining a block of size 2.
4. q_l is the state used when T continuously moves its head to the left side of the tape.
5. q_1, q_2, q_3 are the states used to print the prefixes of the input.
6. q_i is the state where T acts as the identity transducer.
7. q_f is the final state.

Beginning in the start state, T moves its head to the right and enters the pebble placement state

$$\delta(q_s, \dashv, 0) = (q_p, +1).$$

Beginning in q_p, T then reads the next two bits. T first records the first bit and moves right

$$\delta(q_p, b, 0) = (q^b, +1).$$

Then reading the second bit, if it matches the first bit, T places a pebble onto the square and enters the state for scanning to the left. If they do not match, T moves right and enters the identity state. That is

$$\delta(q^b, a, 0) = \begin{cases} (q_l, \text{push}) & \text{if } a = b \\ (q_i, +1) & \text{if } a \neq b. \end{cases}$$

In q_l, T scans left to the end of the tape, i.e. for $b, c \in \{0, 1\}$,

$$\delta(q_l, b, c) = (q_l, -1).$$

When T reaches the end of the tape, it begins reading in chunks of size two, printing every second bit, until it sees the square containing the pebble. T first moves its head right,

$$\delta(q_l, \dashv, 0) = (q_1, +1).$$

T then moves its head to the right to the second square on any bit,

$$\delta(q_1, b, 0) = (q_2, +1).$$

In q_2, if the current square contains the pebble, T pops the pebble and moves it forward two squares. If it does not, T moves right and returns to q_1. That is, on any bit b,

$$\delta(q_2, b, c) = \begin{cases} (q_1, +1) & \text{if } c = 0 \\ (q_3, \text{pop}) & \text{if } c = 1. \end{cases}$$

In q_3, T returns to q_p and moves its head to the right to begin the process of moving the pebble again. That is,

$$\delta(q_3, b, 0) = (q_p, +1).$$

When in state q_i, T moves right regardless of the bit read. That is,

$$\delta(q_i, b, 0) = (q_i, +1).$$

T enters its final state if T reaches the right hand side of the tape in states q_p, q^b or q_i. That is, for $q \in \{q_p, q^b, q_i\}$,

$$\delta(q, \vdash, 0) = (q_f, -1).$$

T outputs the empty string on all transitions except in the following cases where it prints the bit on the current square:

$$\text{For } c \in \{0,1\},\ \nu(q_2, b, c) = b \text{ and } \nu(q_i, b, 0) = b.$$

This completes the construction of T.

□

6.3.2 Separation from Finite-State Depth

The following demonstrates a difference between FS-depth and PB-depth. Recall that for both i.o. FS-depth ([57]) and a.e. FS-depth (Theorem 3.3.5), normal sequences are not deep. On the other hand, the normal sequence from Theorem 5.4.1 (i.e. the sequence from Theorem 4.3 of [97]) which is LZ-deep is also PB-deep.

Remark 6.3.9 ([97])**.** There exists $S \in \{0,1\}^\omega$ such that the normal sequence S' from Theorem 5.4.1 satisfies $S' = \text{pref}(S)$.

As there exists a 1-pebble transducer that is able to compute the pref function, prefixes of S have low k-pebble complexity. Hence PB-depth is achieved.

Theorem 6.3.10. *There exists a normal sequence which is PB-deep.*

Proof. Let S' be the normal sequence from Theorem 4.3 of [97]. S' has the property that there exists $S \in \{0,1\}^\omega$ such that $\text{pref}(S) = S'$. We claim S' satisfies the theorem.

The proof follows from the same logic as the proof of Lemma 6.3.8 by noting that as S' is normal, Equation (6.11) still holds. That is, that for all k and almost every n,

$$D^k_{\text{FS}}(S' \upharpoonright n) \geq (1 - \frac{2\varepsilon}{3}).$$

Similarly, Equation (6.12) still holds for the same pebble transducer T. Thus, Equation (6.13) still holds meaning that S' is PB-deep.

□

6.3.3 Separation from Lempel-Ziv Depth

The following demonstrates the existence of a sequence S with PB-depth of roughly 1/2 and low LZ-depth. The sequence is that from Theorem 5 of [109]. This sequence is broken into blocks where each block is a concatenation of most of the strings of length n. Specifically blocks are composed of subblocks of the form XFY where X is a listing of a selection of strings of length n, F is a flag not contained in any string of length n listed, and Y is a listing of strings of length n such that $Y = X^{-1}$. A pebble transducer can perform well on this sequence as given X, the transducer can use its two-way tape property to print Y also. LZ

does not compress S by much as it is almost a listing of every string in order of length. LZ compresses such sequences poorly.

Theorem 6.3.11. *For each $0 < \beta < \frac{1}{2}$, there exists a sequence S such that PB-depth$(S) \geq \frac{1}{2} - \beta$ and LZ-depth$(S) < \beta$.*

Proof. Let S be a sequence that satisfies Theorem 5 of [109]. We claim that S satisfies the theorem statement. We present the construction of the sequence again for clarity even though it was already given in Theorem 5.5.2.

Let $0 < \beta < \frac{1}{2}$, and let $k > 2$ and v be integers to be determined later. For any $n \in \mathbb{N}$, let T_n denote the set of strings of length n that do not contain the substring 1^j in x for all $j \geq k$. As T_n contains the set of strings whose every k^{th} bit is 0, it follows that $|T_n| \geq 2^{(\frac{k-1}{k})n}$. Note that for every $x \in T_n$, there exists $y \in T_{n-1}$ and $b \in \{0,1\}$ such that $x = yb$. Hence

$$|T_n| < 2|T_{n-1}|. \tag{6.14}$$

Let $A_n = \{a_{1_n}, \ldots, a_{u_n}\}$ be the set of palindromes in T_n. As fixing the first $\lceil \frac{n}{2} \rceil$ bits determines a palindrome, $|A_n| \leq 2^{\lceil \frac{n}{2} \rceil}$. The remaining strings in $T_n - A_n$ are split into $v+1$ pairs of sets $X_{n,i} = \{x_{n,i,1}, \ldots, x_{n,i,t_n^i}\}$ and $Y_{n,i} = \{y_{n,i,1}, \ldots, y_{n,i,t_n^i}\}$ where $t_n^i = \lfloor \frac{|T_n - A_n|}{2v} \rfloor$ if $i \neq v+1$ and

$$t_n^{v+1} = \frac{1}{2}(|T_n - A_n| - 2\sum_{i=1}^{v} |X_{n,i}|),$$

$(x_{n,i,j})^{-1} = y_{n,i,j}$ for every $1 \leq j \leq t_n^i$ and $1 \leq i \leq v+1$ both $x_{n,i,1}$ and y_{n,i,t_n} start with 0 (that is, x_{n,i,t_n^i} ends with a 0) excluding the case where both $X_{n,v+1}$ and $Y_{n,v+1}$ are the empty sets). Note that for convenience we write X_i, Y_i for $X_{n,i}, Y_{n,i}$ respectively.

S is constructed in stages. Let $f(k) = 2k$ and $f(n+1) = f(n) + v + 2$. Note that $n < f(n) < n^2$ for large n. For $n \leq k-1$, S_n is a concatenation of all strings

of length n, i.e. $S_n = 0^n \cdot 0^{n-1}1 \cdots 1^{n-1}0 \cdot 1^n$. For $n \geq k$,

$$S_n = a_{1_n} \dots a_{u_n} 1^{f(n)} z_{n,1} z_{n,2} \dots z_{n,v} z_{n,v+1}$$

where

$$z_{n,i} = x_{n,i,1} x_{n,i,2} \dots x_{n,i,t_n^i - 1} x_{n,i,t_n^i} 1^{f(n)+i} y_{n,i,t_n^i} y_{n,i,t_n^i - 1} \dots y_{n,i,2} y_{n,i,1},$$

with the possibility that $z_{n,v+1} = 1^{f(n)+v+1}$ only. That is, S_n is a concatenation of all strings in A_n followed by a flag of $f(n)$ ones, followed by a concatenation of all strings in the X_i zones and Y_i zones separated by flags of increasing length such that each Y_i zone is the X_i zone written in reverse. Finally we set

$$S = S_1 S_2 \dots S_{k-1} 1^k 1^{k+1} \dots 1^{2k-1} S_k S_{k+1} \dots$$

i.e. the concatenation of all S_j zones with some extra flags between S_{k-1} and S_k.

We first examine the lower randomness density of S for pebble transducers.

Claim 6.3.12.

$$\lim_{k \to \infty} \limsup_{n \to \infty} \frac{D^k_{\mathrm{PB}}(S \upharpoonright n)}{n} \leq \frac{1}{2}.$$

We prove this claim by building the 1-pebble transducer T that acts as follows: T begins moving right and printing its input until it sees the first 0 after a flag of $2k$ ones. Upon seeing this 0, if the succeeding bit is a 1, T stays in the print zone. T moves right and prints what is on its tape until it sees a flag of $2k$ ones followed by a 0 again. If T sees a 0 after $1^{2k}0$, T enters a print-and-reverse zone. T drops its pebble on the succeeding square. T moves its head right printing what it sees until it sees $1^{2k}0$ (without printing the last 0), then scans left past the flag of 1s. Once the flag of 1s ends, T prints what it sees (i.e. printing the reverse of what it just printed) until it reaches the square with the pebble, printing what is on

it. T then moves right until it sees $1^{2k}0$ again and checks the next bit to see if it is in a print or print-and-reverse zone.

Let $y = S_1 \dots S_{k-1} 1^k \dots 1^{2k-1}$. Then $T(y0) = y$. Note that $|y| + 1 < 2^{2k}$. For $n \geq k$ and $1 \leq i \leq v$, let

$$\pi_n = 1a_{1_n} \dots a_{u_n} 1^{f(n)} 0 \text{ and } \sigma_{n,i} = 0x_{n,i,1} \cdots x_{n,i,t_n^i} 1^{f(n)+i} 0.$$

If $i = v + 1$ we let

$$\sigma_{n,v+1} = \begin{cases} 0x_{n,v+1,1} \cdots x_{n,v+1,t_n^{v+1}} 1^{f(n)+v+1} 0 & \text{if } |X_{n,v+1}| \neq 0 \\ 11^{f(n)+v+1} 0 & \text{otherwise.} \end{cases}$$

Lastly we set

$$\tau_n = \pi_n \sigma_{n,1} \sigma_{n,2} \dots \sigma_{n,v+1}.$$

Note that

$$\begin{aligned} |\tau_n| &\leq |A_n| n + (v+2)(f(n) + v + 1) + 2(v+2) + \frac{n}{2}|T_n - A_n| \\ &= |A_n| n + (v+2)(f(n) + v + 3) + \frac{n}{2}|T_n - A_n|. \end{aligned} \tag{6.15}$$

Then as $T(y0\tau_k \dots \tau_{n-1}) = S_1 \dots S_{k-1} 1^k \dots 1^{2k-1} S_k \dots S_{n-1}$, it follows that

$$\begin{aligned} D_{\mathrm{PB}}^{|T|}(S_1 \dots S_{k-1} 1^k \dots 1^{2k-1} S_k \dots S_{n-1}) &\leq |S_1 \dots S_{k-1} 1^k \dots 1^{2k-1}| + 1 \\ &+ \sum_{j=k}^{n-1} [|A_j| j + (v+2)(f(j) + v + 3) + \frac{j}{2}|T_j - A_j|]. \end{aligned} \tag{6.16}$$

Let w_p be the string such that $T(w_p) = S \restriction p$. Note that the ratio $\frac{|w_p|}{|p|}$ is maximal if the suffix of $S \restriction p$ is a full concatenation of a $Y_{n,i}$ zone without the

final bit. That is, $S \restriction p$ ends with a suffix of the form

$$y_{n,i,t_n^i} \dots y_{n,i,2} y_{n,i,1}[0..n-2].$$

This is because T cannot make use of its two-way capability to print the reverse of the X zone since it does not know where to stop. In particular, the ratio is maximal on the zone $i = 1$ as it immediately follows the palindrome portion of S_n where T acts as the identity transducer to output it.

Let $0 \leq I < v$. We do not examine the case where $I = v$ as in this case, T requires the fewest amount of bits to output the $v + 1^{\text{th}}$ zone. We examine the ratio $\frac{|w_p|}{|S \restriction p|}$ inside zone S_n on the second last symbol of the Y_{I+1} zone. Note that T outputs $S \restriction p$ on input

$$y 0 \tau_k \dots \tau_{n-1} \pi_n \sigma_{n,1} \dots \sigma_{n,I} 1 z$$

where

$$z = x_{n,I+1,1} \dots x_{n,I+1,t_n} 1^{f(n)+I+1} y_{n,I+1,t_n} \dots y_{n,I+1,2} y_{n,I+1,1}[0..n-2].$$

Thus

$$\begin{aligned} |w_p| \leq 2^{2k} + \sum_{j=k}^{n-1} [|A_j| j + (v+2)(f(j)+v+3) + \frac{j}{2}|T_j - A_j|] \\ + |A_n| n + (v+2)(f(n)+v+3) + I(\frac{n|T_n - A_n|}{2v}) + \frac{n|T_n - A_n|}{v}. \end{aligned} \tag{6.17}$$

Note first that

$$\sum_{j=k}^{n} |A_j| j \leq n^2 |A_n| \leq n^2 \cdot 2^{\lceil \frac{n}{2} \rceil} \leq n^2 \cdot 2^{\frac{n+1}{2}} \tag{6.18}$$

for n large. Similarly the summation of the $(f(j) + v + 2)$ contributes at most a

polynomial number of bits in n. Along with the 2^{2k} term being a constant term this gives us for all $\varepsilon > 0$, for n large

$$\begin{aligned} |w_p| &\leq 2^{n(\frac{1}{2}+\varepsilon)} + \sum_{j=k}^{n-1} \frac{j}{2}|T_j| + \frac{n|T_n|}{v}(\frac{I}{2}+1) \\ &= 2^{n(\frac{1}{2}+\varepsilon)} + \sum_{j=k}^{n-1} \frac{j}{2}|T_j| + \frac{n|T_n|}{2v}(I+2). \end{aligned} \tag{6.19}$$

The number of bits in such a prefix of S is

$$\begin{aligned} |S \restriction p| &\geq \sum_{j=k}^{n-1} j|T_j| + n|A_n| + 2n\Big\lfloor \frac{|T_n - A_n|}{2v} \Big\rfloor (I+1) \\ &\geq \sum_{j=k}^{n-1} |T_j| + n|A_n| + 2n(\frac{|T_n - A_n|}{2v} - 1)(I+1) \\ &= \sum_{j=k}^{n-1} |T_j| + n|A_n| + n(\frac{|T_n| - |A_n|}{v} - 2)(I+1) \\ &= \sum_{j=k}^{n-1} j|T_j| + n|A_n|(1 - \frac{(I+1)}{v}) + n(I+1)(\frac{|T_n|}{v} - 2) \\ &\geq \sum_{j=k}^{n-1} j|T_j| + \frac{n}{v}|T_n|(I) \end{aligned} \tag{6.20}$$

as $I + 1 \leq v$.

Hence,

$$\limsup_{n\to\infty}\frac{|w_n|}{|S\upharpoonright n|}\leq\limsup_{n\to\infty}\frac{2^{n(\frac{1}{2}+\varepsilon)}+\sum_{j=k}^{n-1}\frac{j}{2}|T_j|+\frac{n|T_n|}{2v}(I+2)}{\sum_{j=k}^{n-1}j|T_j|+\frac{n|T_n|}{v}(I)}$$

(by (6.19) and (6.20))

$$\begin{aligned}&=\limsup_{n\to\infty}\left[\frac{2^{n(\frac{1}{2}+\varepsilon)}+2\cdot\frac{n|T_n|}{2v}}{\sum_{j=k}^{n-1}j|T_j|+\frac{n|T_n|}{v}(I)}\right.\\&\left.+\frac{1}{2}\cdot\frac{\sum_{j=k}^{n-1}j|T_j|+\frac{n|T_n|}{v}(I)}{\sum_{j=k}^{n-1}j|T_j|+\frac{n|T_n|}{v}(I)}\right]\\&=\limsup_{n\to\infty}\left[\frac{2^{n(\frac{1}{2}+\varepsilon)}+\frac{n|T_n|}{v}}{\sum_{j=k}^{n-1}j|T_j|+\frac{n|T_n|}{v}(I)}+\frac{1}{2}\right].\end{aligned}\tag{6.21}$$

By (6.14), as $\sum_{j=k}^{n-1}j|T_j|\geq(n-1)|T_{n-1}|\geq\frac{(n-1)}{2}|T_n|$, we have

$$\begin{aligned}\sum_{j=k}^{n-1}j|T_j|+\frac{n}{v}|T_n|(I)&\geq\frac{n-1}{2}|T_n|+\frac{n}{v}|T_n|(I)\\&=\frac{n|T_n|}{2v}(2I+v-\frac{v}{n}).\end{aligned}\tag{6.22}$$

Thus, when ε is chosen to be such that $0<\varepsilon<\frac{1}{2}-\frac{1}{k}$ we have that

$$\begin{aligned}\limsup_{n\to\infty}\frac{2^{n(\frac{1}{2}+\varepsilon)}}{\sum_{j=k}^{n-1}j|T_j|+\frac{n|T_n|}{v}(I)}&\leq\limsup_{n\to\infty}\frac{2^{n(\frac{1}{2}+\varepsilon)}}{\frac{(n-1)}{2}|T_n|}\leq\limsup_{n\to\infty}\frac{2^{n(\frac{1}{2}+\varepsilon)}}{|T_n|}\\&\leq\limsup_{n\to\infty}\frac{2^{n(\frac{1}{2}+\varepsilon)}}{2^{\frac{(k-1)n}{k}}}=0\end{aligned}$$

as $k>2$. Similarly by (6.22) we have

$$\frac{\frac{n|T_n|}{v}}{\sum_{j=k}^{n-1}j|T_j|+\frac{n|T_n|}{v}(I+1)}\leq\frac{\frac{n|T_n|}{v}}{\frac{n|T_n|}{2v}(2I+v-\frac{v}{n})}\leq\frac{2}{v(1-\frac{1}{n})}\tag{6.23}$$

which can be made arbitrarily small by choosing v appropriately large.

Therefore

$$\limsup_{n\to\infty}\frac{|w_n|}{|S\upharpoonright n|}\leq\frac{1}{2}.$$

This establishes Claim 6.3.12, i.e. for all $0 < \beta' < 1/2 - 3/k$, we can choose v, I and k appropriately such that

$$D_{\mathrm{PB}}^{|T|}(S \upharpoonright n) \leq (\frac{1}{2} + \frac{\beta'}{2})n. \tag{6.24}$$

Next we examine how well any ILFST can compress prefixes of S. We use the equality from Definition 3.2.6 regarding the finite-state dimension of S to relate the compression performance back to k-finite-state complexity.

Recall from the proof of Theorem 5.5.2 it was shown in Equation (5.34) that $\rho_{\mathrm{UPD}}(S) \geq \frac{k-3}{k}$. Hence it follows that

$$\dim_{\mathrm{FS}}(S) = \rho_{\mathrm{ILFST}}(S) \geq \rho_{\mathrm{UPD}}(S) \geq \frac{k-3}{k}.$$

Therefore, for all l and almost every n it holds that

$$D_{\mathrm{FS}}^{l}(S \upharpoonright n) \geq (\frac{k-3}{k} - \frac{\beta'}{2})n. \tag{6.25}$$

Hence, for all l and almost every n it follows that

$$\begin{aligned} D_{\mathrm{FS}}^{l}(S \upharpoonright n) - D_{\mathrm{PB}}^{|T|}(S \upharpoonright n) &\geq (\frac{k-3}{k} - \frac{\beta'}{2})n - (\frac{1}{2} + \frac{\beta'}{2})n \\ &= (\frac{1}{2} - \frac{3}{k} - \beta')n \\ &\geq (\frac{1}{2} - \beta)n \end{aligned} \tag{6.26}$$

where $3/k + \beta' \leq \beta < 1/2$. That is, PB-depth$(S) \geq 1/2 - \beta$.

Recall from [109], for all $\beta < \frac{1}{2}$, k and v can be chosen such that

$$\rho_{\mathrm{LZ}}(S) > (1 - \frac{\beta}{2}). \tag{6.27}$$

Again, therefore we have that for almost every n that

$$|\mathrm{LZ}(S \upharpoonright n)| > (1-\beta)n \tag{6.28}$$

and so

$$|I_{\mathrm{FS}}(S \upharpoonright n)| - |\mathrm{LZ}(S \upharpoonright n)| < n - (1-\beta)n = \beta n. \tag{6.29}$$

That is, LZ-depth$(S) < \beta$.

Thus, if k, v and β' are chosen such that Equations (6.26) and (6.29) hold, we have the desired result.

For completeness, the following is a construction for T: $T = (Q, q_o, F, 1, \delta, \nu)$ is the 1-pebble transducer whose states are follows:

1. q_0 the start state,
2. $q_{i,w}$ for $w \in \{0,1\}^{2k}$ the just printing states,
3. q_1 the state used to check whether the transducer just prints or needs to print the reverse too,
4. q_p a state used to place the pebble,
5. $q_{r,w}$ for $w \in \{0,1\}^{2k}$, the state where T moves right printing but will print the reverse too,
6. q_f the state when scanning left along the flag before printing the reverse,
7. q_l the state used to print the reverse moving left,
8. $q_{s,w}$ for $w \in \{0,1\}^{2k}$ used to scan right,
9. q_F the final state.

So $F = \{q_F\}$.

From the start state, T moves to state $q_{i,0^{2k}}$ and prints nothing. That is,

$$\delta(q_0, \dashv, 0) = (q_{i,0^{2k}}, +1),$$

and

$$\nu(q_0, \dashv, 0) = \lambda.$$

From here, T continuously prints what is under its head moving right until it sees the end of a flag. At the end of the flag it moves to q_1. That is for $w \in \{0,1\}^{2k}$ and $b \in \{0,1\}$

$$\delta(q_{i,w}, b, 0) = \begin{cases} (q_{i,w[1..]b}, +1) & \text{if } w \neq 1^{2k} \text{ or } (w = 1^{2k} \text{ and } b = 1) \\ (q_1, +1) & \text{if } w = 1^{2k} \text{ and } b = 0, \end{cases}$$

and

$$\nu(q_{i,w}, b, 0) = \begin{cases} b & \text{if } w \neq 1^{2k} \text{ or } (w = 1^{2k} \text{ and } b = 1) \\ \lambda & \text{if } w = 1^{2k} \text{ and } b = 0. \end{cases}$$

In q_1, T has just read a 0 after a flag of 1^{2k}. If T reads a 1 in q_1, T moves right and returns to $q_{i,0^{2k}}$ the initial printing state. If T reads a 0, T moves right and enters the state q_p and places its pebble on its tape. That is,

$$\delta(q_1, b, 0) = \begin{cases} (q_p, +1) & \text{if } b = 0 \\ (q_{i,0^{2k}}, +1) & \text{if } b = 1. \end{cases}$$

T prints nothing in q_1. That is, for $b, c \in \{0,1\}$

$$\nu(q_1, b, c) = \lambda.$$

In q_p, T places a pebble on its current square and enters state $q_{r,0^{2k}}$ and prints nothing. That is, for $b \in \{0,1\}$,

$$\delta(q_p, b, 0) = (q_{r,0^{2k}}, \text{push}),$$

and

$$\nu(q_p, b, 0) = \lambda.$$

T moves its head to the right printing what it reads when in states $q_{r,w}$. It does this until it sees the end of a 1^{2k} flag, upon which it enters state q_f moving its head to the left. That is, for $b, c \in \{0,1\}$, $w \in \{0,1\}^{2k}$,

$$\delta(q_{r,w}, b, c) = \begin{cases} (q_{r,w[1..]b}, +1) & \text{if } w \neq 1^{2k} \text{ or } (w = 1^{2k} \text{ and } b = 1) \\ (q_f, -1) & \text{if } w = 1^{2k} \text{ and } b = 0, \end{cases}$$

and

$$\nu(q_{r,w}, b, c) = \begin{cases} b & \text{if } w \neq 1^{2k} \text{ or } (w = 1^{2k} \text{ and } b = 1) \\ \lambda & \text{if } w = 1^{2k} \text{ and } b = 0. \end{cases}$$

T moves its head to the left printing nothing while in q_f until it sees a 0, that is, the end of the 1^{2k} flag zone. When it sees a 0, T begins printing what it reads and enters state q_l. That is for $b \in \{0,1\}$,

$$\delta(q_f, b, 0) = \begin{cases} (q_f, -1) & \text{if } b = 1 \\ (q_l, -1) & \text{if } b = 0, \end{cases}$$

and

$$\nu(q_f, b, 0) = \begin{cases} \lambda & \text{if } b = 1 \\ 0 & \text{if } b = 0. \end{cases}$$

In q_l, T moves its head to the left printing what it sees until it sees the square

with the pebble. When T sees the pebble, T removes the pebble and enters state $q_{s,0^{2k}}$. That is for $b, c \in \{0,1\}$

$$\delta(q_l, b, c) = \begin{cases} (q_l, -1) & \text{if } c = 0 \\ (q_{s,0^{2k}}, \text{pop}) & \text{if } c = 1, \end{cases}$$

and

$$\nu(q_l, b, c) = b.$$

T moves its head to the right printing nothing until it sees the end of a 1^{2k} flag, upon which it enters state q_1 to begin the process of printing a new zone again. That is, for $b \in \{0,1\}, w \in \{0,1\}^{2k}$

$$\delta(q_{s,w}, b, 0) = \begin{cases} (q_{s,w[1..]b}, +1) & \text{if } w \neq 1^{2k} \text{ or } (w = 1^{2k} \text{ and } b = 1) \\ (q_1, +1) & \text{if } w = 1^{2k} \text{ and } b = 0, \end{cases}$$

and

$$\nu(q_{s,w}, b, 0) = \lambda.$$

For $w \in \{0,1\}^{2k}$, if T is in state $q_{i,w}$ (the just printing states without reversing) or in state q_1 (where T checks if the next zone is just printing or printing and reversing) and sees $\vdash$ indicating the right hand side of the tape, T enters q_F the final state and halts, printing nothing. That is for $w \in \{0,1\}^{2k}$

$$\delta(q_{i,w}, \vdash, 0) = \delta(q_1, \vdash, 0) = (q_F, -1),$$

and

$$\nu(q_{i,w}, \vdash, 0) = \nu(q_1, \vdash, 0) = \lambda.$$

This completes the construction of T.

□

Corollary 6.3.13. *There exists a non-normal PB-deep sequence.*

Proof. This follows from Theorem 6.3.11 since the string $01^{2k}0$ only occurs as a substring of the constructed sequence S which satisfies the theorem a finite number of times. This is clear as the only places 01^{2k} can occur is if 0 is the last bit of S_{k-1} or where the 1^{2k} is a prefix to a flag in some zone S_n. However, as the flags increase in length, 01^{2k} will eventually always be followed by another 1.

□

6.3.4 Preliminary Comparison with Pushdown Depth

In this chapter we do not present an example of a sequence which is PB-deep but not PD-deep. More work is to be done to find such a sequence if it exists. Instead we present a preliminary result which states that for all $0 < \beta < 1/2$, one can construct a sequence S such that PB-depth$(S) \geq 1 - \beta$ while PD-depth$(S) \geq 1/2 - \beta$. Hence, the sequence is deep in both notions, and it is possible that their depth levels are in fact equal. However, we show the sequence is not FS-deep.

The sequence is composed of strings of the form $R^{|R|}F(R^{-1})^{|R|}$ where F is a flag and R is a string not containing F with large plain Kolmogorov complexity relative to its length. Note that $R^{|R|}$ is a string of length $|R|^2$. From a single description of R, a 1-pebble transducer can use a single pebble to print $R^{|R|}$. A large ILPDC with no restriction on its stack can be built to push $R^{|R|}$ onto its stack, and then when it sees the flag F, use its stack to compress $(R^{-1})^{|R|}$. These R are built such that an ILUPDC is unable to use its stack to compress R, resulting in minimal compression. For FSTs, the sequence appears almost random and so little depth is achieved.

Note that the sequence is similar to that in Theorem 4.4.3, however the successive repetitions R allows a pebble-transducer to gain an advantage neither the

ILPDC we examine nor an FST have.

Remark 6.3.14. For all $0 < \beta < 1/2$, there exists a sequence S such that PB-depth$(S) \geq 1 - \beta$, PD-depth$(S) \geq 1/2 - \beta$, and FS-depth$(S) < \beta$.

Proof. The sequence S which we construct that satisfies this remark is similar to the sequence constructed in Theorem 4.4.3.

Let $0 < \beta < 1/2$ and let $k > 8$ be such that $\beta \geq 8/k$. For each n, let $t_n = k^{\lceil \frac{\log n}{\log k} \rceil}$. Note that for all n,

$$n \leq t_n \leq kn. \tag{6.30}$$

Consider the set T_j which contains all strings of length j that do not contain 1^k as a substring. As T_j contains strings of the form $x_1 0 x_2 0 x_3 0 \cdots$ where each x_t is a string of length $k-1$, we have that $|T_j| \geq 2^{j(1-\frac{1}{k})}$. For each j, let $R_j \in \{0,1\}^{kt_j}$ have maximal plain Kolmogorov complexity in the sense that

$$K(R_j) \geq |R_j|(1 - \frac{1}{k}). \tag{6.31}$$

Such an R_j exists as $|T_{|R_j|}| > 2^{|R_j|(1-\frac{1}{k})} - 1$. Note that $kj \leq |R_j| \leq k^2 j$. We construct S in stages $S = S_1 S_2 \ldots$ where for each j,

$$S_j = R_j^{|R_j|} 1^k (R_j^{-1})^{|R_j|}.$$

Claim 6.3.15. PD-depth$(S) \geq \frac{1}{2} - \beta$.

First we examine how well any ILUPDC compresses occurrences of R_j zones in S. Let $C \in$ ILUPDC. Consider the tuple

$$(\widehat{C}, q_s, q_e, z, \nu_C(q_s, R_j, z))$$

where $\widehat{C}$ is an encoding of C, q_s is the state that C begins reading R_j in, q_e is the state C ends up in after reading R_j, z is the stack contents of C as it begins reading R_j in q_s (i.e. $z = 0^p z_0$ for some p), and the output $\nu_C(q_s, R_j, z)$ of C on R_j. By Remark 4.2.5, C's stack is only important if $|z| < (c+1)|R_j|$, as if $|z|$ is larger, C will output the same regardless of $|z|$'s true value. Hence, setting

$$z' = \begin{cases} |z| & \text{if } |z| < (c+1)|R_j| \\ (c+1)|R_j| & \text{if } |z| \geq (c+1)|R_j|, \end{cases} \tag{6.32}$$

as C is lossless, having knowledge of the tuple $(\widehat{C}, q_s, q_e, z', \nu_C(q_s, R_j, z))$ means we can recover R_j. If we encode the tuple $(\widehat{C}, q_s, q_e, z', \nu_C(q_s, R_j, z))$ the same way as in (4.9), and noting that z' contributes roughly $O(\log |R_j|)$ bits to the encoding, we have we have by Equation (6.31) that

$$|R_j|(1 - \frac{1}{k}) \leq K(R_j) \leq |\nu_C(q_s, R_j, z)| + O(\log |R_j|) + O(|\widehat{C}|) + O(1). \tag{6.33}$$

Therefore, for j large we have

$$|\nu_C(q_s, R_j, z)| \geq |R_j|(1 - \frac{1}{k}) - O(\log |R_j|) > |R_j|(1 - \frac{2}{k}) \tag{6.34}$$

This is similarly true for R_j^{-1} zones also as $K(R_j) \leq K(R_j^{-1}) + O(1)$. Hence for j large we see that C outputs at least

$$\begin{aligned} |C(\overline{S_j})| - |C(\overline{S_{j-1}})| &\geq 2|R_j|^2(1 - \frac{2}{k}) \\ &= (|S_j| - k)(1 - \frac{2}{k}) \\ &\geq |S_j|(1 - \frac{3}{k}) \end{aligned} \tag{6.35}$$

bits when reading S_j.

Next we examine how well C compresses S on arbitrary prefixes. Consider the prefix $S \upharpoonright n$ and let j be such that $\overline{S_j}$ is a prefix of $S \upharpoonright n$ but $\overline{S_{j+1}}$ is not. Thus $S \upharpoonright n = \overline{S_j} \cdot y$ for some $y \sqsubset S_{j+1}$. Suppose Equation (6.35) holds for all $i \geq j'$. Hence we have that

$$\begin{aligned}
|C(S \upharpoonright n)| &\geq |C(\overline{S_j})| \geq |C(\overline{S_j})| - |C(\overline{S_{j'-1}})| \\
&\geq |S_{j'} \dots S_j|(1 - \frac{3}{k}) - O(1) && \text{(by (6.35))} \\
&= (n - |y| - |\overline{S_{j'-1}}|(1 - \frac{3}{k}) - O(1) \\
&\geq (n - |y|)(1 - \frac{4}{k}). && (6.36)
\end{aligned}$$

Then, noting that $n = \Omega(j^3)$ and that $|y| = O(j^2)$, by Equation (6.36) we have that

$$|C(S \upharpoonright n)| \geq n(1 - \frac{5}{k}). \tag{6.37}$$

As C was arbitrary, we therefore have that

$$\rho_{\text{UPD}}(S) > 1 - \frac{6}{k}. \tag{6.38}$$

Next we examine how well the ILPDC C' from the proof of Theorem 4.4.3 can compress prefixes of S. Recall that C' outputs its input for some prefix $S_1 \dots S_i$. Then, for all $j > i$, C' compresses S_j as follows: On S_j, C' outputs its input on $R_j^{|R_j|}1^k$ while trying to identify the 1^k flag. Once the flag is found, C' pops the flag from its stack and then begins to read an $(R_j^{-1})^{|R_j|}$ zone. On $(R_j^{-1})^{|R_j|}$, C' counts modulo v to output a zero every v bits, and uses its stack to ensure that the input is indeed $(R_j^{-1})^{|R_j|}$. If this fails, C' outputs an error flag and enters an error state and from then on outputs its input. Furthermore, v is cleverly chosen such that for all but finitely many j, v divides evenly in $|R_j|$. Specifically we set $v = k^a$ for some $a \in \mathbb{N}$. A complete description of C' was provided at the end of

the proof of Theorem 4.4.3 on page 84.

Next we will compute the compression ratio of C' on S. We let p be such that for all $j \geq p$, v divides evenly into $|R_j|$. C' will output its input on $\overline{S_p}$ and begin compressing on the succeeding zones. Also, note that the compression ratio of C' on S is largest on prefixes ending with a flag 1^k. Hence, consider some prefix $\overline{S_{j-1}}R_j^{|R_j|}1^k$ of S. We have that for n sufficiently large

$$\begin{aligned}
\frac{|C(\overline{S_{j-1}}R_j^{|R_j|}1^k)|}{|\overline{S_{j-1}}R_j^{|R_j|}1^k|} &\leq \frac{|\overline{S_{p-1}}| + \sum_{i=p}^{j}(|R_i|^2 + k + \frac{|R_i|^2}{v}) - \frac{|R_j|^2}{k}}{|\overline{S_{j-1}}R_j^{|R_j|}1^k|} \\
&\leq \frac{|\overline{S_{p-1}}|}{|\overline{S_{j-1}}|} + \frac{(1+\frac{1}{v})\sum_{i=1}^{j}(kt_i)^2 + jk - \frac{(kt_j)^2}{v}}{|\overline{S_{j-1}}|} \\
&\leq \frac{1}{6v} + \frac{(1+\frac{1}{v})\sum_{i=1}^{j}(kt_i)^2 + jk - \frac{(kt_j)^2}{v}}{2k^2\sum_{i=1}^{j-1}t_i^2} && \text{(for } j \text{ large)} \\
&\leq \frac{1}{6v} + \frac{(1+\frac{1}{v})\sum_{i=1}^{j-1}t_i^2}{2\sum_{i=1}^{j-1}t_i^2} + \frac{t_j^2}{2\sum_{i=1}^{j-1}t_i^2} + \frac{j}{2k\sum_{i=1}^{j-1}t_i^2} \\
&\leq \frac{1}{6v} + \frac{1}{2} + \frac{1}{2v} + \frac{3(jk)^2}{(j-1)(j)(2j+1)} + \frac{3}{k(j-1)(2j+1)} \\
&\leq \frac{1}{6v} + \frac{1}{2} + \frac{1}{2v} + \frac{1}{6v} + \frac{1}{6v} && \text{(for } j \text{ large)} \\
&= \frac{1}{2} + \frac{1}{v}. && (6.39)
\end{aligned}$$

As v can be chosen to be arbitrarily large, we therefore have that

$$R_{\mathrm{PD}}(S) \leq \frac{1}{2}. \tag{6.40}$$

Hence, for n large, by Equations (6.38) and (6.40) it follows that for all $C \in$ ILUPDC

$$\begin{aligned}
|C(S \upharpoonright n)| - |C'(S \upharpoonright n)| &\geq (1 - \frac{6}{k} - \frac{1}{k})n - (\frac{1}{2} + \frac{1}{k})n && (6.41) \\
&= (\frac{1}{2} - \frac{8}{k}). && (6.42)
\end{aligned}$$

Hence, choosing k large such that $\frac{8}{k} \leq \beta$ gives us our desired result of PD-depth$(S) \geq \frac{1}{2} - \beta$.

Claim 6.3.16. FS-depth$(S) < \beta$.

Next we examine the finite-state depth of S. From Equation (4.22) and Definition 3.2.6, it follows that $\dim_{\mathrm{FS}}(S) > 1 - \frac{6}{k}$. Hence, for all l it follows that for all but finitely many n

$$D^l_{\mathrm{FS}}(S \upharpoonright n) \geq (1 - \frac{7}{k})n. \tag{6.43}$$

Therefore, for l' such that $I_{\mathrm{FS}} \in \mathrm{FST}^{\leq l'}$ we have for all l and almost every n that

$$D^{l'}_{\mathrm{FS}}(S \upharpoonright n) - D^l_{\mathrm{FS}}(S \upharpoonright n) \leq n - (1 - \frac{7}{k})n < \frac{8}{k} \cdot n. \tag{6.44}$$

That is, FS-depth$(S) < \beta$ as desired.

Claim 6.3.17. PB-depth$(S) \geq 1 - \beta$.

Finally we examine the pebble depth of S. Consider the pebble-transducer T that reads its input the following way: T reads its input in chunks of size 2 trying to find flags of uneven bits. If T reads a chunk 10 in its input, T then scans right continuing to read its input in chunks of size two until it finds two unequal bits. T uses the two flags and the string of the form $d(x)$ between the flags to print the string $x^{|x|}$. That is, if T reads an input with the substring $10d(x)b_1b_2$, with $b_1, b_2 \in \{0,1\}$, $b_1 \neq b_2$, and $x \in \{0,1\}^*$, then T outputs $x^{|x|}$ on that substring. If instead T reads the chunk 01, then T reads its input in chunks of size 2, outputting a single bit from each chunk if the bits match until it sees an unequal chunk or it reaches the end of the tape. That is, if T reads an input with the substring $01d(x)b_1b_2$, with $b_1, b_2 \in \{0,1\}$, $b_1 \neq b_2$, and $x \in \{0,1\}^*$, or the tape ends with $01d(x) \vdash$, then T outputs x. T enters its final state upon seeing $\vdash$ if the last flag it saw was 01, i.e. T must 'print' at least the empty string to

enter a final state. A full description of T is provided at the end of this proof.

Consider an arbitrary prefix $S \upharpoonright n$ of S. Let j be such that $\overline{S_{j-1}}$ is a prefix of $S \upharpoonright n$ but $\overline{S_j}$ is not. That is, $S \upharpoonright n = \overline{S_{j-1}} \cdot y$ for some $y \sqsubset S_j$. For each i, let x_i denote the string

$$x_i = 10 \cdot d(R_i) \cdot 01 \cdot d(1^k) \cdot 10 \cdot d(R_i^{-1}).$$

Hence we have that

$$T(x_1 \dots x_{j-1} 01 \cdot d(y)) = S \upharpoonright n.$$

Then, for all $\varepsilon > 0$, for n large it follows that

$$\begin{aligned}
\frac{D_{\mathrm{PB}}^{|T|}(S \upharpoonright n)}{n} &= \frac{\sum_{i=1}^{j-1} |x_i| + 2 + 2|y|}{|\overline{S_{j-1}}| + |y|} \\
&\leq \frac{4\sum_{i=1}^{j-1} |R_i| + (6+2k)(j-1) + 2 + 2|y|}{|\overline{S_{j-1}}|} \\
&\leq \frac{4\sum_{i=1}^{j-1} kt_i + (6+2k)(j-1) + 2 + 2|S_j|}{|\overline{S_{j-1}}|} \\
&\leq \frac{4k\sum_{i=1}^{j-1} t_i + (6+2k)(j-1) + 2 + 2k + 4(kt_j)^2}{2k^2 \sum_{i=1}^{j-1} t_i^2} \\
&\leq \frac{4k^2 \sum_{i=1}^{j-1} i}{2k^2 \sum_{i=1}^{j-1} i^2} + \frac{(6+2k)(j-1) + 2(1+k)}{2k^2 \sum_{i=1}^{j-1} i^2} + \frac{4k^3 j^2}{2k^2 \sum_{i=1}^{j-1} i^2} \\
&= \frac{6}{2j-1} + \frac{(6+2k)(j-1) + 2(1+k)}{(j-1)(j)(2j-1)/6} + \frac{12kj^2}{(j-1)(j)(2j-1)} \\
&\leq \varepsilon. \qquad \text{(for } j \text{ large)}
\end{aligned}$$

Hence we have that

$$R_{\mathrm{PB}}(S) = 0. \tag{6.45}$$

Therefore, by Equations (6.43) and (6.45), for all k and almost every n we

have that

$$D^{k}_{\mathrm{FS}}(S \upharpoonright n) - D^{|T|}_{\mathrm{PB}}(S \upharpoonright n) \geq (1 - \frac{7}{k})n - \frac{1}{k}n \geq 1 - \beta. \tag{6.46}$$

That is, PB-depth$(S) \geq 1 - \beta$ as desired.

For completeness, the following is a description of the pebble transducer T. Note that some transitions are omitted if there are never seen. Let Q be the following set of states of T:

1. the start state q_s,
2. the accepting state q_a,
3. the failure state q_d,
4. the initial flag identifying states q_s^i, q_s^0 and q_s^1,
5. the 'just print' states q_p, q_p^0 and q_p^1,
6. the 'place pebble' states q_r^i, q_r^0, q_r^1,
7. the states used to find a flag when scanning left q_l, q_l^0 and q_l^1,
8. the 'print square' states q_i^{-1}, q_i, q_i^0 and q_i^1,
9. the states used to find and pop the pebble from the tape q_f and q_f'.

We first describe the transition function δ of T. Beginning in the start state, T checks whether the next two bits contain 01 or 10 to indicate whether it is entering a *print* or *print-square* zone respectively. T first moves right off of $\dashv$

$$\delta(q_s, \dashv, 0) = (q_s^i, +1).$$

In q^i, T reads what is under its head and moves right to check the next bit. That is,

$$\delta(q_s^i, b, c) = \begin{cases} (q_d, +1) & \text{if } b = \dashv \\ (q_d, -1) & \text{if } b = \vdash \\ (q_s^0, +1) & \text{if } b = 0 \\ (q_s^1, +1) & \text{if } b = 1. \end{cases}$$

T then checks if the next bit is different from the previous bit, i.e. if a flag has just been read. If they are the same or the end of the tape has been read, T enters the failure state. That is,

$$\delta(q_s^b, b', c) = \begin{cases} (q_d, -1) & \text{if } b = \vdash \text{ or } b' = b \\ (q_p, +1) & \text{if } bb' = 01 \\ (q_r, +1) & \text{if } bb' = 10. \end{cases}$$

If the flag read was 01, T enters the 'just print' states beginning with state q_p. Here, T reads its input in chunks of size two. T scans left until it sees a chunk of two unmatching bits, that is, another flag and enters the appropriate state. If it reaches the right end of the tape, it enters the final state. That is, beginning in state q_p, T reads the first bit of a chunk

$$\delta(q_p, b, c) = \begin{cases} (q_p^b, +1) & \text{if } b \in \{0, 1\} \\ (q_a, -1) & \text{if } b = \vdash. \end{cases}$$

Then in state q_p^b, if the next bit read matches b, T enters state q_p again, otherwise

it knows it has just read a flag. That is

$$\delta(q_p^b, b', c) = \begin{cases} (q_p, +1) & \text{if } b = b' \\ (q_p, +1) & \text{if } bb' = 01 \\ (q_r, +1) & \text{if } bb' = 10 \\ (q_a, -1) & \text{if } b = \vdash. \end{cases}$$

If T has read the flag 10, it enters the 'print square' zone. T must first place its pebble on its tape. Starting in state q_r, T reads its input and then moves to the right checking if the two bits it has just read match. If they match, T places its pebble on the tape, otherwise it knows it has just read another flag. That is

$$\delta(q_r, b, 0) = \begin{cases} (q_d, -1) & \text{if } b = \vdash \\ (q_r^0, +1) & \text{if } b = 0 \\ (q_r^1, +1) & \text{if } b = 1, \end{cases}$$

and

$$\delta(q_r^b, b', 0) = \begin{cases} (q_d, -1) & \text{if } b' = \vdash \\ (q_l, \text{push}) & \text{if } b = b' \\ (q_r, +1) & \text{if } bb' = 10 \\ (q_p, +1) & \text{if } bb' = 01. \end{cases}$$

Once the pebble is placed, beginning in state q_l, T scans left while reading in chunks of size two to find the last 10 flag it has read. That is,

$$\delta(q_l, b, c) = (q_l^b, -1)$$

and

$$\delta(q_l^b, b', c) = \begin{cases} (q_l, -1) & \text{if } b = b' \\ (q_i^{-1}, +1) & \text{if } b'b = 10 \\ (q_d, +1) & \text{otherwise.} \end{cases}$$

Once the 10 flag is found, beginning in state q_i^{-1}, T moves to the right

$$\delta(q_i^{-1}, b, c) = (q_i, +1).$$

Using states q_i, q_i^0 and q_i^1, T scans right reading in chunks of size two trying to find the next flag. That is

$$\delta(q_i, b, c) = \begin{cases} (q_d, -1) & \text{if } b = \vdash \\ (q_i^b, +1) & \text{if } b \in \{0, 1\}, \end{cases}$$

and

$$\delta(q_i^b, b', c) = \begin{cases} (q_d, -1) & \text{if } b = \vdash \\ (q_i, +1) & \text{if } b = b' \\ (q_f, -1) & \text{if } b \neq b'. \end{cases}$$

In state q_f, T has just read a flag. T then scans left to find its pebble on its tape to pop it. That is,

$$\delta(q_f, b, c) = \begin{cases} (q_d, +1) & \text{if } b = \vdash \\ (q_f, -1) & \text{if } c = 0 \\ (q_f', \text{pop}) & \text{if } c = 1. \end{cases}$$

In state q'_f, T moves right and re-enters state q_r to place a pebble on its tape. That is,

$$\delta(q_f, b, c) = (q_r, +1).$$

In the failure state q_d, T enters a loop and so never enters q_a. That is

$$\delta(q_d, b, c) = \begin{cases} (q_d, -1) & \text{if } b = \vdash \\ (q_d, +1) & \text{otherwise.} \end{cases}$$

T outputs nothing on all transitions except in the following two cases:

- $\nu(p_b, b, c) = b$ (when in a 'just print' state and it sees an equal chunk)
- $\nu(q_i^b, b, c) = b$ (when in a 'print square' zone and it sees an equal block)

This completes the construction of T.

□

6.4 Discussion

6.4.1 Why not Compressors?

Unlike in Chapters 4 and 5 where compressors were used to define pushdown and Lempel-Ziv depth, for pebble depth we took the minimal description decompression approach. The main reason for this is that the power of pebble-transducers comes from their ability to compute string to string functions that other types of transducers cannot, e.g. the pref and the pow_k functions.

For pebble compressors to be of interest, as opposed to ordinary two-way finite-state compressors, one would hope that on all inputs of the form $x^{|x|}$ for instance, a 1-pebble compressor exists which would output x. However, this compressor would recognise the non-regular language of $L = \{x^{|x|} : x \in \{0,1\}^*\}$ contradicting that pebble-automata recognise only the regular languages [74].

The main issue is as pebble-automata are two-way, each position of the input can be visited multiple times which introduces a problem of defining what compression by a pebble-compressor even means. When considering the output's length, one could consider the length up to the first instance of a visit to a position of the input, or the last visit of a position, or any other variation. This is explored by Carton and Heiber in [35]. They demonstrate how for ordinary two-way finite-state compressors (0-pebble compressors) these possible definitions are equivalent on normal inputs. However, if an additional counter is added to the machine, that is a stack whose stack alphabet has only one character, the definition of compressibility collapses with the counter providing unbounded memory. A k-pebble transducer only has access to k extra bits of information at any one time compared to a finite-state transducer. Hence its memory is bounded. However compressibility still collapses as shall be demonstrated next.

The following adapts Carton and Heiber's approach of compressibility for 0-pebble compressors to k-pebble compressors [35].

Definition 6.4.1. Let $f_n = \min\{j \,|\, i_j = n\}$ and $l_n = \max\{j \,|\, i_j = n\}$ be the first and last visit of position n in the input of the accepting run of an information lossless k-pebble compressor whose sequence of configurations is given by

$$r = (q_0, 0, \bot^k, \lambda) \to (q_1, i_1, b_1, v_1) \to (q_2, i_2, b_2, v_1v_2) \to (q_3, i_3, b_3, v_1v_2v_3) \cdots$$

The *first-hit*, *middle* and *last-hit outputs* of the compressor at position n of its input are respectively given by

$$\sum_{j \le f_n} |v_{j+1}|, \sum_{i_j \le n} |v_{j+1}|, \text{ and } \sum_{j \le l_n} |v_{j+1}|.$$

Similarly for a 0-pebble compressor, where $f_n = \min\{j \,|\, i_j = n\}$ and $l_n =$

$\max\{j|\, i_j = n\}$ on an accepting run whose sequence of configurations is given by

$$r = (q_0, 0, \bot, \lambda) \to (q_1, i_1, \bot, v_1) \to (q_2, i_2, \bot, v_1v_2) \to (q_3, i_3, \bot, v_1v_2v_3) \cdots$$

Carton and Heiber then define the *first-hit*, *middle* and *last-hit ratios* at position n are respectively given by

$$\liminf_{n\to\infty} \frac{1}{n} \sum_{j \le f_n} |v_{j+1}|,\ \liminf_{n\to\infty} \frac{1}{n} \sum_{i_j \le n} |v_{j+1}|,\ \text{ and } \liminf_{n\to\infty} \frac{1}{n} \sum_{j \le l_n} |v_{j+1}|.$$

In Theorem 3.2 of [35], it is shown that these three ratios coincide on normal inputs.

The above characterizations of the three ratios can be adapted to k-pebble transducers also. As an input of size n's output has size $O(n^{k+1})$, the above ratios may be adapted by dividing by $\frac{1}{n^{k+1}}$ instead of $\frac{1}{n}$. Regardless of whether $\frac{1}{n^{k+1}}$ or $\frac{1}{n}$ is used, we next demonstrate these ratios are not equivalent for certain k-pebble compressors. This is shown by the existence of a 1-pebble compressor such that for longer prefixes of any sequence, the middle and last hit ratio tend to infinity while the first hit ratio also tends to infinity (if we divide by $\frac{1}{n}$) and to $1/2$ if we divide by $\frac{1}{n^2}$.

Consider the 1-pebble T compressor which computes the function pref. Beginning on $\dashv$, T moves one square to the right and places its pebble. T then scans left to $\dashv$ printing nothing, and then scans right printing every bit it reads up until (and including) it sees the square with the pebble. T then lifts the pebble and moves it one square to the right and scans left again repeating the process. T enters an accepting state when it reaches the right end of the tape. Since $|\text{pref}(x)|$ is a triangular number, $|x|$ can be retrieved from knowing $|\text{pref}(x)|$. Hence T is lossless as then x will equal the suffix of $\text{pref}(x)$ of length $|x|$.

Let $S \in \{0,1\}^\omega$. Note for all m, $T(S \upharpoonright m) \sqsubset T(S \upharpoonright (m+1))$. We examine the

first-hit, middle and last hit ratio of character $S[n-1]$ (the n^{th} bit) of S when T is given the input $S \upharpoonright m$, where $m > n$.

The first time T reads $S[n-1]$ it has already printed out $\text{pref}(S \upharpoonright (n-1))$. T also prints nothing during its first time reading $S[n-1]$ as it only places its pebble on the square. Hence the first hit output is simply

$$\sum_{j=1}^{n-1} j = \frac{(n-1)n}{2}. \tag{6.47}$$

For the middle output, T prints $\text{pref}(S \upharpoonright n)$ and $S \upharpoonright n$ a further $(m-n)$ times. That is, the middle output is given by

$$n(m-n) + \sum_{j=1}^{n} j = n(m-n) + \frac{n(n+1)}{2}. \tag{6.48}$$

For the last-hit output, T prints $\text{pref}(S \upharpoonright (m-1))$ followed by $S \upharpoonright n$. That is, the last-hit output is given by

$$n + \sum_{j=1}^{m-1} j = n + \frac{(m-1)m}{2}. \tag{6.49}$$

Hence as we take longer prefixes of S, position n is visited infinitely many times resulting in m tending to infinity. Thus the middle and last hit ratios cannot be defined. The first-hit ratio similarly cannot be defined if we divide by $1/n$ while if we divide by $1/n^2$ it tends to $1/2$.

6.4.2 Why not Pebble vs Pebble?

The notion of pebble depth discussed in this chapter is based on the difference in minimal descriptional complexity of finite-state transducers and pebble transducers. Ideally a notion solely comparing pebble transducers would be examined (as in Chapter 3 where finite-state transducers are compared against themselves).

One of the main obstacles to this approach is finding a result analogous to Lemma 3.3.9. Ideally we would like something similar to being able to say that for all strings $x, y \in \{0,1\}^*$ and for all k, there exists a k' such that

$$D^k_{\mathrm{PB}}(xy) \geq D^{k'}_{\mathrm{PB}}(x) + D^{k'}_{\mathrm{PB}}(y). \tag{6.50}$$

In the finite-state approach, if there were two strings p and q and an FST T such that $T(p) = x$ and $T(pq) = xy$, we were able to build an FST T' identical to T with only the start state possibly being different such that $T'(q) = y$.

This approach would not work for pebble transducers. For instance, suppose N is a pebble transducer and $r \in \{0,1\}^*$ such that $N(r) = xy$. Consider the split $r = st$ where N's reading head is above the first character of t immediately after it has finished outputting x. Simply building a new pebble transducer N' identical to N but switching the start state to be the one that N is in after outputting x does not aid us as before since we cannot say that $N'(t) = y$. Due to the two-way nature of N, N may scan left and read suffixes of s as part of its process of outputting y. N' on t would not have this information available to it. Similarly the placement of N's pebbles on its tape after outputting x may play an important role in the process of outputting y. This is a problem as N' would not have any pebbles placed on its tape at the beginning of its computation of t.

This is best illustrated with the following example. Suppose k is large enough to contain the pebble-transducer R where $R(x) = xx^{-1}$ for all x. Let r be a string such that $K(r) = |r|$. Therefore, $K(r^{-1}) = |r| + O(1)$ as rev is an injective function. Similarly note that $K(r) \leq D^{k'}_{\mathrm{PB}}(r) + O(k')$ and $K(r^{-1}) \leq D^{k'}_{\mathrm{PB}}(r^{-1}) + O(k')$ for all k'. Hence, if we had a result such as in Equation (6.50) we would have that

$$|r| \geq D^k_{\mathrm{PB}}(rr^{-1}) \geq D^{k'}_{\mathrm{PB}}(r) + D^{k'}_{\mathrm{PB}}(r^{-1}) \geq 2|r| - O(k')$$

which is a contradiction for long enough r.

However, as with other notions of depth such as Bennett's which compare one form of descriptional complexity against a weaker notion, we are content with the pebble depth version we have provided.

6.5 Summary

In this chapter we presented a new notion of depth based on pebble-transducers. We showed that PB-depth satisfies versions of the fundamental properties of depth. Specifically, we first showed that FST-trivial and PB-incompressible sequences are not PB-deep in Theorem 6.3.5. We demonstrated a slow growth type law holds in Theorem 6.3.6. We differentiated PB-depth from FS-depth by showing the existence of a normal PB-deep sequence in Theorem 6.3.10. We also demonstrated the existence of non-normal PB-deep sequences in Corollary 6.3.13 via Theorem 6.3.11 which, if they are LZ-deep, have low LZ-depth in Theorem 6.3.11. A preliminary comparison with pushdown depth was also performed in Remark 6.3.14.

Chapter 7

Prediction by Partial Matching and Normal Sequences

Contents of this chapter were presented (virtually) at SOFSEM 2021 in Italy during January 2021. doi: 10.1007/978-3-030-67731-2_28.

7.1 Introduction

In previous chapters we have used compression algorithms to examine the complexity of sequences (pushdown compressors, finite-state compressors and the Lempel-Ziv 78 algorithm) and have occasionally focused specifically on the complexity of *normal* sequences (Theorems 3.3.5, 5.4.1 and 6.3.10). The following chapter continues this line of study, away from defining new logical depth notions, by analysing the performance of members of the Prediction by Partial Matching (PPM) family of compressors (PPM* [46] and the original Bounded PPM algorithm [45]). As discussed in Chapter 1, a common question studied about normal sequences is whether or not they are compressible by certain families of compressors. Such an analysis has not been performed previously on the PPM algorithms. The main objective of this chapter is to show the existence of a normal sequence

which PPM* can compress but is Lempel-Ziv 78 incompressible. We also show that Bounded PPM cannot compress normal sequences. We briefly compare the performance of the bounded and unbounded versions of PPM of specific inputs also.

7.2 Description of the PPM Algorithms

The PPM family of compression algorithms was first introduced by Cleary and Witten in 1984 [45]. PPM algorithms work by building a model of their input as it reads each character. The model keeps track of previously seen substrings of the input, known as *contexts*, and the characters that follow them. Characters of the input are encoded based on their *prediction probabilities* in the relevant contexts via arithmetic encoding [149]. After a character is read, the model is updated by incrementing the frequency counts of the character in the relevant contexts and, if needed, adding new contexts to the model.

Implementations of the PPM algorithm family often rely on the *exclusion principal* to achieve better compression ratios. We ignore this is in our analysis of the algorithms for simplicity as, even without this, the normal sequence we later build achieves a compression ratio of 0 via PPM*.

7.2.1 Bounded PPM

In the original presentation of PPM, a bounded version was introduced [45]. Prior to encoding, a value $t \in \mathbb{N}$ must be provided to the encoder which sets the maximum *context* length that the model can store. We refer to this version of Bounded PPM with bound t as PPM_t. By context, we mean previously seen substrings of the input stream contained in the model. For each context, the model records what characters have followed the context in the input stream, and the frequency with which each character has occurred. These frequencies are used to build *pre-*

diction probabilities that the encoder uses to encode the rest of the input stream. When reading the next character of the input stream, the encoder examines the *relevant* contexts and encodes the next character based on its current prediction probabilities in these contexts. By relevant context, we mean suffixes of the input already read by the encoder that are contained in the model. The longest relevant context available is chosen as the first *current* context, as it is the one the model uses to first encode the next character seen. Once encoded, new contexts are added to the model if necessary, and the prediction probabilities of the relevant contexts are updated to reflect the character that has just been read.

In some implementations of PPM algorithms (such as Moffat's [111]), only the prediction probabilities of the context used to encode the character read and all longer relevant contexts are updated. For instance if the context of length two b_1b_2 is used to encode the character b_3, only the counts of contexts with suffix b_1b_2 are updated while the contexts b_2 and λ are not. This type of updating is referred to as *shallow* updating or *update exclusion*. We do not use this method of updating the counts and instead take the approach of Cleary and Witten's original approach of updating the counts of all relevant contexts.

When a character being encoded has never occurred previously in the current context, an *escape* symbol (denoted by \$) is transmitted and the next shortest relevant context becomes the new current context. If the character has not been seen before even when the current context is λ, that is, the context where none of the previous characters are used to predict the next character, an escape is outputted and the character is assigned the prediction probability from the order-(-1) table. By convention, this table assigns each character equal probability.

Several different approaches exist to calculate the prediction probabilities. In their original paper, Cleary and Witten present two approaches known as Method A and Method B [45]. Moffat proposed Method C in 1990 [111]. Howard and Vitter present Method D in 1992 in [79]. Åberg et al. in 1997 presented Method E

ctxt	pred	cnt	pb	ctxt	pred	cnt	pb	ctxt	pred	cnt	pb
Order $t = 3$				Order $t = 2$				Order $t = 1$			
001	1	1	$\frac{1}{2}$	00	1	1	$\frac{1}{2}$	0	0	1	$\frac{1}{6}$
	\$	1	$\frac{1}{2}$		\$	1	$\frac{1}{2}$		1	3	$\frac{1}{2}$
010	0	1	$\frac{1}{2}$	01	0	1	$\frac{1}{5}$		\$	2	$\frac{1}{3}$
	\$	1	$\frac{1}{2}$		1	2	$\frac{2}{5}$	1	0	3	$\frac{3}{7}$
011	0	2	$\frac{2}{3}$		\$	2	$\frac{2}{5}$		1	2	$\frac{2}{7}$
	\$	1	$\frac{1}{3}$	10	0	1	$\frac{1}{4}$		\$	2	$\frac{2}{7}$
100	1	1	$\frac{1}{2}$		1	1	$\frac{1}{4}$	Order $t = 0$			
	\$	1	$\frac{1}{2}$		\$	2	$\frac{1}{2}$	λ	0	5	$\frac{5}{12}$
101	1	1	$\frac{1}{2}$	11	0	2	$\frac{2}{3}$		1	5	$\frac{5}{12}$
	\$	1	$\frac{1}{2}$		\$	1	$\frac{1}{3}$		\$	2	$\frac{1}{6}$
110	1	1	$\frac{1}{2}$					Order $t = -1$			
	\$	1	$\frac{1}{2}$						0	1	$\frac{1}{2}$
									1	1	$\frac{1}{2}$

Table 7.1: PPM_3 model for the input 0100110110 for the binary alphabet.

[18]. Steinruecken presented Method G in 2014 [140]. Witten and Bell described Methods P and X based on Poisson processes in 1991 [148]. Bloom's Method Z approach uses a secondary model to calculate escape probabilities [21].

This chapter uses Method C to assign probabilities to \$ in each context as it is the approach taken by Cleary and Witten when introducing PPM^*. Here, \$ is given a frequency equal to the number of distinct characters predicted in the context so far. Hence for the binary alphabet, \$'s count is bounded above by 2.

For instance, in Table 7.1 the model for the string 0100110110 with bound $t = 3$ is given. In the context 01, the escape symbol \$ has count 2 as both 0 and 1 have followed 01, while \$ in 101 has count 1 as it has only been followed by a 1.

Suppose 0 is the next character to be encoded after input stream 0100110110 by PPM_3. The relevant contexts are $110, 10, 0$ and λ. The longest relevant context is 110. The encoder escapes to the shorter context 10 since 0 is not seen in context 110. The escape is encoded by the prediction probability 1/2. From context 10, 0 is encoded with probability 1/4. The frequency counts of 0 will be updated

in contexts 10, 0 and λ. Also, 0 will be added as a prediction to context 110. Following this, if the next character to be encoded was another 0 the model would start in context 100 and, since 0 is not predicted here, it would transmit an escape symbol with probability 1/2 and then examine the next longest context 00 and proceed as necessary. If there was another bit b in the input stream after this, (as 000 would be the current suffix of the input stream but no context for 000 exists yet as it would have never been seen before) the encoder would begin in context 00 and proceed as before, and a context for 000 would be created predicting the character b when the model updates.

7.2.2 PPM*

Like PPM_t, PPM^* builds a model of contexts of its input, continuously updates the model, and encodes each character it sees based on its frequency probability in the current context. The key difference is that there is no upper bound on the maximum context length stored in the model. Instead of building a context for every substring seen, a context is only extended until it is unique. Suppose PPM^* has read the string x. For any string w with $\text{occ}(w, x) \geq 2$, the context wb must be built in the model for each b such that $\text{occ}(wb, x) \geq 1$. When reading a new character to encode, unlike PPM_t, PPM^* first chooses the shortest *deterministic* context available to be the current context. By deterministic, we mean a context where \$ has a frequency of 1, i.e. it has only been followed by a single character previously in the input stream. If no such context exists, the longest relevant context is chosen. We also use the Method C approach to compute \$ probabilities for PPM^*.

For instance, in Table 7.2 the model for the string $s = 0100110110$ is found when restricted to the binary alphabet. Suppose the next character read is a 0. The relevant contexts are $\lambda, 0, 10, 110$ and 0110. The shortest deterministic

ctxt	pred	cnt	pb	ctxt	pred	cnt	pb	ctxt	pred	cnt	pb
Order $t=5$				101	1	1	$\frac{1}{2}$	Order $t=1$			
01101	1	1	$\frac{1}{2}$		\$	1	$\frac{1}{2}$	0	0	1	$\frac{1}{6}$
	\$	1	$\frac{1}{2}$	110	1	1	$\frac{1}{2}$		1	3	$\frac{1}{2}$
Order $t=4$					\$	1	$\frac{1}{2}$		\$	2	$\frac{1}{3}$
0110	1	1	$\frac{1}{2}$	Order $t=2$				1	0	3	$\frac{3}{7}$
	\$	1	$\frac{1}{2}$	00	1	1	$\frac{1}{2}$		1	2	$\frac{2}{7}$
1101	1	1	$\frac{1}{2}$		\$	1	$\frac{1}{2}$		\$	2	$\frac{2}{7}$
	\$	1	$\frac{1}{2}$	01	0	1	$\frac{1}{5}$	Order $t=0$			
Order $t=3$					1	2	$\frac{2}{5}$	λ	0	5	$\frac{5}{12}$
010	0	1	$\frac{1}{2}$		\$	2	$\frac{2}{5}$		1	5	$\frac{5}{12}$
	\$	1	$\frac{1}{2}$	10	0	1	$\frac{1}{4}$		\$	2	$\frac{1}{6}$
011	0	2	$\frac{2}{3}$		1	1	$\frac{1}{4}$	Order $t=-1$			
	\$	1	$\frac{1}{3}$		\$	2	$\frac{1}{2}$		0	1	$\frac{1}{2}$
100	1	1	$\frac{1}{2}$	11	0	2	$\frac{2}{3}$		1	1	$\frac{1}{2}$
	\$	1	$\frac{1}{2}$		\$	1	$\frac{1}{3}$				

Table 7.2: PPM* model for the input 0100110110 for the binary alphabet.

context is 110. It does not predict 0 so \$ is transmitted with probability 1/2 and then 0 is transmitted by context 10 with probability 1/4. The model is updated as follows: The frequency count of 0 in contexts λ, 0 and 10 are incremented by one. 0 is added as a prediction to 110 and 0110. Furthermore $\text{occ}(00, s0) \neq 1$ and $\text{occ}(100, s0) \neq 1$ while $\text{occ}(00, s) = \text{occ}(100, s) = 1$. Therefore contexts 00 and 100 must be extended to create new contexts 001 and 1001 which predict a 1. If another 0 is read after $s0$, since both a 0 and 1 now have been seen to follow 110 and 0110, contexts for 1100 and 01100 will be created both predicting a 0, since a context has to be made for each branching path of 110 and 0110 (1101 and 01101 already exist).

7.2.3 Arithmetic Encoding

PPM's output is found via arithmetic encoding. A sample of works, including tutorials on arithmetic encoding, can be found in [15, 78, 80, 85, 149]. Beginning

with the interval $[0, 1)$, it is split into subintervals of lengths corresponding to the probabilities of the current context. The subinterval corresponding to the character or \$ transmitted is carried forward to the next stage. When complete, a number $c \in [a, b)$ is transmitted, where $[a, b)$ is the final interval. Note that c can be encoded in $-\lceil \log(|b-a|) \rceil$ characters. In particular, if $p_1, p_2, \ldots, p_m$ is the sequence of probabilities transmitted to encode the input, then $\prod_{i=1}^{m} p_i = |b-a|$. With knowledge of c and the length of the input, the input can be recovered.

For simplicity, we assume the encoder can calculate the endpoints of the intervals with infinite precision. In reality, a fixed finite limit precision is used to represent the intervals and their endpoints and a process known as *renormalisation* occurs to prevent intervals becoming too small to handle.

7.3 A Compressible Champernowne Sequence

Recall the Definition 2.5.5 for Champernowne sequences. These are sequences which are broken into substrings $C_1C_2C_3\ldots$ such that each zone C_i is a listing of all strings in $\{0,1\}^i$ exactly once. In this section we demonstrate our main result of the chapter in Theorem 7.3.8 which proves the existence of a Champernowne sequence C such that $R_{\mathrm{PPM}^*}(C) = 0$. Specifically, we show that a Champernowne sequence can be built using Pierce and Shields' algorithm from [119] which satisfies this.

7.3.1 Pierce and Shields' Construction

We now describe Pierce and Shields' construction of Champernowne sequences from [119] and write PSC to denote the set of sequences built using their method. It relies heavily on de Bruijn strings which were defined in Definition 2.6.1. Recall that a string $x \in \{0,1\}^{2^n}$ is a de Bruijn string of order n if for all $u \in \{0,1\}^n$ it holds that $\mathrm{occ}(u, x \cdot x[0..n-2]) = 1$.

Suppose we wished to construct substring C_n of a Champernowne sequence. Let d_n be a de Bruijn string of order n. For $0 \leq j \leq 2^n - 1$, let $d_{n,j}$ represent a cyclic shift to the left of the first j bits of d_n. That is, $d_{n,j} = d_n[j..] \cdot d_n[0..j-1]$. We write d_n instead of $d_{n,0}$ when no cyclic shift occurs. Note that each n can be written uniquely in the form $n = 2^s t$ where $s \geq 0$ and $t \geq 1$ are non-negative integers with t being odd. Each substring C_n is broken into further substrings $C_n = B_{n,0} \cdot B_{n,1} \cdots B_{n,2^s-1}$ where $B_{n,j}$ is a concatenation of $d_{n,j}$ with itself t times. That is, $B_{n,j} = d_{n,j}^t$. $B_{n,j}$ is referred to as the j^{th} *block* of C_n. Hence, if n is odd then $C_n = d_n^n$ and if $n = 2^k$ for $k \geq 1$ then $C_n = d_n d_{n,1} \cdots d_{n,n-1}$.

The following lemma demonstrates that any sequence in the set PSC is a Champernowne sequence. We re-present Pierce and Shields' proof using our own notation. It relies on some results from group theory of which details can be found in Rotman's introductory text [124].

Lemma 7.3.1 ([119]). *Let $C \in$ PSC. Then C is a Champernowne sequence.*

Proof. Let $C \in$ PSC. In order to show that C is a Champernowne sequence we must show that for each zone C_n, for all $x \in \{0,1\}^n$, $\text{occ}_b(x, C_n) = 1$.

Consider substring C_n. Let G_{2^n} be the cyclic group of order 2^n, i.e. $G_{2^n} = \langle x \,|x^{2^n} = e\rangle$, where $e = x^0$ is the identity element and x is the generator of the group. There exists a bijective mapping $f : G_{2^n} \to \{0,1\}^n$ such that for $0 \leq a < 2^n$, x^a is mapped to the substring of d_n of length n beginning in position a when d_n is viewed cyclically. That is, $f(e) = d_n[0..n-1]$, $f(x) = d_n[1..n], \ldots$ $f(x^{2^n-1}) = d_n[2^n - 1] \cdot d_n[0..n-2]$.

Let $s \geq 0$ and $t \geq 1$ where t is odd such that $n = 2^s t$. Consider the subgroup $\langle x^n \rangle$ of G_{2^n}. It follows that

$$|\langle x^n \rangle| = \frac{2^n}{\gcd(n, 2^n)} = 2^{2^s t - s} = 2^{n-s}.$$

So

$$\langle x^n \rangle = \bigcup_{i=0}^{2^{n-s}-1} \{x^{in \bmod 2^n}\} = \{e, x^n, x^{2n}, \ldots x^{(2^{n-s}-1)n \bmod 2^n}\}.$$

Concatenating the result of applying f to each element of $\langle x^n \rangle$ beginning with e in the natural order gives the string

$$\sigma = f(e) \cdot f(x^n) \cdot f(x^{2n}) \cdots f(x^{(2^{n-s}-1)n \bmod 2^n}).$$

σ can be thought of as beginning with the prefix of d_n of length n, cycling through d_n in blocks of size n until the block containing d_n's suffix of length n is seen. As $\frac{2^{n-s}n}{2^n} = t$, we have that $\sigma = d_n^t = B_0$.

As $|G_{2^n}|/|\langle x^n \rangle| = 2^s$, there are 2^s cosets of $\langle x^n \rangle$ in G_{2^n}. As cosets are disjoint, each represents a different set of 2^{n-s} strings of $\{0,1\}^n$. Specifically each coset represents one of the 2^s blocks of C_n, i.e. each coset represents the string $B_j = d_{n,j}^t$ for some j. Therefore, for each $x \in \{0,1\}^n$, for some $j \in \{0, \ldots, 2^s - 1\}$, $\mathrm{occ}_b(x, B_j) = 1$ and $\mathrm{occ}_b(x, B_i) = 0$ for each $i \neq j$. Thus $\mathrm{occ}_b(x, C_n) = 1$.

□

7.3.2 A Sequence which satisfies Theorem 7.3.8

Henceforth, we let $C \in \mathrm{PSC}$ denote the sequence such that for each n, the least lexicographic de Bruijn of order n was used to construct substring C_n. We furthermore also let d_n denote the least lexicographic de Bruijn of order n. We point the reader back to Section 2.6.1 to see how to construct these strings. We will show that C satisfies Theorem 7.3.8.

To help the reader visualise C, Figures 7.1 and 7.2 show the substrings C_3, C_4 and C_6 of C.

00010111	0000100110101111
00010111	0001001101011110
00010111	0010011010111100
	0100110101111000

Figure 7.1: Concatenating the three rows on the left hand side produces the substring C_3 and concatenating the four rows on the right hand side produces the substring C_4 if d_3 and d_4 are chosen to be the least lexicographic de Bruijn string of their order respectively.

000000100001100010100011100100101100110100111101010111011011111
000000100001100010100011100100101100110100111101010111011011111
000000100001100010100011100100101100110100111101010111011011111
000001000011000101000111001001011001101001111010101110110111110
000001000011000101000111001001011001101001111010101110110111110
000001000011000101000111001001011001101001111010101110110111110

Figure 7.2: Concatenating the six rows produces the substring C_6 where d_6 is chosen to be the lexicographic least de Bruijn string of order 6. The first three rows are B_0 while the second three rows are B_1.

Before we proceed, we make note of the following properties of d_n.

Remark 7.3.2. $d_1 = 01$, $d_2 = 0011$, and for $n \geq 3$,

1. $d_n[0..2n] = 0^n 10^{n-2}11$,

2. $d_n[2^n - n - 1..] = 1^n$.

The first property is clear when constructing the string. The second property is proven by Martin when showing how his algorithm to construct the string terminates [107].

For clarity, we write $\overline{C_n}$ to denote the prefix $C_1 \cdots C_n$ of C. Suppose the prefix $\overline{C_{n-1}}$ of C has already been read. While PPM*'s model may contain contexts of length n, the following lemma shows it will contain all possible contexts of length n after reading the first $2^n + n$ characters of C_n. The idea is that in the first $2^n + n - 1$ bits of C_n, for each $x \in \{0,1\}^{n-1}$, x occurs at least twice, and $x0$ and $x1$ occur once. Hence a context for each branching path of x must be created.

Lemma 7.3.3. *For $n > 2$, once the prefix $\overline{C_{n-1}}C_n[0..2^n + n - 1]$ is processed, the encoder will contain a context for all $x \in \{0,1\}^n$.*

Proof. Consider $x = C_n[0..2^n + (n-2)] = d_n \cdot 0^{n-1}$ (as d_n has prefix $0^n 10^{n-2}11$ by Remark 7.3.2). Let $v \in \{0,1\}^{n-1}$. By the definition of de Bruijn strings, $\text{occ}(v0, x) = \text{occ}(v1, x) = 1$. As $\text{occ}(v, x) \geq 2$ (as $\text{occ}(0^{n-1}, x) = 3$), a context for v would have been created, and as v is not unique in x, contexts for each its branching paths have to be created, namely $v0$ and $v1$. However, one more bit is required to finish building the context in the case where $v0 = 10^{n-1}$ (the final n bits of x) as the model cannot build a context until it can say what it predicts. Hence $|x| + 1 = 2^n + n$ bits are needed in total.

□

The following lemma bounds the maximum number of characters contributed by any singular character to the encoding within a C_n zone.

Lemma 7.3.4. *For all but finitely many n, if $\overline{C_{n-1}}$ has already been read, each character in C_n contributes at most $5 \log n$ bits to the final encoding.*

Proof. For n large, let b be the current character of C_n being encoded. Let x be the context used to predict b. Then $|x| \geq n - 1$ as all contexts of length $n-2$ and below are non-deterministic in C_n as seen in Lemma 7.3.3. Worst case scenario, x will be deterministic but will not predict b correctly and so transmits \$. For $j \leq n - 1$, $\text{occ}(x, C_j \cdot C_{j+1}[0..n-3]) \leq j$, and $\text{occ}(x, C_n) \leq 2n$. Thus for n large, we can bound the maximum possible number of occurrences above by $\sum_{j=1}^{n-1} j + 2n \leq n^2$. This results in \$ contributing at most $\lceil \log(n^2+1) \rceil$ characters.

Next, b will be transmitted by the non-deterministic context $x[1..|x|-1]$ of length at least $n-2$. Using the same logic, this context will have appeared at most j times in $C_j \cdot C_{j+1}[0..n-4]$ for $j \leq n-2$, and a most $2(n-1)$ times in $C_{n-1} \cdot C_n[0..n-4]$ and at most $4n$ times in C_n. Thus, we can bound the maximum

possible number of occurrences by

$$(\sum_{j=1}^{n-2} j) + 2(n-1) + 4n \leq n^2$$

for n large. As such, for all but finitely many n, b contributes at most

$$\lceil \log(n^2+2) \rceil + \lceil \log(n^2+1) \rceil \leq 5 \log n \tag{7.1}$$

characters to the final encoding. As b was arbitrary, we have the desired result.

□

For each C_n, we will refer to its first $2^n + 2n$ bits as its *bad zone*. This is the section of the string where contexts often incorrectly predict bits, requiring \$ to be transmitted. Ideally after the first $2^n + n$ bits of C_n are encoded, either the contexts used to predict $C_n[0..2^n + n - 1]$ will have been deterministic and will continue to correctly predict the remaining bits of C_n, or new deterministic contexts will have been created that correctly predict the remaining bits of C_n. This may not always occur in the succeeding n characters and often occurs if the original contexts used straddle two previous C_i sections.

For instance, consider the substring $C_7[0..135] = d_7 \cdot 0^7 1$. The final 1 will be predicted by a context of length at least 7 by Lemma 7.3.3. 0^7 is the context of length 7 that may be used. However 0^7 is not a deterministic context. Since C_4 ends with 10^3 and C_5 begins with $0^5 1$, this results in the substring $10^8 1$ straddling two zones. Specifically $\text{occ}(0^7, 10^8 1) = 2$ with $\text{occ}(0^7 0, 10^8 1) = 1$ and $\text{occ}(0^7 1, 10^8 1) = 1$. Instead 10^7 is a context of length 8 that may be used. We know it exists as $\text{occ}(10^6, \overline{C_6} \cdot C_7[0..135]) = 2$. It occurs once in the straddle of C_4 and C_5 ($10^3 0^5 1$), and again in the straddle of C_6 and C_7 ($100^7 1$), both times followed by 0. It does not appear anywhere else (as 0^7 cannot occur anywhere else) and so it incorrectly predicts 0. Therefore \$ is transmitted.

Corollary 7.3.5. *For all but finitely many n, if $\overline{C_{n-1}}$ has already been read, the* bad zone *of C_n contributes at most $5(2^n + 2n)\log n$ characters to the encoding of C.*

Proof. Lemma 7.3.4 tells us that each character of the *bad zone* contributes at most $5 \log n$ characters to the final encoding. As the *bad zone* has length (2^n+2n), the result follows.

□

7.3.3 Main Result

In this subsection we prove our main result in Theorem 7.3.8 that C satisfies $R_{\mathrm{PPM}^*}(C) = 0$. Compression is achieved from the repetition of the de Bruijn strings which lead to the repeated use of deterministic contexts which make correct predictions. When deterministic contexts are used, correct predictions are performed with probability $p/(p+1)$, for some $p \in \mathbb{N}$. As p increases, the number of characters contributed to the encoding by that prediction approaches 0.

The following lemma shows that deterministic contexts correctly predict every character outside of the *bad zone* for infinitely many C_n zones.

Lemma 7.3.6. *For all but finitely many n, whenever n is odd or $n = 2^j$ for some $j > 0$, all bits not in the* bad zone *of C_n are correctly predicted by deterministic contexts.*

Proof. For odd n, the $2^n + 2n + 1^{\text{th}}$ bit in S_n will always be a 1 (if n is a power of 2, a similar argument holds but we look at the $2^n + 2n^{\text{th}}$ bit). This is because $d_n \cdot d_n[0..2n] = d_n \cdot 0^n 10^{n-2}11$ by Remark 7.3.2. The context used to predict this 1 will always be a suffix of the context $010^{n-2}1$. This context exists as we have that $\mathrm{occ}(010^{n-2}1, C_n[0..2^n + 2n]) = 2$.

First we show $010^{n-2}1$ is deterministic by showing it never occurs as a substring in $\overline{C_{n-1}}$, i.e. that $\mathrm{occ}(010^{n-2}1, \overline{C_{n-1}}) = 0$. The only place the string 10^{n-2}

occurs is in C_{n-1} where it would be preceded by 1^{n-2} and not a 0, in C_{n-2} where it would be preceded by 1^{n-3} and not a 0, or along a straddle between two prior C_i's for $i \leq n-1$, but again, it would be preceded by a 1, and not a 0. Hence, $010^{n-2}1$ first appears in $C_n[0..2^n-1] = d_n$ where it is followed by a 1. Thus $010^{n-2}1$ is deterministic.

As the context $010^{n-2}1$ is deterministic, all extensions of this context (those of the form $010^{n-2}1y$ for the appropriate $y \in \{0,1\}^*$) that are built while reading C_n, must be deterministic also. They remain deterministic throughout the reading of C_n since due its construction, any substring of C_n of length at least n is always followed by the same bit. Thus, every bit not in the *bad zone* of C_n is predicted by a deterministic context which is a suffix of an extension of the deterministic context $010^{n-2}1$.

□

For n-even but not of the form 2^j for some j, Lemma 7.3.6 does not hold. While most bits are predicted by deterministic contexts, the shifts of the de Bruijn strings in the construction of C_n between blocks $B_{n,0}$ and $B_{n,1}$ mean that some contexts which may have originally been deterministic in $B_{n,0}$, soon see the opposite bit due to the shift of de Bruijn strings used to construct the different blocks.

For example, consider context 1^60^5. We have that $\text{occ}(1^60^5, \overline{C_5}) = 0$, but it does occur in C_6 multiple times. The first two times it occurs it sees a 0 (as $d_6[2^6-6..2^6-1] \cdot d_6[0..5] = 1^60^6$) by Remark 7.3.2). The third time it sees a 1 due to the shift in $B_{6,1}$ (as $d_6[2^6-6..2^6-1] \cdot d_{6,1}[0..5] = 1^60^51$). Hence, 1^60^5 is no longer deterministic. This can be seen in Figure 7.2.

The following result bounds the number of characters each C_n zone contributes to the encoding. Here $|\text{PPM}^*(C_n \,|\, \overline{C_{n-1}})|$ represents the number of bits contributed to the output by C_n if the encoder has already processed $\overline{C_{n-1}}$.

Lemma 7.3.7. *For all but finitely many n, $|\mathrm{PPM}^*(C_n \,|\, \overline{C_{n-1}})| \leq 13(2^n \log n)$.*

Proof. By Lemma 7.3.6, every character outside the *bad zone* is predicted correctly by a deterministic context when n is odd or when $n = 2^j$ for some $j > 0$. This is not true for the remaining n as mentioned in the discussion preceding this lemma. As such, the output contributed by the case where n is even but not a power of 2 acts as an upper bound for all n.

In this case, $n = 2^s t$ for some $s \geq 1$ and $t \geq 3$ where t is odd. Recall that $C_n = B_0 \cdot B_1 \cdots B_{2^s-1}$ where $B_i = d^t_{n,i}$. Let $b_n = 2^s$, the number of blocks in C_n.

After processing the *bad zone*, a 1 is deterministically correctly predicted by the context $010^{n-2}1$ with probability at least $1/2$ by the same argument as in Lemma 7.3.6. Following this, the next $2^n - n - 2$ bits are also predicted by deterministic contexts as these contexts are suffixes of extensions of $010^{n-2}1$ and see the same bits within C_n. The next time $010^{n-2}1$ is seen it predicts a 1 with probability at least $2/3$ and so on. When $010^{n-2}1$ is seen for the n^{th} time, there are only $2^n - 2n + b_n - 1$ bits left to encode as the encoder has 'fallen behind' by $b_n - 1$ bits due to the $b_n - 1$ shifts that occur. These bits are encoded with probability at least $(n-1)/n$.

Excluding the *bad zone* and using that $b_n < 2^n$, the characters we have accounted for (of which there are $(2^n - n - 2)(n-2) + 2^n - 2n + b_n - 1$) contribute at most

$$\left\lceil -\log\left(\left(\frac{n-1}{n}\right)^d \prod_{i=2}^{n-1} \left(\frac{i-1}{i}\right)^c\right)\right\rceil < \left\lceil \log\left((n-1)^n n^{2^{n+1}}\right)\right\rceil < 3(2^n \log n) \qquad (7.2)$$

bits where $c = 2^n - n - 1$ and $d = 2^n - 2n + b_n - 1$. To help visualise this, Table 7.3 shows the bits of C_6 which will be correctly predicted by deterministic contexts that we use in our calculations.

Things differ with the encoding of the remaining characters to account for as they may be impacted by the shifts that occur between blocks. The number of

such characters is bounded above by $n^2 - n - 1 - b_n < n^2$.

Then by Equation (7.2), Lemma 7.3.4, Corollary 7.3.5 and that $2^n + 2n + n^2 < 2(2^n)$ for n large, we have that

$$|\mathrm{PPM}^*(C_n \,|\, \overline{C_{n-1}})| \leq 5(2(2^n)\log n) + 3(2^n \log n) = 13(2^n \log n). \tag{7.3}$$

□

00000010000110001010001110010010110011010011110101011101101111111
00000010000110001010001110010010110011010011110101011101101111111
00000010000110001010001110010010110011010011110101011101101111111
00000100001100010100011100100101100110100111101010111011011111110
00000100001100010100011100100101100110100111101010111011011111110
00000100001100010100011100100101100110100111101010111011011111110

Table 7.3: The bits of C_6 shaded in blue above are those which we know will be correctly predicted by deterministic contexts.

We now can prove the main result of this chapter.

Theorem 7.3.8. $R_{\mathrm{PPM}^*}(C) = 0$.

Proof. Note that the worst compression of prefixes of C is achieved if the input ends with a complete *bad zone*, i.e. prefixes of the form $C_n^* = \overline{C_{n-1}} C_n[0..2^n + 2n - 1]$.

Let j be such that Lemma 7.3.7 holds for all zones C_i with $i \geq j$. The prefix $\overline{C_{i-1}}$ will always contribute $O(1)$ bits. Using Lemma 7.3.7 to bound the contribution to the output by $C_j \cdots C_{n-1}$ and Lemma 7.3.4 to bound $C_n[0..2^n + 2n - 1]$'s contribution to the output by $5(2^n + 2n)\log n$ characters, we have that

$$|\mathrm{PPM}^*(C_n^*)| \leq \sum_{i=j}^{n-1} (13(2^i \log(i)) + 5(2^n + 2n)\log n + O(1). \tag{7.4}$$

As $|C_n^*| = 2^n(n-1) + 2n + 2$, it follows that $R_{\mathrm{PPM}^*}(C^k) = 0$.

□

7.3.4 Bounded PPM on Normal Sequences

In this section we prove that for any t, no normal sequence can be compressed by PPM_t.

Theorem 7.3.9. *Let $S \in \{0,1\}^\omega$ be normal. Then for all t, $\rho_{\mathrm{PPM}_t}(S) = 1$.*

Proof. Let S and t be as above. Let $x \in \{0,1\}^t$ and $\varepsilon > 0$. As S is normal, for almost every n and for $b \in \{0,1\}$,

$$n(2^{-(t+1)} - \varepsilon) < \mathrm{occ}(xb, S[0..n-1]) < n(2^{-(t+1)} + \varepsilon). \tag{7.5}$$

Thus, for almost every n, each time x is used to predict the next character it has prediction probability p such that

$$p \leq \frac{n(2^{-t-1} + \varepsilon)}{2n(2^{-t-1} - \varepsilon) + 2} \leq \frac{(2^{-t-1} + \varepsilon)}{2(2^{-t-1} - \varepsilon)}.$$

That is, $p \leq 1/2 + \beta_\varepsilon$ for all but finitely many n where $\beta_\varepsilon > 0$ and can be made arbitrarily small by choosing ε small. Hence, almost every time x makes a prediction, it contributes at least $-\log(1/2 + \beta_\varepsilon)$ bits to the encoding. This value gets arbitrarily close to 1 as β_ε is chosen to be closer to zero. As x was an arbitrary context, it follows that $\rho_{\mathrm{PPM}_t}(S) = 1$.

□

We do not explore a notion of PPM-depth in this thesis. However, with a bounded and unbounded version of the algorithm, the most intuitive definition would be that a sequence S is PPM-deep if

$$(\exists \alpha > 0)(\forall t \in \mathbb{N})(\forall^\infty n \in \mathbb{N})\, |\mathrm{PPM}_t(S \upharpoonright n)| - |\mathrm{PPM}^*(S \upharpoonright n)| \geq \alpha n.$$

The existence of such a PPM-deep sequence would then easily follow from Theorems 7.3.8 and 7.3.9.

Whether or not this definition of PPM-depth is meaningful, in the sense that 'easy' and 'random' are not deep and that a slow growth law holds, remains open.

Remark 7.3.10. As an aside, as C is a Champernowne sequence, it holds that $\rho_{\mathrm{LZ}}(C) = 1$ and so C is not LZ-deep by Theorem 5.3.3. However, in Chapter 5 we discussed how similar versions of LZ-depth could be defined based on other members of the Lempel-Ziv family of algorithms. For instance, Pierce and Shields show that $R_{\mathrm{ULZ}}(C) = 0$ [119] where ULZ is a variation of the Lempel-Ziv 77 algorithm. It then follows easily that C would be an example of an ULZ-deep sequence.

7.4 Bounded versus Unbounded

The main question that must be asked when choosing between Bounded PPM and PPM* is what bound of context length should be used: finite or infinite? Choosing a length too small may mean the encoder cannot spot patterns that may lead to compression. On the other hand, setting the upper bound for length to be large means that while the contexts may have a greater chance of being deterministic, they may not get to be used frequently enough to achieve compression. In practical terms, another aspect to consider is that larger upperbounds require more memory. We do not consider memory needs in our analysis.

Cleary and Witten explored this problem in their original paper, compressing various files ranging from English text to binary data, and found that the optimal context length for many of their experiments was around four [45]. Moffat found similar results in his experiments [111].

This leads to the question as to whether there exists a sequence S such that for all bounds t, $\rho_{\mathrm{PPM}^*}(S) > R_{\mathrm{PPM}_t}(S)$? To our knowledge, all comparisons between the performances of the PPM family of algorithms have been experimental in nature and no examples have been proven to be mathematically different.

We partially explore this question by showing that for every $t \geq 1$, there exists a sequence S^t such that

$$\lim_{n\to\infty} \frac{|\mathrm{PPM}_t(S^t \upharpoonright n)|}{|\mathrm{PPM}^*(S^t \upharpoonright n)|} = \frac{1}{t}.$$

In each case, S^t is a rather simple sequence where it holds that $R_{\mathrm{PPM}^*}(S^t) = R_{\mathrm{PPM}_t}(S^t) = 0$. Thus, while the S^t is both PPM*-trivial and PPM$_t$-trivial, PPM$_t$ compresses much 'faster' than PPM*.

Fix a bound $t \in \mathbb{N}$. We construct the sequence S^t in blocks $B_{t,1} \cdot B_{t,2} \cdots$ where for each i, $B_{t,i} = 0^{it}1$. For instance, $S^3 = 0001 \cdot 0000001 \cdot 0000000001 \ldots$. For all i, we write $\overline{B_{t,i}}$ to denote the prefix $B_{t,1} \cdots B_{t,i}$ of S^t.

The number of occurrences of 0^t will be important to us in this section. A simple counting exercise shows that

$$\mathrm{occ}(0^t, \overline{B_{t,n-1}}) = \sum_{j=1}^{n-1} (t(j-1)+1) = \frac{n-1}{2}(tn - 2t + 2). \tag{7.6}$$

We henceforth use $k_{t,n-1}$ to denote $\mathrm{occ}(0^t, \overline{B_{t,n-1}})$.

7.4.1 PPM$_t$'s Performance on S^t

We first examine the output of PPM$_t$ on prefixes of the form $\overline{B_{t,n}}$.

Lemma 7.4.1. *For all $t \in \mathbb{N}$, we have that*

$$|\mathrm{PPM}_t(\overline{B_{t,n}})| \leq t\log(n-1) + (n+2)\log(t(n+1)^2) - \log((n-1)!) + O(1).$$

Proof. Once $B_{t,1}$ and $B_{t,2}$ have been read by PPM$_t$, the $t+1$ contexts 0^l10^m and 0^t with $0 \leq l, m \leq t-1$ where $l + m = t - 1$ are used to predict every bit of the remainder of S^t.

Beginning with block $B_{t,3}$, the first t 0s are each predicted using one of the t

contexts containing a 1 with prediction probability 1/2 as it will be the first time they are used and have only seen 0 in $B_{t,1} \cdot B_{t,2}$. The remaining $2t$ 0s are predicted by the context 0^t. As $\mathrm{occ}(0^t, B_{t,1}B_{t,2}) = t + 2$, where t times it is followed by a 0 and twice followed by a 1, the first time 0^t predicts a 0 in $B_{t,3}$ will be with probability $t/(t + 4)$. The prediction probabilities of the next $2t - 1$ 0s is easily found by incrementing the numerator and denominator of $t/(t + 4)$ by one for each 0. This means that when the final 1 of $B_{t,3}$ is read, it is predicted with probability $2/((t + 4) + 2t)$.

Next, the bits of $B_{t,4}$ are read. The first t are predicted by contexts involving 1s with probabilities now 2/3. The prediction probability of the first 0 predicted by 0^t is found by incrementing the numerator by one of the prediction probability used the last time the 0^t predicted a 0 and incrementing the denominator by one of the prediction probability used the last time 0^t predicted 1, i.e. with probability $(t + 2t)/((t + 4) + 2t + 1)$.

This can easily be generalised to any block $B_{t,n}$ as follows: The first t 0s are predicted each with probability $(n - 2)/(n - 1)$. Using Formula (7.6), we have that the first 0 in $B_{t,n}$ predicted by 0^t is done so with probability

$$\frac{k_{t,n-1} - (n - 1)}{k_{t,n-1} + 2}. \tag{7.7}$$

Note the numerator is decremented by $n - 1$ to account for $n - 1$ instances when 0^t is followed by a 1. Thus, every instance 0 is predicted by 0^t in $B_{t,n}$, it is done so with a probability of the form

$$\frac{k_{t,n-1} - (n - 1) + j}{k_{t,n-1} + 2 + j} \tag{7.8}$$

n	**Prediction Probabilities of** $B_{3,3} \cdots B_{3,6}$																			
3	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{3}{7}$	$\frac{4}{8}$	$\frac{5}{9}$	$\frac{6}{10}$	$\frac{7}{11}$	$\frac{8}{12}$	$\frac{2}{13}$										
4	$\frac{2}{3}$	$\frac{2}{3}$	$\frac{2}{3}$	$\frac{9}{14}$	$\frac{10}{15}$	$\frac{11}{16}$	$\frac{12}{17}$	$\frac{13}{18}$	$\frac{14}{19}$	$\frac{15}{20}$	$\frac{16}{21}$	$\frac{17}{22}$	$\frac{3}{23}$							
5	$\frac{3}{4}$	$\frac{3}{4}$	$\frac{3}{4}$	$\frac{18}{24}$	$\frac{19}{25}$	$\frac{20}{26}$	$\frac{21}{27}$	$\frac{22}{28}$	$\frac{23}{29}$	$\frac{24}{30}$	$\frac{25}{31}$	$\frac{26}{32}$	$\frac{27}{33}$	$\frac{28}{34}$	$\frac{29}{35}$	$\frac{4}{36}$				
6	$\frac{4}{5}$	$\frac{4}{5}$	$\frac{4}{5}$	$\frac{30}{37}$	$\frac{31}{38}$	$\frac{32}{39}$	$\frac{33}{40}$	$\frac{34}{41}$	$\frac{35}{42}$	$\frac{36}{43}$	$\frac{37}{44}$	$\frac{38}{45}$	$\frac{39}{46}$	$\frac{40}{47}$	$\frac{41}{48}$	$\frac{42}{49}$	$\frac{43}{50}$	$\frac{44}{51}$	$\frac{5}{52}$

Table 7.4: The prediction probabilities used for $B_{3,3} \cdots B_{3,6}$ by PPM_3.

where $0 \leq j \leq t(n-1)-1$. Similarly, the 1 in $B_{t,n}$ is predicted with probability

$$\frac{n-1}{k_{t,n-1}+t(n-1)+2}. \tag{7.9}$$

In summary, this means that $B_{t,n}$ contributes the following probabilities to the final encoding:

$$\underbrace{\Big(\frac{n-2}{n-1}\Big)^t}_{\text{The first } t \text{ 0s}} \cdot \underbrace{\Big(\prod_{j=0}^{t(n-1)-1} \frac{k_{t,n-1}-(n-1)+j}{k_{t,n-1}+2+j}\Big)}_{0^t \text{ predicting a 0}} \cdot \underbrace{\Big(\frac{n-1}{k_{t,n-1}+t(n-1)+2}\Big)}_{0^t \text{ predicting a 1}}. \tag{7.10}$$

To help visualise this, Table 7.4 contains the probabilities for the substring $B_{3,3} \cdots B_{3,6}$ of S^3.

Let

$$a_{t,n} = \Big(\frac{n-2}{n-1}\Big)^t,$$

$$b_{t,n} = \Big(\prod_{j=0}^{t(n-1)-1} \frac{k_{t,n-1}-(n-1)+j}{k_{t,n-1}+2+j}\Big),$$

and

$$c_{t,n} = \Big(\frac{n-1}{k_{t,n-1}+t(n-1)+2}\Big).$$

Multiplying the prediction probabilities together and simplifying gives us that

$$\prod_{i=3}^{n} a_{t,i} \cdot b_{t,i} \cdot c_{t,i} = \Big(\frac{(n-1)!}{(n-1)^t}\Big)\Big(\frac{(t+3)!}{(t-1)!}\Big)\Big(\frac{(k_{t,n-1}-(n-1)+t(n-1)-1)!}{(k_{t,n-1}+t(n-1)+2)!}\Big)$$
$$= \frac{(n-1)!(t+3)(t+2)(t+1)(t)}{(n-1)^t(\prod_{l=0}^{n+1}(k_{t,n-1}+t(n-1)+2-l))}. \tag{7.11}$$

Noting that for $n \geq 1$

$$k_{t,n-1}+t(n-1)+2 = \frac{n-1}{2}(t(n+1)-2t+2)+t(n-1)+2 \leq \frac{t(n+1)^2}{2},$$

the denominator of Equation (7.11) can be bounded above by

$$\Big(\frac{t(n+1)^2}{2}\Big)^{n+2}.$$

Hence, since $|\mathrm{PPM}_t(\overline{B_{t,n}})| = \log((\prod_{i=3}^{n} a_{t,i}\cdot b_{t,i}\cdot c_{t,i})^{-1})+O(1)$, by Equation (7.11) we have that

$$|\mathrm{PPM}_t(\overline{B_{t,n}})| \leq \log((n-1)^t) + \log((\frac{t(n+1)^2}{2})^{n+2}) - \log((n-1)!) + O(1)$$
$$\leq t\log(n-1) + (n+2)\log(t(n+1)^2) - \log((n-1)!) + O(1). \tag{7.12}$$

This proves the lemma.

□

Next we show that S^t is compressible by PPM_t.

Proposition 7.4.2. $R_{\mathrm{PPM}_t}(S^t) = 0$.

Proof. Note that PPM_t performs the worst on prefixes of S^t of the form $\overline{B_{t,n}}$ as these are the prefixes where the final character encoded is always a 1, i.e. the characters which contribute the most to the output.

Hence, by Lemma 7.4.1 we have that

$$\lim_{n\to\infty} \frac{|\mathrm{PPM}_t(\overline{B_{t,n}})|}{|\overline{B_{t,n}}|} \leq \lim_{n\to\infty} \frac{t\log(n-1) + (n+2)\log(t(n+1)^2) - \log((n-1)!)}{\frac{n}{2}(tn+t+2)}$$
$$= 0. \tag{7.13}$$

Thus $R_{\mathrm{PPM}_t}(S^t) = 0$.

□

7.4.2 PPM*'s Performance on S^t

We now examine the output of PPM* on prefixes of the form $\overline{B_{t,n}}$.

Lemma 7.4.3. *For all $t \geq 1$, we have that*

$$|\mathrm{PPM}^*(\overline{B_{t,n}})| \leq (n-2)\log(p_t^{-1}) + t\log(n-1) + t\log((n-1)!) + O(1),$$

where $p_t = 1/(20(t-1)(t)(t+3))$ if $t > 1$ and $p_t = 1/24$ if $t = 1$.

Proof. Assume the encoder has read $B_{t,1} \cdot B_{t,2}$. Beginning with block $B_{t,3}$, the first $2t$ 0s are encoded with the contexts $1, 10, \ldots, 10^{2t-1}$. These contexts exist as they are seen straddling $B_{t,1}$ and $B_{t,2}$, and just before the current bit being predicted is read. They each deterministically predict a 0 with probability 1/2.

Only the suffix 0^t1 is left to encode in $B_{t,3}$. The next bit seen is a 0 where the context 0^{2t} is used to predict it. It exists as it occurs in $B_{t,2}$ and has just been read in the previous $2t$ bits. However it has only seen a 1 so far and so an escape is transmitted with probability 1/2. Then, 0^{2t-1} predicts a 0 with probability 2/5 (it has occurred 3 times previously: once followed by 1 and twice by 0). The next 0 is encoded by 0^{2t} (0^{2t+1} does not exist yet as a context). It predicts a 0 with probability 1/4 (it has occurred twice previously, once followed by 1 and 0). The remaining $t-2$ zeroes are predicted by the context 0^{2t+1} which has just

been created. It deterministically predicts a 0 and so the first 0 is predicted with probability 1/2 and the $t-2^{th}$ 0 with probability $(t-2)/(t-1)$. For the 1, an escape is transmitted by 0^{2t+1} with probability $1/t$ and then transmitted by 0^{2t} with probability $1/(t+3)$ (as it has been seen $t+1$ times where it has been followed by a 1 once at the end of $B_{t,2}$).

Next we examine $B_{t,4}$. The first $2t$ 0s are encoded by the contexts 10^i for $0 \leq i \leq 2t-1$ as before, but this time with probability 2/3. The next 0 (whose equivalent positioned 0 in $B_{t,3}$ required an escape) is now deterministically predicted by the context $0^{t+1}10^{2t}$ with probability 1/2. The next $t-1$ 0s are predicted deterministically by the just created contexts $10^{2t+1}, 10^{2t+2}, \ldots, 10^{3t-1}$ with probability 1/2.

The final $t+1$ bits (0^t1) are predicted as follows: The next bit seen is a 0 where the context 0^{3t} is used to predict it. It exists as it occurs in $B_{t,3}$ and has just been read in the previous $3t$ bits. However, it has only seen a 1 so far and so an escape is transmitted with probability 1/2. Then, 0^{3t-1} predicts a 0 with probability 2/5 (it has occurred 3 times previously: once followed by 1 and twice by 0). The next 0 is encoded by 0^{3t} (0^{3t+1} does not exist as a context). It predicts a 0 with probability 1/4 (it has occurred twice previously, once followed by 1 and 0). The remaining $t-2$ zeros are predicted by the context 0^{3t+1} which has just been created. It deterministically predicts a 0 and so the first 0 is predicted with probability 1/2 and the $t-2^{th}$ 0 with probability $(t-2)/(t-1)$. For the 1, an escape is transmitted by 0^{3t+1} with probability $1/t$ and then transmitted by 0^{3t} with probability $1/(t+3)$ (as it has been seen $t+1$ times where it has been followed by a 1 once at the end of $B_{t,3}$).

We can generalise this to any block $B_{t,j}$ as follows: The first $2t$ 0s are predicted by contexts $1, 10, \ldots, 10^{2t-1}$ with probability $(j-2)/(j-1)$. The next $(j-3)t$

bits are split into $(j-3)$ windows of size t, i.e.

$$B_{t,j} = 0^{2t} \cdot \underbrace{0^t \cdot 0^t \cdots 0^t \cdot 0^t}_{j-3 \text{ copies of } 0^t} \cdot 0^t 1.$$

These windows are numbered 1 to $j-3$ in order of appearance. For window i, the first 0 is predicted by context $0^{it+1}10^{(2+(i-1))t}$ with probability $(j-2-i)/(j-1-i)$. The next $t-1$ 0s are predicted by contexts $10^{(2+(i-1))t+1}, \ldots, 10^{(2+(i-1))t+t-1}$ with probability $(j-2-i)/(j-1-i)$ also. The suffix of 0^t1 is encoded as follows. First suppose $t \neq 1$. The context

$$0^{(2+(j-3)-1))t+t} = 0^{(j-2)t+t} = 0^{(j-1)t}$$

is used to predict the first 0. However it has only seen a one so far (in $B_{t,j-1}$) and so an escape is transmitted with probability 1/2. $0^{(j-1)t-1}$ then predicts the 0 with probability 2/5. The next 0 is encoded by $0^{(j-1)t}$ ($0^{(j-1)t+1}$ does not exist as a context yet). It predicts a 0 with probability 1/4 (it has occurred twice previously, once followed by 1 and 0). The remaining $t-2$ zeroes are predicted by the context $0^{(j-1)t+1}$ which has just been created. It deterministically predicts a 0 and so the first 0 is predicted with probability 1/2 and the $t-2^{th}$ 0 with probability $(t-2)/(t-1)$. For the 1, an escape is transmitted by $0^{(j-1)t+1}$ with probability $1/t$ and then transmitted by $0^{(j-1)t}$ with probability $1/(t+3)$ (as it has been seen $t+1$ times where it has been followed by a 1 once at the end of $B_{t,j-1}$).

Otherwise if $t = 1$, the suffix 01 is encoded as follows. Firstly the context 0^{j-1} is used to predict the 0. However it has only been followed by a 1 so far (in $B_{1,j-1}$) and so an escape is transmitted with probability 1/2. 0^{j-2} then predicts the 0 with probability 1/3 as it has been followed by a 0 twice (once in $B_{1,j}$ and $B_{1,j-1}$) and a 1 twice (once in $B_{1,j-1}$ and $B_{1,j-2}$). The 1 is then predicted by the

n	**Prediction Probabilities of** $B_{3,3} \cdots B_{3,6}$
3	$\frac{1}{2}$ $\frac{1}{2}$ $\frac{1}{2}$ $\frac{1}{2}$ $\frac{1}{2}$ $\frac{1}{2}$ $\frac{1}{2}$ $\frac{2}{5}$ $\frac{1}{4}$ $\frac{1}{2}$ $\frac{1}{3}$ $\frac{1}{6}$
4	$\frac{2}{3}$ $\frac{2}{3}$ $\frac{2}{3}$ $\frac{2}{3}$ $\frac{2}{3}$ $\frac{2}{3}$ $\frac{1}{2}$ $\frac{1}{2}$ $\frac{1}{2}$ $\frac{1}{2}$ $\frac{2}{5}$ $\frac{1}{4}$ $\frac{1}{2}$ $\frac{1}{3}$ $\frac{1}{6}$
5	$\frac{3}{4}$ $\frac{3}{4}$ $\frac{3}{4}$ $\frac{3}{4}$ $\frac{3}{4}$ $\frac{3}{4}$ $\frac{2}{3}$ $\frac{2}{3}$ $\frac{2}{3}$ $\frac{1}{2}$ $\frac{1}{2}$ $\frac{1}{2}$ $\frac{1}{2}$ $\frac{2}{5}$ $\frac{1}{4}$ $\frac{1}{2}$ $\frac{1}{3}$ $\frac{1}{6}$
6	$\frac{4}{5}$ $\frac{4}{5}$ $\frac{4}{5}$ $\frac{4}{5}$ $\frac{4}{5}$ $\frac{4}{5}$ $\frac{3}{4}$ $\frac{3}{4}$ $\frac{3}{4}$ $\frac{2}{3}$ $\frac{2}{3}$ $\frac{2}{3}$ $\frac{1}{2}$ $\frac{1}{2}$ $\frac{1}{2}$ $\frac{1}{2}$ $\frac{2}{5}$ $\frac{1}{4}$ $\frac{1}{2}$ $\frac{1}{3}$ $\frac{1}{6}$

Table 7.5: The prediction probabilities used in $B_{3,3} \cdots B_{3,6}$ by PPM*.

context 0^{j-1} with probability $1/4$ as it has now been followed by both a 0 and a 1 once.

Hence, setting

$$p_t = \begin{cases} \frac{1}{24} & \text{if } t = 1 \\ \frac{1}{20(t-1)(t)(t+3)} & \text{if } t \neq 1, \end{cases} \tag{7.14}$$

to represent the bits contributed by the suffix 0^t1, $B_{t,j}$ contributes

$$\underbrace{\left(\frac{j-2}{j-1}\right)^{2t}}_{\text{prefix } 0^{2t}} \cdot \underbrace{\prod_{l=2}^{j-2}\left(\frac{l-1}{l}\right)^{t}}_{j-3 \text{ windows of } 0^t} \cdot \underbrace{p_t}_{\text{suffix } 0^t1} \tag{7.15}$$

bits to the encoding. Note that the $1/(20(t-1)(t)(t+3))$ term in Equation (7.14) comes from the fact that when $t \neq 1$, the formula for p_t is as follows:

$$p_t = \frac{1}{2} \cdot \frac{2}{5} \cdot \frac{1}{4} \cdot \frac{1}{t} \cdot \frac{1}{t+3} \cdot \prod_{j=2}^{t-1} \frac{j-1}{j} = \frac{1}{20(t-1)(t)(t+3)}. \tag{7.16}$$

Note that statement (7.15) can be simplified to

$$\frac{(j-2)^t}{(j-1)^{2t}} \cdot p_t. \tag{7.17}$$

To help visualise this, Table 7.5 contains the prediction probabilities PPM* uses on $B_{3,3} \cdots B_{3,6}$.

In total, zones $B_{t,3} \cdots B_{t,n}$ contribute the following number of bits to the output:

$$p_t^{n-2} \cdot \prod_{j=3}^{n} \frac{(j-2)^t}{(j-1)^{2t}} = \frac{p_t^{n-2}}{(n-1)^{2t}} \cdot \prod_{j=1}^{n-2} \frac{1}{j^t} = \frac{p_t^{n-2}}{(n-1)^t((n-1)!)^t}. \tag{7.18}$$

Therefore we have that

$$|\mathrm{PPM}^*(\overline{B_{t,n}})| = (n-2)\log(p_t^{-1}) + t\log(n-1) + t\log((n-1)!) + O(1). \tag{7.19}$$

This proves the lemma.

□

Next we show that S^t is compressible by PPM*.

Proposition 7.4.4. $R_{\mathrm{PPM}^*}(S^t) = 0$.

Proof. Note that PPM* performs the worst on prefixes of S^t of the form $\overline{B_{t,n}}$ as these are the prefixes where the final character is encoded is always a 1, i.e. the characters which contribute the most to the output.

Hence, by Lemma 7.4.3 we have that

$$\begin{aligned}
\lim_{n\to\infty} \frac{|\mathrm{PPM}^*(\overline{B_{t,n}})|}{|\overline{B_{t,n}}|} &= \lim_{n\to\infty} \frac{(n-2)\log(p_t^{-1}) + t\log(n-1) + t\log((n-1)!)}{\frac{n}{2}(tn+t+2)} \\
&\leq \lim_{n\to\infty} \frac{(n-2)\log(p_t^{-1}) + t\log(n-1) + t(n-1)\log((n-1))}{\frac{n}{2}(tn+t+2)} \\
&= 0.
\end{aligned} \tag{7.20}$$

Thus $\rho_{\mathrm{PPM}^*}(S) = 0$.

□

7.4.3 Comparing the Two

Above we saw that (unsurprisingly) both PPM_t and PPM^* compress S^t with a compression ratio of 0. We now prove the main theorem of this section.

Theorem 7.4.5. *Let $t \geq 1$. It holds that*

$$\lim_{m\to\infty} \frac{|\mathrm{PPM}_t(S^t \upharpoonright m)|}{|\mathrm{PPM}^*(S^t \upharpoonright m)|} = \frac{1}{t}.$$

Proof. For each m, let n_m denote the least non-negative integer such that $S \upharpoonright m \sqsubseteq \overline{B_{t,n_m}}$. That is $S \upharpoonright m$ is a prefix of $\overline{B_{t,n_m}}$ but not $\overline{B_{t,n_m-1}}$. Note that for all m it holds that

$$\frac{|\mathrm{PPM}_t(\overline{B_{t,n_m-1}})|}{|\mathrm{PPM}^*(\overline{B_{t,n_m}})|} \leq \frac{|\mathrm{PPM}_t(S^t \upharpoonright m)|}{|\mathrm{PPM}^*(S^t \upharpoonright m)|} \leq \frac{|\mathrm{PPM}_t(\overline{B_{t,n_m}})|}{|\mathrm{PPM}^*(\overline{B_{t,n_m-1}})|}. \tag{7.21}$$

We will use this inequality to prove the theorem.

Firstly, we shall examine

$$\frac{|\mathrm{PPM}_t(\overline{B_{t,n_m}})|}{|\mathrm{PPM}^*(\overline{B_{t,n_m-1}})|}.$$

From Equation (7.19) we have that

$$|\mathrm{PPM}^*(\overline{B_{t,n_m-1}})| = (n_m - 3)\log(p_t^{-1}) + t\log(n_m - 2) + t\log((n_m - 2)!) + O(1)$$

where $p_t = \frac{1}{(t-1)(t)(t+3)}$ if $t > 1$ and $t = 1/24$ otherwise. From Equation (7.12) we have that

$$|\mathrm{PPM}_t(\overline{B_{t,n_m}})| \leq t\log(n_m - 1) + (n_m + 2)\log(t(n_m + 1)^2) - \log((n_m - 1)!) + O(1).$$

It follows that

$$\lim_{m\to\infty} \frac{t\log(n_m-1)+(n_m+2)\log(t(n_m+1)^2)-\log((n_m-1)!)+O(1)}{(n_m-3)\log(p_t^{-1})+t\log(n_m-2)+t\log((n_m-2)!)+O(1)} = \frac{1}{t}. \tag{7.22}$$

Hence

$$\lim_{m\to\infty} \frac{|\mathrm{PPM}_t(\overline{B_{t,n_m}})|}{|\mathrm{PPM}^*(\overline{B_{t,n_m-1}})|} \leq \frac{1}{t}. \tag{7.23}$$

Next we examine

$$\frac{|\mathrm{PPM}_t(\overline{B_{t,n_m-1}})|}{|\mathrm{PPM}^*(\overline{B_{t,n_m}})|}.$$

Again from Equation (7.19) we have that

$$|\mathrm{PPM}^*(\overline{B_{t,n_m}})| = (n_m-2)\log(p_t^{-1})+t\log(n_m-1)+t\log((n_m-1)!)+O(1).$$

Recall that from Equation (7.11), rearranging we find that

$$\begin{aligned}|\mathrm{PPM}_t(\overline{B_{t,n_m-1}})| =& t\log(n_m-2)+\log(\prod_{l=0}^{n_m}(k_{t,n_m-2}+t(n_m-2)+2-l)\\ &-\log((n_m-2)!)+O(1).\end{aligned}$$

Noting that

$$\begin{aligned}k_{t,n_m-2}+t(n_m-2)+2-n_m &\geq \frac{(n_m-2)}{2}\Big(t(n_m-1)-2t+2\Big)-n_m\\ &\geq \frac{(n_m-2)}{2}\Big(t(n_m-1)-2t\Big)-n_m\\ &\geq \frac{n_m}{2}(tn_m-5t-2),\end{aligned}$$

we have that

$$\begin{aligned}(|\overline{B_{t,n_m-1}})| \geq & t\log(n_m-2)+(n_m+1)\log(\frac{n_m}{2}(tn_m-5t-2)) \\ & -\log((n_m-2)!)+O(1). \end{aligned} \tag{7.24}$$

It follows that

$$\lim_{m\to\infty}\frac{t\log(n_m-2)+(n_m+1)\log(\frac{n_m}{2}(tn_m-5t-2))-\log((n_m-2)!)}{(n_m-2)\log(p_t^{-1})+t\log(n_m-1)+t\log((n_m-1)!)}=\frac{1}{t}. \tag{7.25}$$

That is,

$$\lim_{m\to\infty}\frac{|\mathrm{PPM}_t(\overline{B_{t,n_m-1}})|}{|\mathrm{PPM}^*(\overline{B_{t,n_m}})|}\geq\frac{1}{t}. \tag{7.26}$$

Hence by Equations (7.21), (7.23) and (7.26), it follows that

$$\lim_{m\to\infty}\frac{|\mathrm{PPM}_t(S^t\upharpoonright m)|}{|\mathrm{PPM}^*(S^t\upharpoonright m)|}=\frac{1}{t}.$$

□

7.5 Summary

In this chapter we performed the first analysis of PPM* on normal sequences. By using a construction of Pierce and Shields, we demonstrated that there exists a Champernowne sequence C such that $R_{\mathrm{PPM}^*}(C)=0$ in Theorem 7.3.8. This construction method will be explored again in Chapter 8. We furthermore showed that regardless of the bound of context length chosen, Bounded PPM cannot compress normal sequences in Theorem 7.3.9.

In Section 7.4, we conducted a small investigation into whether a maximum context length can be cleverly chosen such that Bounded PPM can beat PPM*.

From a chosen bound t, we showed in Theorem 7.4.5 that one can construct a simple sequence S^t such that PPM*'s output on prefixes of S^t is t times longer than PPM$_t$'s output.

Chapter 8

Automatic Complexity of Normal Sequences

Contents of this chapter were presented (virtually) at FSTTCS 2021 in India during December 2021. doi: 10.4230/LIPIcs.FSTTCS.2021.47.

8.1 Introduction

The following chapter examines the complexity of finite strings and sequences via the complexity measure based on deterministic finite-state automata (DFA) introduced by Shallit and Wang known as *automatic complexity* [130]. As such, this chapter can be viewed as a cousin of Chapter 3 which similarly used finite-state automata with outputs to examine strings and sequences. Being a finite-state based complexity, we are interested whether normal sequences must have maximal automatic complexity. To answer this, we demonstrate the existence of normal sequences whose prefixes have an automatic complexity ratio of between 0 and 1/2. We furthermore show that prefixes of the Champernowne sequences constructed in Chapter 7 (Section 7.3.1) have an automatic complexity ratio bounded above by 2/3.

8.1.1 Motivation

In Chapter 3 we examined one method of calculating the complexity of strings and sequences via finite-state transducers and their minimal descriptions. Recall that the complexity of a string x was related to the length of the smallest string y such that a transducer on input y would output x. The longer y must be, the more complex x is as it requires more bits to be described. This approach of defining finite-state complexity has been taken in many places [31, 34, 56, 57].

Instead of being based on outputting the string x, one may define the complexity to be the amount of computing power required to distinguish x from all other strings of length $|x|$. As described in Definition 2.2.10, Sipser was the first to introduce this idea of a *Distinguishing Complexity* [134]. For a string x, Shallit and Wang define its automatic complexity $A(x)$ to be the minimum number of states required by a deterministic finite-state automaton such that x is the only string of length $|x|$ it accepts. A non-deterministic variation was first examined by Hyde [83]. A similar complexity notion based on finding the size of the smallest context-free grammar such that x is the only string of length $|x|$ the grammar generates was explored in [41].

In their original paper, Shallit and Wang proved upper and lower bounds of automatic complexity for various sets of strings and of prefixes of the infinite Thue-Morse [19] sequence. Expanding on this line of research, Kjos-Hanssen has recently studied the automatic complexity of Fibonacci and Tribonacci sequences [87].

Depending on how finite-state complexity is measured, normal sequences may have high or low complexity. As previously discussed, if complexity is defined as compressibility by lossless finite-state compressors, normal sequences have maximum complexity (see previously mentioned [11, 14, 35, 49, 128]). For the minimal description approach, if the size of the transducers are restricted when examining

all prefixes of the sequence, normal sequences have maximal complexity, as seen in Theorem 3.2.8. If not, normal sequences can have arbitrarily low complexity, as seen in Theorem 3.2.10. A survey of old and new results can be found in [95].

As automatic complexity is more of a 'combinatorial' rather than an 'information content' measurement[1], this leads to the question as to how low can the automatic complexity of normal sequences be? As the number of DFAs with n states exceeds the number of strings of length n, one would expect that most strings would have an automatic complexity value which is smaller than their length. In fact, Shallit and Wang demonstrate in Theorem 6 of [130] that almost every string x satisfies

$$A(x) \leq \frac{3|x|}{4} + \log|x|\sqrt{\frac{|x|}{8}}. \tag{8.1}$$

In terms of lower bounds, in Theorem 8 of [130] Shallit and Wang show that for almost every string x it holds that

$$A(x) \geq \frac{|x|}{13}. \tag{8.2}$$

They also state that the 13 term above can be replaced with 7 after personal communication with Petersen. Kjos-Hanssen has recently shown that 13 can be replaced with $2 + \varepsilon$ for all $\varepsilon > 0$ [89]. For completeness of both Equations (8.1) and (8.2), a set $S \subseteq \{0,1\}^*$ contains 'almost every' string means that

$$\lim_{n\to\infty} \frac{|S \bigcap \{0,1\}^n|}{2^n} = 1.$$

Previously, the automatic complexity of finite strings produced by linear feedback shift registers which have a maximal number of distinct substrings (otherwise

[1] In an information content measurement, we would like there to be roughly 2^n objects with a complexity of n. For automatic complexity, it holds that all strings of the form 0^n and 1^n have an automatic complexity of 2.

known as m-sequences) [88] along with sequences and finite strings which do not contain k-powers, i.e. substrings of the form x^k, have been studied [84, 87, 130]. Normal sequences by definition contain x^k as a substring infinitely often for every possible pair (x, k). Is there a trade-off between the randomness of normal sequences resulting in high complexity, in the sense that they contain every string as a substring infinitely often, and does the fact that some of those substrings have the form x^k result in low automatic complexity? We explore this question in this chapter.

8.1.2 Automatic Complexity Definitions

Recall Definition 3.2.1 of a finite-state transducer. The definition for finite-state automata is similar.

Definition 8.1.1. A *deterministic finite-state automaton (DFA)* M is defined by a 4-tuple $M = (Q, q_0, \delta, F)$, where

- Q is a non-empty, finite set of *states*,
- $q_0 \in Q$ is the *initial state*,
- $\delta : Q \times \{0, 1\} \to Q$ is the *transition function*,
- $F \subseteq Q$ is the set of *final* or *accepting states*.

The transition function and extended transition function of a DFA for strings is defined recursively the same way as for an FST. We say a DFA *accepts* the string x if $\delta(q_0, x) \in F$. The *language* of a DFA M, denoted by $L(M)$, is the set of strings that M accepts. That is, $L(M) = \{x : \delta(q_0, x) \in F\}$. Shallit and Wang define *automatic complexity* as follows.

Definition 8.1.2 ([130])**.** Let $x \in \{0, 1\}^*$. The *automatic complexity* of x, denoted by $A(x)$, is the minimal number of states required by a deterministic finite automaton M such that $L(M) \bigcap \{0, 1\}^{|x|} = \{x\}$.

Definition 8.1.3. A DFA M *uniquely accepts* a string x if $L(M) \bigcap \{0,1\}^{|x|} = \{x\}$.

Shallit and Wang compute the following two ratios to examine the automatic complexity of sequences.

Definition 8.1.4. The *lower* and *upper* bounds for the automatic complexity ratio of a sequence $T \in \{0,1\}^{\omega}$ are respectively given by

$$I(T) = \liminf_{m\to\infty} \frac{A(T \upharpoonright m)}{m} \text{ and, } S(T) = \limsup_{m\to\infty} \frac{A(T \upharpoonright m)}{m}.$$

8.2 Normal Sequences with a Low Automatic Complexity Ratio

We require the use of de Bruijn strings during constructions in this chapter. We point towards Definition 2.6.1 for a refresher.

In our first result we construct a normal sequence T such that $I(T) = 0$, that is, infinitely many prefixes have minimal automatic complexity. We furthermore show that $S(T) \leq 1/2$, indicating that the sequence does not have high complexity. We first require the following theorem of Nandakumar and Vangapelli.

Theorem 8.2.1 ([116])**.** *Let $f : \mathbb{N} \to \mathbb{N}$ be increasing such that for all n, $f(n) \geq n^n$. Then every sequence of the form $T = d_1^{f(1)} d_2^{f(2)} d_3^{f(3)} \cdots$ where d_n is a de Bruijn string of order n, is normal* [2].

[2]Nandakumar and Vangapelli's original result was for when $f(n) = n^n$. However, their argument easily carries over for $f(n) \geq n^n$ also and this fact has been used by other authors such as in [33, 34].

Theorem 8.2.2. *There is a normal sequence T such that $I(T) = 0$ and $S(T) \leq 1/2$.*

Proof. We recursively define the sequence $T = T_1T_2\ldots$ and the function $f : \mathbb{N} \to \mathbb{N}$ as follows. For all $j \geq 1$, let d_j be a de Bruijn string of order j such that if j is odd, d_j begins with a 1 and if j is even, d_j begins with a 0. We set $f(1) = 2$ and $T_1 = d_1^{f(1)} = d_1^2$. For $j \geq 2$, we define $f(j) = |T_1 \ldots T_{j-1}|^{|T_1 \ldots T_{j-1}|}$ and $T_j = d_j^{f(j)}$. Note that $f(1) > 1$ and for all $j \geq 2$, $f(j) \geq |T_{j-1}|^{|T_{j-1}|}$ and that $|T_{j-1}| = 2^{j-1}f(j-1) \geq j$. Hence by Theorem 8.2.1, T is normal. For simplicity, we write $\overline{T_j}$ for the prefix $T_1 \cdots T_j$ of T.

We first show that $I(T) = 0$. Consider a prefix of the form $\overline{T_n}$. $\overline{T_n}$ is uniquely accepted by the DFA M_1 which has a state for each bit of $\overline{T_{n-1}}$ and then has a loop of length 2^n for the string d_n which can be seen in Figure 8.1. M_1 has $|\overline{T_{n-1}}| + 2^n + 1$ states. Thus we have that

$$\begin{aligned}\frac{A(\overline{T_n})}{|\overline{T_n}|} &\leq \frac{|\overline{T_{n-1}}| + 2^n + 1}{|T_n| + |\overline{T_{n-1}}|} = \frac{|\overline{T_{n-1}}| + 2^n + 1}{2^n f(n) + |\overline{T_{n-1}}|} \\ &\leq \max\left\{\frac{|\overline{T_{n-1}}|}{2^n|\overline{T_{n-1}}|^{|\overline{T_{n-1}}|}}, \frac{2^n+1}{|\overline{T_{n-1}}|}\right\} \leq \max\left\{\frac{1}{2^n}, \frac{2^n+1}{(n-1)^{n-1}}\right\}. \qquad (8.3)\end{aligned}$$

Hence it follows that $I(T) = 0$.

Next consider an arbitrary prefix $T \upharpoonright m$ of T. Let n be largest such that $\overline{T_n}$ is a prefix of $T \upharpoonright m$ but $\overline{T_{n+1}}$ is not. Thus $T \upharpoonright m = \overline{T_n} \cdot w$ for some $w \sqsubset T_{n+1}$ and is uniquely accepted by the DFA M_2 in Figure 8.1. M_2 has $|\overline{T_{n-1}}| + 2^n + |w| + 1$ states.

Consider when $1 \leq |w| \leq 2^n(f(n) - 1) + 2^{n+1}$. Note that

$$|T_{n+1}| = 2^{n+1}f(n+1) = 2^{n+1}(f(n+1) - 1) + 2^{n+1} > 2^n(f(n) - 1) + 2^{n+1}$$

so w can be this length. Hence for such w we have that

$$\begin{aligned}
\frac{A(T \upharpoonright m)}{m} &\leq \frac{|\overline{T_{n-1}}| + 2^n + |w| + 1}{|\overline{T_n}| + |w|} \\
&\leq \frac{|\overline{T_{n-1}}| + 2^n + 2^n(f(n)-1) + 2^{n+1} + 1}{|\overline{T_n}| + 2^n(f(n)-1) + 2^{n+1}} \\
&= \frac{|\overline{T_{n-1}}| + |T_n| + 2^{n+1} + 1}{|\overline{T_{n-1}}| + 2|T_n| + 2^n} \\
&= \frac{|\overline{T_{n-1}}| + 2^n(|\overline{T_{n-1}}|^{|\overline{T_{n-1}}|} + 2) + 1}{|\overline{T_{n-1}}| + 2(2^n|\overline{T_{n-1}}|^{|\overline{T_{n-1}}|}) + 2^n} \\
&\leq \max\left\{\frac{1 + 2^n(|\overline{T_{n-1}}|^{|\overline{T_{n-1}}|-1} + 2|\overline{T_{n-1}}|^{-1})}{1 + 2(2^n|\overline{T_{n-1}}|^{|\overline{T_{n-1}}|-1})}, \frac{1}{2^n}\right\}. \qquad (8.4)
\end{aligned}$$

Note, for all $\varepsilon > 0$, Equation (8.4) is bounded above by $1/2 + \varepsilon$ as n increases.

Furthermore consider when $2^n(f(n)-1) + 2^{n+1} < |w| \leq |T_{n+1}|$. Instead of looping on d_n, it becomes more beneficial to loop on d_{n+1} via a DFA similar to M_1 in Figure 8.1 where the accepting state is a single state in the loop depending on the length of w. Thus for such prefixes $A(\overline{T_n} \cdot w) \leq |\overline{T_n}| + 2^{n+1} + 1$, i.e. it does not depend on w. Hence the ratio $A(T \upharpoonright m)/m$ decreases and approaches $I(T)$ for such w.

Therefore by Equation (8.4), $S(T) \leq 1/2$.

□

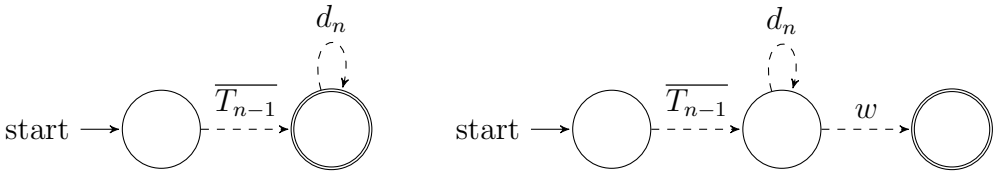

Figure 8.1: DFA M_1 (left) and M_2 (right) from Theorem 8.2.2. The error state (the state traversed to if the bit seen is not the expected bit) and arrows to it are not included. By the construction of T, $d_n[0] \neq w[0]$ to ensure determinism.

8.3 Automatic Complexity of a Champernowne Sequence

Recall the subset of normal sequences known as Champernowne sequences introduced in Chapter 2 which were formed by listing all strings of length 1 followed by all strings of length 2 and so on. With this restrictive property on their construction, one may wonder if such sequences must have maximal automatic complexity. We answer this in the following section by demonstrating the existence of a Champernowne sequence C such that $S(C) \leq 2/3$. We specifically examine sequences built using Pierce and Shields' construction [119] of Champernowne sequences discussed in Section 7.3.1 of Chapter 7 involving successive concatenations of de Bruijn strings. Recall that the set of sequences constructed using their method was denoted by PSC.

Before examining the main result of this section, we require the following result from number theory.

Theorem 8.3.1. *For $a, b, c \in \mathbb{Z}$, consider the Diophantine equation $ax + by = c$. If there exists a solution to the equation (x_0, y_0) where $x_0, y_0 \in \mathbb{Z}$, then all other solutions (x', y') such that $x', y' \in \mathbb{Z}$ are of the form $x' = x_0 + (b/g)d$ and $y' = y_0 - (a/g)d$ where $d \in \mathbb{Z}$ is arbitrary and $g = \gcd(a, b)$.*

Theorem 8.3.2. *There exists a Champernowne sequence $C \in$ PSC such that $S(C) \leq 2/3$.*

Proof. For each n, let d_n be a de Bruijn string of order n with prefix 0^n. Let $C \in$ PSC be a Champernowne sequence constructed using Pierce and Shields' method where for each substring C_n, d_n is the de Bruijn string used to construct it. Recall, for example, the construction of zone C_n of C. n can be uniquely written in the form $n = 2^s t$ for $s \geq 0$ and $t \geq 1$ where t is odd. Then $C_n = B_{n,0} \cdots B_{n,2^s-1}$ where $B_{n,j} = d_{n,j}^t$ such that $d_{n,j} = d_n[j..2^n - 1] \cdot d_n[0..j-1]$. Recall then if n is

odd that $C_n = d_n^n$ and if n is a power of 2 (i.e. $t = 1$) that $C_n = d_n \cdot d_{n,1} \cdots d_{n,n-1}$. Again, we use $\overline{C_n}$ to denote the prefix $C_1 C_2 \cdots C_n$ of C.

Consider an arbitrary prefix $C \upharpoonright m$ of C. Let n be largest such that $\overline{C_{n+1}}$ is a prefix of $C \upharpoonright m$, but $\overline{C_{n+2}}$ is not. To examine $A(C \upharpoonright m)$ we build automata which make use of two loops which exploit the repetitions of the de Bruijn strings in C_n and C_{n+1}. The automata have one accepting state which depends on the length of the prefix being examined. There are four cases to consider which are dependent on the value of n, and the four automata can be seen in Figure 8.4.

- **Case 1**: n is a power of 2,
- **Case 2**: $n + 1$ is a power of 2,
- **Case 3**: n is even but not a power of 2,
- **Case 4**: $n + 1$ is even but not a power of 2.

Notation wise, we let v_n be the string such that $d_n = 0^n 1 v_n$. Note that $d_n[n] = 1$ as otherwise the string 0^n would appear twice as a substring of d_n. Also not that the final bit of d_n must be a 1 also.

Suppose we are in Case 3 where $n = 2^s t$ where $s \geq 1$ and $t \geq 3$ where t is odd. Let p_{n+2} denote the prefix of C_{n+2} such that $C \upharpoonright m = \overline{C_{n+1}} p_{n+2}$. The automaton for Case 3 in Figure 8.4 accepts $C \upharpoonright m$ by reading the prefix $\overline{C_{n-1}} \cdot 0^n$ state by state, then traversing the first loop 2^s times, then reading 0^{2^s+1}, then traversing the second loop fully n times and then up to reading $1 v_{n+1}$ on the $n+1^{\text{th}}$ traversal of it. Then, depending on the length of p_{n+2}, we read the final 0^{n+1} of the second loop and exit it to read the remainder of $C_{n+2}[n+1..]$ as needed. That is, if $|p_{n+2}| \leq n+1$ then the final state is contained in the last $n+1$ states (including the root state) of the second loop of the DFA, else once finishing the loop, we traverse through $|p_{n+2}| - (n+1)$ extra states to the accepting state.

To help the reader visualise these loops, Figures 8.2 and 8.3 are provided to show which bits of zones C_3, C_4 and C_6 are read on a single traversal of a loop when the lexicographic least de Bruijn strings of order $3, 4$ and 6 are used.

To see that $C \restriction m$ is accepted by the DFA, note that the traversal through the DFA described above can be factored as $\overline{C_{n-1}} x p_{n+2}$ where

$$x = 0^n (1 v_n d_n^{t-1} 0^{n-1})^{2^s} 0^{2^s+1} (1 v_{n+1} 0^{n+1})^n (1 v_{n+1}).$$

00010111	0000100110101111
00010111	0001001101011110
00010111	0010011010111100
	0100110101111000

Figure 8.2: The bits shaded in blue indicate which bits are read on a single traversal of the loops used when reading C_3 (left) and C_4 (right) respectively.

0000001000011000101000111001001011001101001111010101110110111111
0000001000011000101000111001001011001101001111010101110110111111
0000001000011000101000111001001011001101001111010101110110111111
0000010000110001010001110010010110011010011110101011011011111110
0000010000110001010001110010010110011010011110101011011011111110
0000010000110001010001110010010110011010011110101011011011111110

Figure 8.3: The bits shaded in blue indicate which bits are read on a single traversal of the loop used when reading C_6.

Note that $x = C_n C_{n+1}$ since

$$\begin{aligned}
x &= 0^n(1v_n d_n^{t-1}0^{n-1})^{2^s}0^{2^s+1}(1v_{n+1}0^{n+1})^n(1v_{n+1}) \\
&= (0^n 1v_n d_n^{t-1})0^{n-1}(1v_n d_n^{t-1}0^{n-1})^{2^s-1}0^{2^s+1}(1v_{n+1}0^{n+1})^n(1v_{n+1}) \\
&= B_0(0^{n-1}1v_n d_n^{t-1}0)0^{n-2}(1v_n d_n^{t-1}0^{n-1})^{2^s-1}0^{2^s+1}(1v_{n+1}0^{n+1})^n(1v_{n+1}) \\
&= B_0B_1(0^{n-2}1v_n d_n^{t-1}0^2)0^{n-3}(1v_n d_n^{t-1}0^{n-1})^{2^s-2}0^{2^s+1}(1v_{n+1}0^{n+1})^n(1v_{n+1}) \\
&\vdots \qquad \text{(Keep repeating this process of sectioning into the } 2^s \text{ blocks)} \\
&= B_0B_1\cdots B_{2^s-1}0^{n-2^s}0^{2^s+1}(1v_{n+1}0^{n+1})^n(1v_{n+1}) \\
&= C_n(0^{n+1}1v_{n+1})^{n+1} = C_nC_{n+1}.
\end{aligned}$$

Next we show that the DFA uniquely accepts $C \restriction m$. Note that all strings accepted have length

$$|\overline{C_{n-1}}| + n + (2^n t - 1)a + 2^s + 1 + 2^{n+1}b + |p_{n+2}| - (n+1)$$

where $a, b \geq 0$ are positive integers. As stated $(a, b) = (2^s, n+1)$ is a solution to the Diophantine equation

$$|\overline{C_{n-1}}| + n + (2^n t - 1)a + 2^s + 1 + 2^{n+1}b + |p_{n+2}| - (n+1) = |C \restriction m|. \quad (8.5)$$

By Theorem 8.3.1, as the first loop has odd length and the second has even length, all integer solutions to Equation (8.5) take the form

$$(2^s + 2^{n+1}c,\ n + 1 - (2^n t - 1)c)$$

where $c \in \mathbb{Z}$. As $2^s = n/t$ and $t \geq 3$, we have that $(n+1) - (2^n t - 1)c < 0$ when $c > 0$ and $2^s + 2^{n+1}c < 0$ when $c < 0$, it follows that $c = 0$ is the only possibility that gives non-negative integer solutions, i.e. $C \restriction m$ is uniquely accepted by the

DFA.

The number of states of the automaton is bounded above by

$$|C_{n-1}| + n + 2^n t + 2^s + 2^{n+1} + |p_{n+2}|.$$

As $2^s \leq n/3$, we then have that

$$\frac{A(C \upharpoonright m)}{m} \leq \frac{|\overline{C_{n-1}}| + 2^n t + \frac{4n}{3} + 2^{n+1} + |p_{n+2}|}{|\overline{C_{n+1}}| + |p_{n+2}|}. \tag{8.6}$$

Hence for $|p_{n+2}| \leq 2^n(n-t) - \frac{n}{3} + 1 + 2^{n+2}(\frac{n+2}{2})$ we find that

$$\begin{aligned}
\frac{A(C \upharpoonright m)}{m} &\leq \frac{|\overline{C_{n-1}}| + 2^n t + \frac{4n}{3} + 2^{n+1} + |p_{n+2}|}{|\overline{C_{n+1}}| + |p_{n+2}|} \\
&\leq \frac{|\overline{C_{n-1}}| + |C_n| + n + 2^{n+1} + 1 + 2^{n+2}(\frac{n+2}{2})}{|\overline{C_{n+1}}| - \frac{n}{3} + 1 + 2^{n+2}(\frac{n+2}{2})} \\
&= \frac{|\overline{C_n}| + n + 2^{n+1} + 1 + 2^{n+2}(\frac{n+2}{2})}{|\overline{C_{n+1}}| - \frac{n}{3} + 1 + 2^{n+2}(\frac{n+2}{2})}.
\end{aligned} \tag{8.7}$$

For all $\varepsilon > 0$, we note that Equation (8.7) is bounded above by $2/3 + \varepsilon$ as n increases.

However as $|p_{n+2}|$ increases, it becomes more advantageous to use a loop for the repetitions in C_{n+2} as opposed to the loop for C_n (similar to the proof of Theorem 8.2.2). In the worst case scenario, $n+2$ has the form $2^{s'}t'$ for $s' \geq 1$ and $t' \geq 3$ where t' is odd. This results in an automaton of Case 4, as shown in Figure 8.4, where the accepting state is one of that states of the second loop. One can show that the prefix is uniquely accepted as before with a similar argument. Such an automaton requires at most $|\overline{C_n}| + 2^{n+1} + n + 1 + 2^{n+2}(\frac{n+2}{2})$ states (as $t' \leq (n+2)/2$).

Hence for $j \geq 1$ such that $2^n(n-t) - \frac{n}{3} + 1 + 2^{n+2}(\frac{n+2}{2}) \leq |p_{n+2}| + j < |C_{n+2}|$

we use the automaton from Case 4 and find that

$$\frac{A(C \upharpoonright m)}{m} \leq \frac{|\overline{C_n}| + n + 2^{n+1} + 1 + 2^{n+2}(\frac{n+2}{2})}{|\overline{C_{n+1}}| - \frac{n}{3} + 1 + 2^{n+2}(\frac{n+2}{2}) + j}. \tag{8.8}$$

One can see that for such p_{n+2}, the number of states of the automaton used to calculate (8.8) remains constant and the ratio decreases as j increases.

Similar calculations for the other three cases show that none achieve an upper bound greater than $2/3 + \varepsilon$ resulting in $S(C) \leq 2/3$. As the calculations are similar, for completeness and readability purposes, these are presented in Section 8.3.2 at the end of this chapter.

□

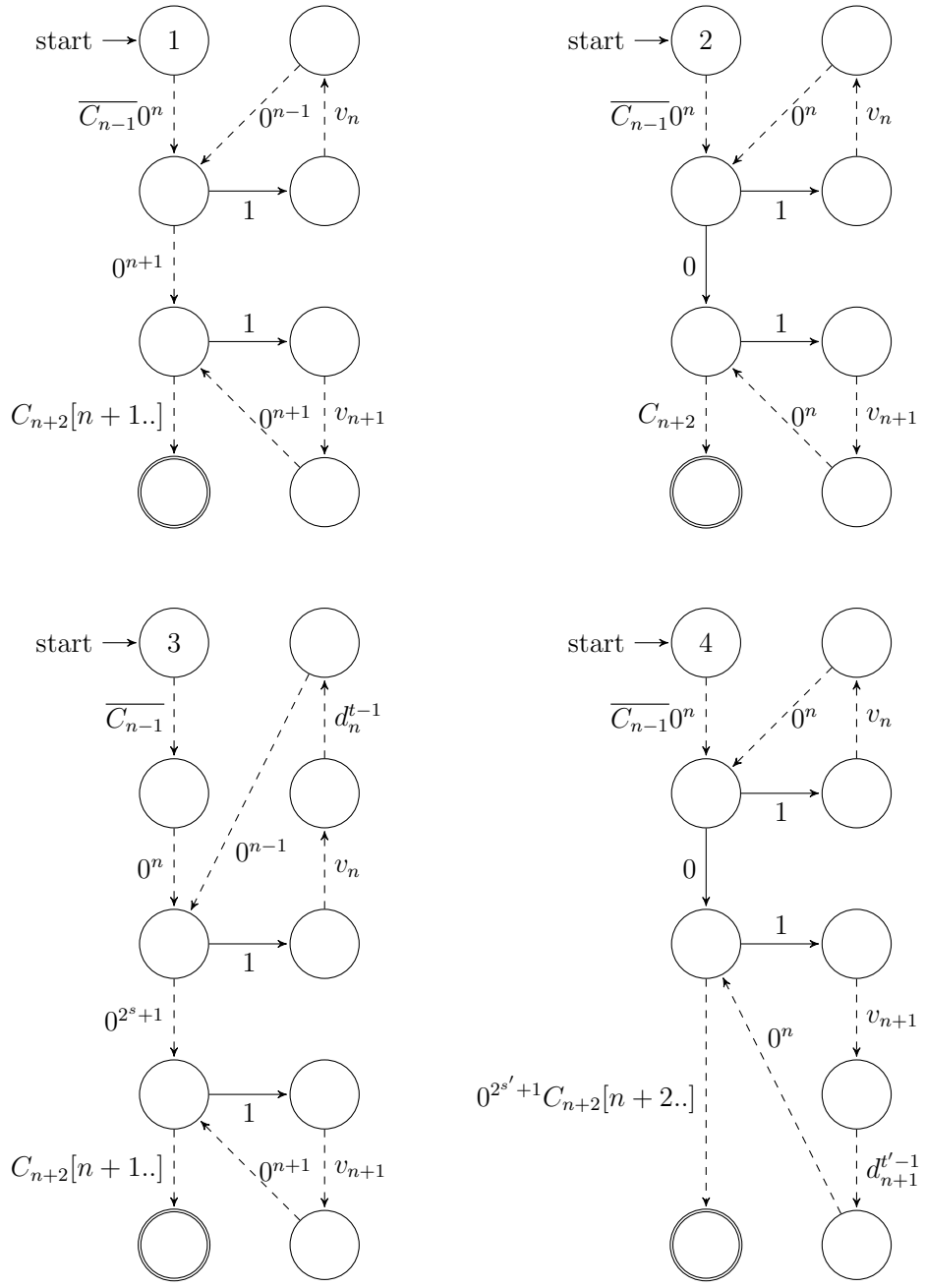

Figure 8.4: Automata for Case 1 (top left), Case 2 (top right), Case 3 (bottom left) and Case 4 (bottom right) for Theorem 8.3.2. The dashed arrows represent the missing states belonging to their labels. The error states and arrows to it are not included.

8.3.1 Lower Bounds for Champernowne Sequences

Let $C \in$ PSC satisfy Theorem 8.2.2. As part of Theorem 8.2.2, we do not provide any insight into the value of $I(C)$. Currently many of the techniques for calculating lower bounds rely on the absence of k-powers (such as proofs in [87, 130]). In particular, Shallit and Wang show that for every x without k-powers, x satisfies $A(x) \geq (|x|+1)/k$. However, as C is a Champernowne sequence, long enough prefixes will contain k-powers, i.e. there eventually is a substring x such that $x = u^k$ for some string u.

However, one can easily identify an upper bound for $I(C)$ as follows. Consider prefixes of the form $\overline{C_{n+1}}$ where n is a power of 2, i.e. we are in Case 1. The automaton in Figure 8.4 for Case 1, where the final state is contained appropriately in the second loop, will uniquely accept the prefix and simple calculations give us that $I(C) \leq 1/4$.

One prefix x of C such that $A(x)/|x| < 1/4$ which is in Case 1 is $\overline{C_{65}}$. The automaton for Case 1 in Figure 8.4 which uniquely accepts $\overline{C_{65}}$ has $n_1 = |\overline{C_{63}}| + 2^{64} + 2^{65} + 128$ states. However, the number of states can be reduced further by using two more loops for zones C_{62} and C_{63} instead of having a state for each of their bits. Consider the DFA $\widehat{M}$ shown in Figure 8.5:

$\widehat{M}$ reads the prefix $\overline{C_{61}} \cdot 0^{62}$. It then traverses a loop for $1v_{62}(d_{62})^{30}0^{61}$. It then reads 0^3 and enters a loop for the string $(1v_{63}0^{63})^{21}$. Following this it reads $01v_{64}0^{63}$ and then enters a loop for the string $(1v_{64}0^{63})^7$. It then reads 0^{65} and enters a loop for the string $(1v_{65}0^{65})^5$, with the final state being that state of this loop after reading $(1v_{65}0^{65})^4 \cdot 1v_{65}$. $\widehat{M}$ can be thought of as combining the DFAs from Figure 8.4, but altering the length of the loops for the zones. Strings of length $|\overline{C_{65}}|$ that $\widehat{M}$ accepts satisfy the equation

$$|\overline{C_{61}}| + (2^{62} \cdot 31 - 1)a + (21 \cdot 2^{63})b + (7 \cdot (2^{64} - 1))c + (5 \cdot 2^{65})d + 65 + 2^{64} = |\overline{C_{65}}| \quad (8.9)$$

where it must hold $d \geq 1$ so the final loop is traversed to reach the accepting state. $a = 2$, $b = 3$, $c = 9$ and $d = 13$ is the only non-negative integer solution to Equation (8.9) and so $\widehat{M}$ uniquely accepts C_{65}. $\widehat{M}$ has

$$n_2 = |\overline{C_{61}}| + 31 \cdot 2^{62} + 7 \cdot 2^{63} + 8 \cdot 2^{64} + 5 \cdot 2^{65} + 120$$

states which is less than n_1. Hence $\widehat{M}$ gives us that $A(\overline{C_{65}})/|\overline{C_{65}}| < 0.173 < 1/4$.

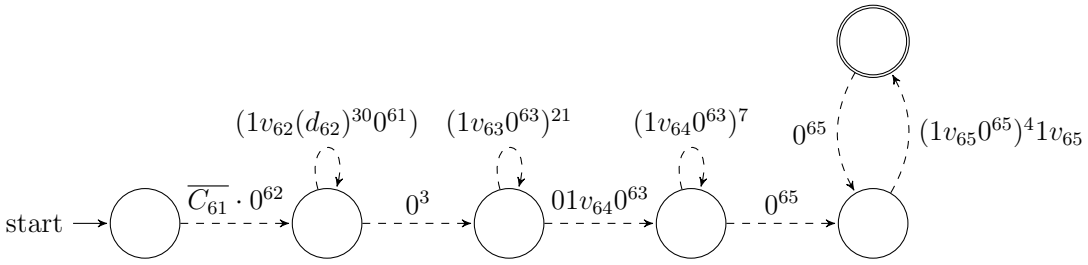

Figure 8.5: Automaton $\widehat{M}$ for $\overline{C_{65}}$. The dashed arrows represent the missing states belonging to their labels. The error state (the state traversed to if the bit seen is not the expected bit) and arrows to it are not included.

While the above demonstrates that more than two loops can be used, the size of the loops are limited in each case. The following remark examines the potential lengths of the possible loops that can be used in an arbitrary C_n zone.

Remark 8.3.3. Let $n \in \mathbb{N}$ and $s \geq 0$ and $t \geq 1$ where t is odd such that $n = 2^s t$. Let $x \in \{0,1\}^*$ such that $|x| \geq n$. If $\mathrm{occ}(xx, C_n) \geq 1$, then the possibilities for the length of x include:

1. $|x| = 2^n a$, where $1 \leq a \leq \lfloor \frac{t}{2} \rfloor$,
2. $|x| = 2^n b - 1$, where $1 \leq b \leq t$,
3. $|x| = (2^n t - 1)c$, where $1 \leq c \leq 2^{s-1}$.

Proof. Recall that $C_n = B_0 \cdots B_{2^s-1}$, where for each i, $B_i = d_{n,i}^t$. So $|B_i| = 2^n t$. In the following we will write d_i instead of $d_{n,i}$ for notational clarity as we are

exclusively examining zone C_n.

1. First suppose that $\text{occ}(xx, B_i) \geq 1$ for some i, i.e. xx is contained in a full block B_i. Assume $n \leq |x| < 2^n$. Then x can be factored into $x = yz$ where $|y| = n$ and $|z| < 2^n - n$. Thus yzy is a substring of B_i which has length at most $2^n + n - 1$. By the construction of B_i, yzy is therefore a prefix of $\sigma = d_i[k..] \cdot d_i[0..k + n - 2]$ for some k. For all $\tau \in \{0,1\}^n$, $\text{occ}(\tau, \sigma) = 1$, however $\text{occ}(y, \sigma) = 2$, which is a contradiction.

   Next suppose $2^n a < |x| < 2^n(a + 1)$. If $a > \lfloor \frac{t}{2} \rfloor$, then $|xx| > |B_i|$ which contradicts $\text{occ}(xx, B_i) \geq 1$. Hence $a \leq \lfloor \frac{t}{2} \rfloor$. x can be factored into substrings $x = yz$ where $|y| = 2^n a$ and $|z| < 2^n$. Hence, yzy is a substring of B_i. We therefore have that

$$yzy = (d_i[k..2^n - 1] \cdot d_i[0..k - 1])^a \cdot z \cdot (d_i[k..] \cdot d_i[0..k - 1])^a$$

   for some k. This forces

$$z = (d_i[k..] \cdot d_i[0..k - 1])^c$$

   for some $c \geq 1$ or $z = \lambda$, both of which contradict the possible lengths of z. Hence we must have that $|x| = 2^n a$, where $1 \leq a \leq \lfloor \frac{t}{2} \rfloor$ establishing the first case.

2. Next consider the case where xx is a substring of C_n such that the first x is a suffix of block B_i and the second is a prefix of block B_{i+1}. This forces x to be of the form $d_i[1..] \cdot (d_i)^b$ where $b < t$, as otherwise, x is not a prefix of B_{i+1}. Noting that for all i,

$$d_i[1..2^n - 1] \cdot 0 = d_{i+1} \text{ and } d_i = 0 \cdot d_{i+1}[0..2^n - 2],$$

we see that xx is a substring of C_n as

$$\begin{aligned} d_i[1..2^n-1] \cdot d_i^b &= d_i[1..2^n-1] \cdot 0 \cdot d_i[1..2^n-1] \cdot d_i^{b-1} \\ &= d_{i+1} \cdot d_i[1..2^n-1] \cdot 0 \cdot d_i[1..2^n-1] \cdot d_i^{b-2} \\ &= d_{i+1}^2 \cdot d_i[1..2^n-1] \cdot d_i^{b-2} \\ &\vdots \\ &= d_{i+1}^{b-1} \cdot d_i[1..2^n-1] \\ &= d_{i+1}^{b-1} \cdot d_{i+1}[0..2^n-2]. \end{aligned}$$

As $d_i[1..2^n-1] \cdot d_i^b$ is a suffix of B_i and $d_{i+1}^{b-1} \cdot d_{i+1}[0..2^n-2]$ is a prefix of B_{i+1}, this establishes the second case.

3. Next suppose xx is a substring of C_n where x can be factored into substrings $x = ywz$ such that y is a suffix of B_i, z is a prefix of B_{i+c}, and $w = B_{i+1} \cdots B_{i+c-1}$. This forces x to be of the form

$$x = d_i[k..] \cdot d_i^p \cdot B_{i+1} \cdots B_{i+c-1} \cdot d_{i+c}^{t-p-1} \cdot d_{i+c}[0..k-1-c]$$

where $p < t$ and $c \leq 2^{s-1}$. Note that $|x| = (2^n t - 1)c$. We have the previous restriction on p and c as if $p \geq t$ then $d_i[k..] \cdot d_i^p$ is not a suffix of B_i and if $c > 2^{s-1}$ then $|xx| > |C_n|$.

To see this, we consider the simpler case of

$$x = d_i[k..] \cdot d_i^p \cdot B_{i+1} \cdot d_{i+2}^{t-p-1} \cdot d_{i+2}[0..k-3]$$

and show that x can be parsed into the string

$$x = d_{i+2}[k-1..] \cdot d_{i+2}^p \cdot B_{i+3} \cdot d_{i+4}^{t-p-1} \cdot d_{i+4}[0..k-5].$$

We have that

$$\begin{aligned}
&d_i[k..] \cdot d_i^p \cdot B_{i+1} \cdot d_{i+2}^{t-p-1} \cdot d_{i+2}[0..k-3] \\
&= d_i[k..] \cdot 0 \cdot d_i[1..] \cdot d_i^{p-1} \cdot B_{i+1} \cdot d_{i+2}^{t-p-1} \cdot d_{i+2}[0..k-3] \\
&= d_{i+1}[k-1..] \cdot d_{i+1} \cdot d_i[1..] \cdot d_i^{p-2} \cdot B_{i+1} \cdot d_{i+2}^{t-p-1} \cdot d_{i+2}[0..k-3] \\
&\vdots \\
&= d_{i+1}[k-1..] \cdot \ d_{i+1}^{p-1} \cdot d_{i+1}[1..] \cdot d_{i+1}^{t} \cdot d_{i+2}^{t-p-1} \cdot d_{i+2}[0..k-3] \\
&= d_{i+1}[k-1..] \cdot d_{i+1}^{p} \cdot d_{i+1}[1..] \cdot d_{i+1}^{t-1} \cdot d_{i+2}^{t-p-1} \cdot d_{i+2}[0..k-3] \\
&= d_{i+1}[k-1..] \cdot d_{i+1}^{p} \cdot d_{i+2} \cdot d_{i+1}[1..] \cdot d_{i+1}^{t-2} \cdot d_{i+2}^{t-p-1} \cdot d_{i+2}[0..k-3] \\
&\vdots \\
&= d_{i+1}[k-1..] \cdot d_{i+1}^{p} \cdot d_{i+2}^{t-1} \cdot d_{i+1}[1..] \cdot 0 \cdot d_{i+2}^{t-p-2}[1..] \cdot d_{i+2}^{t-p-2} \cdot d_{i+2}[0..k-3] \\
&= d_{i+1}[k-1..] \cdot d_{i+1}^{p} \cdot d_{i+2}^{t} \cdot d_{i+2}[1..] \cdot d_{i+2}^{t-p-2} \cdot d_{i+2}[0..k-3] \\
&= d_{i+1}[k-1..] \cdot d_{i+1}^{p} \cdot B_{i+2} \cdot d_{i+3} \cdot d_{i+2}[1..] \cdot d_{i+2}^{t-p-3} \cdot d_{i+2}[0..k-3] \\
&\vdots \\
&= d_{i+1}[k-1..] \cdot d_{i+1}^{p} \cdot B_{i+2} \cdot d_{i+3}^{t-p-2} \cdot d_{i+2}[1..] \cdot \cdot d_{i+2}[0..k-3] \\
&= d_{i+1}[k-1..] \cdot d_{i+1}^{p} \cdot B_{i+2} \cdot d_{i+3}^{t-p-2} \cdot d_{i+2}[1..] \cdot 0 \cdot d_{i+2}[1..k-3] \\
&= d_{i+1}[k-1..] \cdot d_{i+1}^{p} \cdot B_{i+2} \cdot d_{i+3}^{t-p-1} \cdot d_{i+3}[0..k-4].
\end{aligned}$$

To see that xx is a substring of C_n, note then that x can be further parsed again as above to get that

$$x = d_{i+2}[k-2..] \cdot d_{i+2}^{p} \cdot B_{i+3} \cdot d_{i+4}^{t-p-1} \cdot d_{i+4}[0..k-5].$$

Hence it follows that

$$\begin{aligned} xx = &\underbrace{d_i[k..] \cdot d_i^p \cdot B_{i+1} \cdot d_{i+2}^{t-p-1} \cdot d_{i+2}[0..k-3]}_{x} \\ &\cdot \underbrace{d_{i+2}[k-2..] \cdot d_{i+2}^p \cdot B_{i+3} \cdot d_{i+4}^{t-p-1} \cdot d_{i+4}[0..k-5]}_{x} \\ &= d_i[k..] \cdot d_i^p \cdot B_{i+1} \cdot d_{i+2}^t \cdot B_{i+3} \cdot d_{i+4}^{t-p-1} \cdot d_{i+4}[0..k-5] \\ &= d_i[k..] \cdot d_i^p \cdot B_{i+1} \cdot B_{i+1} \cdot B_{i+3} \cdot d_{i+4}^{t-p-1} \cdot d_{i+4}[0..k-5] \end{aligned}$$

which is a substring of C_n.

The case where w contains more than one full block B_i follows from this case.

□

8.3.2 Remaining Calculations for Theorem 8.3.2

The following is the reasoning why Equation (8.7) from the proof of Theorem 8.3.2 gives us the upper bound of $S(C)$. We perform similar calculations as in the proof for Cases $1, 2$ and 4 to see this.

Case 1: n is a power of 2.

Suppose we are in Case 1. Let p_{n+2} be such that $C \upharpoonright m = \overline{C_{n+1}}p_{n+2}$. We have an automaton as in Case 1. It requires at most $|\overline{C_{n-1}}|+1+2n+2^n+2^{n+1}+|p_{n+2}|$ states. Thus

$$\frac{A(C \upharpoonright m)}{m} \leq \frac{|\overline{C_{n-1}}| + 1 + 2n + 2^n + 2^{n+1} + |p_{n+2}|}{|\overline{C_{n+1}}| + |p_{n+2}|}.$$

For $|p_{n+2}|$ long enough, it is more beneficial to loop in C_{n+2} instead of C_n and use an automaton in the style of Case 4 since in the worst case scenario, $n+2$ has the form $2^{s'}t'$. Such an automaton has at most $|\overline{C_n}| + 1 + n + 2^{n+1} + 2^{n+2}(\frac{n+2}{2})$

states.

So for $|p_{n+2}| \leq 2^n(n-1) - n + 2^{n+2}(\frac{n+2}{2})$, it holds that

$$\begin{aligned}\frac{A(C \upharpoonright m)}{m} &\leq \frac{|\overline{C_{n-1}}| + 1 + 2n + 2^n + 2^{n+1} + |p_{n+2}|}{|\overline{C_{n+1}}| + |p_{n+2}|} \\ &\leq \frac{|\overline{C_{n-1}}| + 1 + 2n - n + 2^n + 2^n(n-1) + 2^{n+1} + 2^{n+2}(\frac{n+2}{2})}{|\overline{C_{n+1}}| - n + 2^n(n-1) + 2^{n+2}(\frac{n+2}{2})} \\ &= \frac{|\overline{C_n}| + 1 + n + 2^{n+1} + 2^{n+2}(\frac{n+2}{2})}{|\overline{C_{n+1}}| - n + 2^n(n-1) + 2^{n+2}(\frac{n+2}{2})}.\end{aligned}$$

For $j \geq 1$ such that $2^n(n-1) - n + 2^{n+2}(\frac{n+2}{2}) \leq |p_{n+2}| + j < |C_{n+2}|$, we use an automaton from Case 4 and have that

$$\frac{A(C \upharpoonright m)}{m} \leq \frac{|\overline{C_n}| + 1 + n + 2^{n+1} + 2^{n+2}(\frac{n+2}{2})}{|\overline{C_{n+1}}| - n + 2^n(n-1) + 2^{n+2}(\frac{n+2}{2}) + j},$$

i.e. the number of states required remains constant. Furthermore

$$\lim_{n\to\infty} \frac{|\overline{C_n}| + 1 + n + 2^{n+1} + 2^{n+2}(\frac{n+2}{2})}{|\overline{C_{n+1}}| - n + 2^n(n-1) + 2^{n+2}(\frac{n+2}{2})} = \frac{4}{7} < \frac{2}{3}.$$

Case 2: $n+1$ is a power of 2

Suppose we are in Case 2. Let p_{n+2} be such that $C \upharpoonright m = \overline{C_{n+1}}p_{n+2}$. We have an automaton as in Case 2. It requires at most $|\overline{C_{n-1}}| + n + 2^n + 2^{n+1} + |p_{n+2}|$ states. Thus

$$\frac{A(C \upharpoonright m)}{m} \leq \frac{|\overline{C_{n-1}}| + n + 2^n + 2^{n+1} + |p_{n+2}|}{|\overline{C_{n+1}}| + |p_{n+2}|}.$$

For $|p_{n+2}|$ long enough, it is more beneficial to loop in C_{n+2} instead of C_n and use an automaton in the style of Case 1 as $n+2$ is odd and $n+1$ is a power of 2. As the final state will be contained in the second loop, such an automaton requires $|\overline{C_n}| + 2 + 2n + 2^{n+1} + 2^{n+2}$ states.

Thus for $|p_{n+2}| \leq 2 + n + 2^n(n-1) + 2^{n+2}$, it holds that

$$\begin{aligned}\frac{A(C \upharpoonright m)}{m} &\leq \frac{|\overline{C_{n-1}}| + n + 2^n + 2^{n+1} + |p_{n+2}|}{|\overline{C_{n+1}}| + |p_{n+2}|} \\ &\leq \frac{|\overline{C_{n-1}}| + 2 + n + n + 2^n + 2^n(n-1) + 2^{n+1} + 2^{n+2}}{|\overline{C_{n+1}}| + 2 + n + 2^n(n-1) + 2^{n+2}} \\ &= \frac{|\overline{C_n}| + 2 + 2n + 2^{n+1} + 2^{n+2}}{|\overline{C_{n+1}}| + 2 + n + 2^n(n-1) + 2^{n+2}}.\end{aligned}$$

For $j \geq 1$ such that $2 + n + 2^n(n-1) + 2^{n+2} \leq |p_{n+2}| + j < |C_{n+2}|$, we use an automaton from Case 1 and have that

$$\frac{A(C \upharpoonright m)}{m} \leq \frac{|\overline{C_n}| + 2 + 2n + 2^{n+1} + 2^{n+2}}{|\overline{C_{n+1}}| + 2 + n + 2^n(n-1) + 2^{n+2} + j},$$

i.e. the number of states required remains constant. Furthermore

$$\lim_{n\to\infty} \frac{|\overline{C_n}| + 2 + 2n + 2^{n+1} + 2^{n+2}}{|\overline{C_{n+1}}| + 2 + n + 2^n(n-1) + 2^{n+2}} = \frac{2}{5} < \frac{2}{3}.$$

Case 4: $n+1$ is even but not a power of 2

Suppose we are in Case 4, i.e. $n + 1 = 2^s t$ for some $s \geq 1$ and $t \geq 3$ odd. Let p_{n+2} be such that $C \upharpoonright m = \overline{C_{n+1}} p_{n+2}$. We have an automaton as in Case 4. It requires at most $|\overline{C_{n-1}}| + n + 2^n + 2^{n+1}t + |p_{n+2}|$ states.

Thus

$$\frac{A(C \upharpoonright m)}{m} \leq \frac{|\overline{C_{n-1}}| + n + 2^n + 2^{n+1}t + |p_{n+2}|}{|\overline{C_{n+1}}| + |p_{n+2}|}.$$

For $|p_{n+2}|$ long enough, it is more beneficial to loop in C_{n+2} instead of C_n and use an automaton in the style of Case 3 as $n+2$ is odd and $n+1$ is even but not power of 2. As the final state will be contained in the second loop, such an automaton requires $|\overline{C_n}| + n + 2^{n+1}t + 2^s + 2^{n+2}$ states. As $2^s \leq n/3$, we have that the number of states required is bounded above by $|\overline{C_n}| + 4n/3 + 2^{n+1}t + 2^{n+2}$.

So for $|p_{n+2}| \leq n/3 + 2^n(n-1) + 2^{n+2}$, it follows that

$$\begin{aligned}\frac{A(C \upharpoonright m)}{m} &\leq \frac{|\overline{C_{n-1}}| + n + 2^n + 2^{n+1}t + |p_{n+2}|}{|\overline{C_{n+1}}| + |p_{n+2}|} \\ &\leq \frac{|\overline{C_{n-1}}| + n + \frac{n}{3} + 2^n + 2^n(n-1) + 2^{n+1}t + 2^{n+2}}{|\overline{C_{n+1}}| + \frac{n}{3} + 2^n(n-1) + 2^{n+2}} \\ &= \frac{|\overline{C_n}| + \frac{4n}{3} + 2^{n+1}t + 2^{n+2}}{|\overline{C_{n+1}}| + \frac{n}{3} + 2^n(n-1) + 2^{n+2}}\end{aligned}$$

For $j \geq 1$ such that $n/3 + 2^n(n-1) + 2^{n+2} \leq |p_{n+2}| + j < |C_{n+2}|$, we use an automaton from Case 3 and have that

$$\frac{A(C \upharpoonright m)}{m} \leq \frac{|\overline{C_n}| + \frac{4n}{3} + 2^{n+1}t + 2^{n+2}}{|\overline{C_{n+1}}| + \frac{n}{3} + 2^n(n-1) + 2^{n+2} + j},$$

i.e. the number of states required remains constant. Furthermore

$$\begin{aligned}\lim_{n\to\infty} \frac{|\overline{C_n}| + \frac{4n}{3} + 2^{n+1}t + 2^{n+2}}{|\overline{C_{n+1}}| + \frac{n}{3} + 2^n(n-1) + 2^{n+2}} &\leq \lim_{n\to\infty} \frac{|\overline{C_n}| + \frac{4n}{3} + 2^{n+1}\frac{(n+1)}{2} + 2^{n+2}}{|\overline{C_{n+1}}| + \frac{n}{3} + 2^n(n-1) + 2^{n+2}} \\ &= \frac{3}{5} < \frac{2}{3}.\end{aligned}$$

8.4 Summary

In this chapter we explored whether normal sequences must have maximal automatic complexity. We answered this first in Theorem 8.2.2 by constructing a normal sequence T such that $S(T) \leq 1/2$. This separates automatic complexity from other finite-state based complexities where normal sequences have maximal complexity. We furthermore showed that $I(T) = 0$ which demonstrates that longer prefixes of T have arbitrarily small automatic complexity.

We further demonstrated the existence of a Champernowne sequence C and showed that $S(C) \leq 2/3$ in Theorem 8.3.2. This shows that even amongst a

restricted class of normal sequences in terms of how they must be constructed, normal sequences with non-maximal automatic complexity exist. We further showed that the size of the loops used in DFAs which uniquely accept C are restricted. Is there a limit to the number of beneficial loops we can use to ensure prefixes of C are uniquely accepted? We leave open the question of finding a lower bound for the value of $I(C)$.

Chapter 9

Concluding Remarks

In this thesis we offered new insights into the study of the complexity of sequences in various settings involving finite-state automata and transducers, along with popular compression algorithms. This was done to avoid the uncomputability of Kolmogorov complexity. This research had two main goals. Firstly, we aimed to develop new feasible notions of Bennett's Logical Depth and to examine their properties. Secondly, to identify whether normal sequences must have maximal complexity in certain compression-based and finite automata settings. We summarise below the core contributions of each chapter before discussing and identifying some of the limitations of this research and potential future areas of work.

To achieve our first goal, using Moser's framework for depth, four new notions of depth were studied. This began in Chapter 3 where we expanded upon Doty and Moser's infinitely often finite-state depth to develop an almost everywhere finite-state depth notion. This brings the study of finite-state depth more in line with Bennett's original version which is an almost everywhere notion. Like Doty and Moser's, our notion was based on the minimal length of an input string required by finite-state transducers to output a desired string. We demonstrated that normal sequences, that is FS-incompressible sequences, are not deep in our

notion and that a slow growth law applies, thus showing that our notion satisfies two of the fundamental laws of depth. By cleverly using repeated blocks of repetitions of finite-state random strings, we constructed an almost everywhere finite-state deep sequence. We furthermore showed that our notion differs from Doty and Moser's by constructing a sequence which is deep in their notion but not in ours.

In Chapter 4 we followed up by developing a notion of depth based on information lossless pushdown compressors. We showed that our pushdown depth satisfied all the fundamental properties of depth: ILUPDC-trivial and ILPDC-incompressible sequences are not deep and that a slow growth law holds. We also differentiated our pushdown depth from Doty and Moser's finite-state depth as follows. We first constructed a sequence which is deep in their finite-state depth notion and showed it is not pushdown deep. We then proved the existence of a sequence which is pushdown deep, and if it is deep in their finite-state depth notion, it must have a low finite-state depth level.

In Chapter 5, a depth notion based on the Lempel-Ziv 78 algorithm was presented. This was based on the difference in compression of the Lempel-Ziv 78 algorithm against all lossless finite-state compressors. We showed that sequences with a finite-state strong dimension of 0 and which are Lempel-Ziv 78 incompressible are not LZ-deep. With regards to normal sequences, we showed that there exists normal sequences which are LZ-deep. This automatically differentiates LZ depth from both version of finite-state depth studied in this thesis. We successfully separated LZ-depth from pushdown depth by presenting the existence of a sequence which is LZ-deep but not pushdown deep. We also showed that there exists sequences which are pushdown deep, and if they are LZ-deep, have a low LZ depth level.

The final new depth notion we developed was presented in Chapter 6. It was based on the difference between the minimal length of inputs required by

a finite-state transducer and by a pebble transducer to output the same string. We demonstrated that a slow growth law held, sequences which do not have a finite-state strong dimension of 0 are not deep and sequences which are PB-incompressible are not deep. By utilising the ability of certain pebble transducers to compute the pref function, we demonstrated the existence of a normal sequence which is pebble deep, thus differentiating it from both versions of finite-state depth. Similarly, by utilising the two-way nature of pebble transducers, we showed that there exists a sequence which is pebble deep, and if it is LZ-deep, has a low depth level. A preliminary investigation into comparing pebble depth with pushdown depth was also conducted, but a full separation was not achieved.

In Chapter 7 we performed the first analysis of the Prediction by Partial Matching family of compressor algorithms on normal sequences. The main result of this chapter was showing that there exists a Champernowne sequence C which can be fully compressed by the PPM* algorithm, that is, $R_{\text{PPM}^*}(C) = 0$. We furthermore showed that regardless of what upper bound of context length is used, normal sequences cannot be compressed by Bounded PPM. We also conducted a preliminary investigation into comparing Bounded PPM and PPM*. We showed that for every bound t, there exists a sequence S_t such that on long enough prefixes of S_t, PPM*'s output's length is roughly t times longer than PPM$_t$'s.

We concluded in Chapter 8 by exploring whether normal sequences must have maximal automatic complexity. Being a finite automata based complexity approach, as with many other finite-state based complexities, one might assume they must. Our investigation shows that the answer is no. We first showed that there exists a normal sequence which has infinitely many prefixes whose automatic complexity can get arbitrarily close to 0. We also showed that for the same sequence, the automatic complexity of its prefixes never goes above 1/2 for long enough prefixes. We furthermore studied the automatic complexity of a specific Champernowne sequence and showed that the automatic complexity of

its prefixes never goes above 2/3 for long enough prefixes. This illustrates that even amongst a restricted class of normal sequences in terms of how they must be constructed, normal sequences with non-maximal automatic complexity exist.

9.1 Limitations and Potential Future Work

As discussed in Section 2.3.1, many of the choices made when developing new notions of depth can seem arbitrary. In an effort to create a meaningful notion, i.e. one which satisfies all of or most of the fundamental properties of depth, sacrifices are made in terms of the succinctness of the notion to achieve this goal. In this section we shall identify some of the limitations of the depth notions developed in this thesis and identify areas where future research could be performed to improve them. We also identify some open questions which remain in this thesis.

For the depth notions developed, we tried to show that they satisfy a subset of the fundamental properties of Bennett's depth: deep sequences exist, random sequences are not deep, trivial sequences are not deep, and that a slow growth law holds. There is one other property of Bennett's depth we did not study: the property of deep sequences being *useful*. For each notion developed, can analogous results to Theorem 2.3.12 be found (which states that every Bennett deep sequence is weakly useful in the sense of Definition 2.3.11)? Other feasible depth notions previously studied satisfy analogous properties such as Doty and Moser's polynomial-time depth [57].

Weakly useful sequences are related to Turing reductions. Hence to explore the question, we must develop approaches to performing reductions in each of our settings. Similar ideas have been explored by Becher et al. in [12] (and further explored in [4]) in which a notion of *finite-state independence* is developed. Here, two sequences are independent if neither sequence aids in the compression of the other via finite-state compressors when provided as a secondary input. Hence,

one potential avenue for developing a finite-state usefulness notion would be to find two sequences which are not FS-deep but when the secondary sequence is provided as an auxiliary input, they become FS-deep in this altered setting. Here, both sequences could be considered 'useful' in helping to find minimal descriptions for prefixes of the other.

In Chapter 3, our almost everywhere finite-state depth notion has one clear potential area of improvement. For instance, one limitation we had was the need to switch the order of quantifiers in Definition 3.3.2 of a.e. FS-depth in contrast to the original i.o. notion's Definition 3.3.1. Ideally we would have a definition where the depth-level achieved would be the same for each k as opposed to varying based on the value of k being examined, i.e. a single α as opposed to differing α_k's. That is, we would like a notion where a sequence S is a.e. FS-deep if

$$(\exists \alpha > 0)(\forall k \in \mathbb{N})(\exists k' \in \mathbb{N})(\forall^\infty n \in \mathbb{N})\, D^k_{\mathrm{FS}}(S \upharpoonright n) - D^{k'}_{\mathrm{FS}}(S \upharpoonright n) \geq \alpha n.$$

This above limitation led to a second limitation for our notion of having to hard-code the property of FS-trivial sequences not being deep into the definition. The development of a such an almost everywhere notion is left open.

As discussed in Section 3.2.1, the finite-state dimension of sequences has been widely studied. We showed in Theorem 3.3.5 that sequences with a finite-state dimension of 1 are not deep. A question that could be explored in the future is whether for all $0 < s < 1$, does there exist a FS-deep sequence (in either our a.e. notion or Doty and Moser's i.o. notion) which has a finite-state dimension of s?

In Chapter 4, our pushdown depth was based on the difference between ILUPDCs and ILPDCs. While it mirrors Bennett's idea of comparing H^t against H, ILUPDCs are much more restricted than ordinary ILPDCs. Can a pushdown notion of depth be developed which compares ILPDCs against ILPDCs? Furthermore, the version of depth presented in this chapter is based on compressors.

An open question is whether or not a worthwhile notion can be developed which takes on the minimal description approach of Bennett's depth and the finite-state versions in Chapter 3. One obstacle to this currently is finding analogous results to Lemmas 3.3.9 and 3.3.12. While in the finite-state setting, if xy is a description for vw and x is a description for v via some FST, y can become a description for w by simply altering the start state in the FST. This is not as straightforward for pushdown transducers as the stack contents after reading x may play a vital role in the outputting of w while reading y.

With regards to Theorem 4.4.3, the factor of $1/2$ may be a fundamental constant in demonstrating a difference between the notions. In essence, an ILPDC must act similarly to an ILFST when pushing characters onto its stack or when it is not altering it, as in such cases the ILPDC has only one extra bit of information (the top of its stack) compared to an ILFST. The ILPDC only gains an advantage when it is popping bits from its stack to compare against its input. As the ILPDC can only pop as many characters from its stack that it pushed on, this gives rise to the $1/2$ term. As such, we leave open as to whether there exists a sequence S such that for $0 < \beta < 1/2$, PD-depth$(S) > 1/2$ while i.o. FS-depth$(S) < \beta$.

Other questions exist also. For instance, does there exist a pushdown deep normal sequence? Results from [11] show that normal sequences are not compressible by any ILUPDC. As such, based on our current definition of PD-depth, the existence of a PD-deep normal sequences hinges on whether or not there exists a normal sequence which is compressible by some ILPDC. This is currently an open question.

With regards to pebble depth in Chapter 6, depth was defined based on comparing finite-state transducers against pebble transducers. As discussed in Section 6.4, can a meaningful notion of depth be developed which compares the descriptional complexity of pebble transducers against other pebble transducers

as opposed to against finite-state transducers? A full comparison with PD-depth is also not performed. Both Theorems 5.5.2 and 6.3.11 and Remark 6.3.14 present sequences which are both PB-deep and PD-deep. Neither makes use of the ability of pebble transducers to compute the pref function. Perhaps this is the approach to take to identify a sequence which is PB-deep but not PD-deep? Similarly, does there exist a sequence which is LZ-deep but not PB-deep?

In this thesis, knowing that PB-incompressible sequences exist was sufficient for our desired results. Based on this, a result which could also potentially be expanded upon is Lemma 6.3.4 in which we showed that ML-random sequences are PB-incompressible. Just as it is known that a sequence is FS-incompressible if and only if the sequence is normal, a similar result which classifies what sequences are PB-incompressible is welcome.

Away from depth, potential avenues of research are evident Chapter 7. As discussed previously in the Chapter 1 and Section 3.2.2, normal sequences and incompressible sequences are equivalent in many variations of finite-state compressibility [11, 14, 35]. Similarly, it is known that for any sequence S which satisfies $\rho_{\mathrm{LZ}}(S) = 1$, S is normal [97]. While in Theorem 7.3.8 it was shown that there exists a normal sequence C such that $R_{\mathrm{PPM}^*}(C) = 0$, a similar result which answers the question as to whether or not for for any sequence C' which satisfies $\rho_{\mathrm{PPM}^*}(C') \geq 1$, must C' be normal is open. Is it possible to take a non-normal sequence and add in 'adversarial' bits periodically in a way which maintains non-normality but results in the sequence being PPM*-incompressible?

In Section 7.4 we showed that one can construct a simple sequence S^t such that PPM*'s output on prefixes of S^t is t times longer than PPM$_t$'s output. However, $R_{\mathrm{PPM}^*}(S^t) = R_{\mathrm{PPM}^*}(S^t) = 0$, meaning that S^t is highly compressible by both compressors. For a fixed t, does there in fact exist a sequence S' such that $R_{\mathrm{PPM}_t}(S') < \rho_{\mathrm{PPM}^*}(S')$, meaning that PPM$_t$ compresses S' better than PPM*? Furthermore, does there exist a sequence S'' such that for all t, $R_{\mathrm{PPM}_t}(S'') <$

$\rho_{\mathrm{PPM}^*}(S'')$? In this case, S'' must be cleverly constructed such that regardless of the maximum context length chosen, Bounded PPM always performs better than PPM*. A comparison between PPM and other compression algorithms, such as in the vein of Mayordomo, Moser and Perifel's work in [109] can still be undertaken. For instance, Theorem 7.3.8 demonstrates the existence of a sequence C such that $R_{\mathrm{PPM}^*}(C) = 0$ but $\rho_{\mathrm{LZ}}(C) = 1$. Does there exist a sequence such that the converse holds? The question of separating PPM from pushdown compression is also open.

In Chapter 8, we asked the question as to whether or not there exists a normal sequence T such that its upper automatic complexity ratio satisfies $S(T) < 1$. We answer this question positively in Theorems 8.2.2 and 8.3.2. However, this question may be redundant in the sense that it is currently unknown as to whether or not the set $\{T \in \{0,1\}^\omega \mid S(T) = 1\}$ is non-empty. The answer to this is not obvious since it holds that for almost every string x, $A(x) \leq 3/4 + O(\sqrt{|x|}\log|x|)$ as stated in Equation 8.1.

Alternatively, we could consider the non-deterministic approach as first introduced by Hyde [83] (denoted by A_N). In this setting, it is known that every string x satisfies $A_N(x) \leq \left\lfloor \frac{|x|}{2} \right\rfloor + 1$ [83, 84]. It is also known that this bound is tight [84]. Similarly, it is known that for all $\varepsilon > 0$, for almost every string x it holds that $A_N(x) \geq |x|/(2+\varepsilon)$ [89]. This means that almost every string x has a non-deterministic automatic complexity close to $1/2$. While we were able to show that there exists a normal sequence T such that $S(T) \leq 1/2$ in Theorem 8.2.2, we would be interested in knowing whether there exists a normal sequence T' such that $S(T') < 1/2$, i.e. its automatic complexity, and hence its non-deterministic automatic complexity, lies outside of this 'close to $1/2$' range.

Bibliography

[1] Pilar Albert, Elvira Mayordomo, and Philippe Moser. Bounded pushdown dimension vs Lempel Ziv information density. In Adam R. Day, Michael R. Fellows, Noam Greenberg, Bakhadyr Khoussainov, Alexander G. Melnikov, and Frances A. Rosamond, editors, *Computability and Complexity - Essays Dedicated to Rodney G. Downey on the Occasion of His 60th Birthday*, volume 10010 of *Lecture Notes in Computer Science*, pages 95–114. Springer, 2017. doi:10.1007/978-3-319-50062-1_7.

[2] Pilar Albert, Elvira Mayordomo, Philippe Moser, and Sylvain Perifel. Pushdown compression. In Susanne Albers and Pascal Weil, editors, *STACS 2008, 25th Annual Symposium on Theoretical Aspects of Computer Science, Bordeaux, France, February 21-23, 2008, Proceedings*, volume 1 of *LIPIcs*, pages 39–48. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2008. doi:10.4230/LIPIcs.STACS.2008.1332.

[3] Eric Allender, Harry Buhrman, Michal Koucký, Dieter van Melkebeek, and Detlef Ronneburger. Power from random strings. *SIAM J. Comput.*, 35(6):1467–1493, 2006. doi:10.1137/050628994.

[4] Nicolás Alvarez, Verónica Becher, and Olivier Carton. Finite-state independence and normal sequences. *J. Comput. Syst. Sci.*, 103:1–17, 2019. doi:10.1016/j.jcss.2019.02.001.

[5] Luis Filipe Coelho Antunes, Lance Fortnow, Dieter van Melkebeek, and N. V. Vinodchandran. Computational depth: Concept and applications. *Theor. Comput. Sci.*, 354(3):391–404, 2006. doi:10.1016/j.tcs.2005.11.033.

[6] Luis Filipe Coelho Antunes, Andre Souto, and Paul M. B. Vitányi. On the rate of decrease in logical depth. *Theor. Comput. Sci.*, 702:60–64, 2017. (Corrigendum: see [144]). doi:10.1016/j.tcs.2017.08.012.

[7] Krishna B. Athreya, John M. Hitchcock, Jack H. Lutz, and Elvira Mayordomo. Effective strong dimension in algorithmic information and computational complexity. *SIAM J. Comput.*, 37(3):671–705, 2007. doi:10.1137/S0097539703446912.

[8] J.M. Barzdin. Complexity of programs to determine whether natural numbers not greater than n belong to a recursively enumerable set. *Soviet Math. Dokl*, 9:1251–1254, 1968.

[9] Bruno Bauwens, Anton Makhlin, Nikolai K. Vereshchagin, and Marius Zimand. Short lists with short programs in short time. *Comput. Complex.*, 27(1):31–61, 2018. doi:10.1007/s00037-017-0154-2.

[10] Bruno Bauwens and Marius Zimand. Linear list-approximation for short programs (or the power of a few random bits). In *IEEE 29th Conference on Computational Complexity, CCC 2014, Vancouver, BC, Canada, June 11-13, 2014*, pages 241–247. IEEE Computer Society, 2014. doi:10.1109/CCC.2014.32.

[11] Verónica Becher, Olivier Carton, and Pablo Ariel Heiber. Normality and automata. *J. Comput. Syst. Sci.*, 81(8):1592–1613, 2015. doi:10.1016/j.jcss.2015.04.007.

[12] Verónica Becher, Olivier Carton, and Pablo Ariel Heiber. Finite-state independence. *Theory Comput. Syst.*, 62(7):1555–1572, 2018. doi:10.1007/s00224-017-9821-6.

[13] Verónica Becher and Pablo Ariel Heiber. A linearly computable measure of string complexity. *Theor. Comput. Sci.*, 438:62–73, 2012. doi:10.1016/j.tcs.2012.03.007.

[14] Verónica Becher and Pablo Ariel Heiber. Normal numbers and finite automata. *Theor. Comput. Sci.*, 477:109–116, 2013. doi:10.1016/j.tcs.2013.01.019.

[15] Timothy C. Bell, John G. Cleary, and Ian H. Witten. *Text Compression*. Prentice-Hall, Inc., Upper Saddle River, New Jersey, USA, 1990. ISBN: 0-13-911991-4.

[16] Charles H. Bennett. On random and hard-to-describe numbers. Technical Report RC 7483, IBM, Yorktown Heights, New York, USA, May 1979. doi:doi=10.1.1.27.3492.

[17] Charles H. Bennett. Logical depth and physical complexity. *The Universal Turing Machine, A Half-Century Survey*, pages 227–257, 1988.

[18] Jan Åberg, Yuri M. Shtarkov, and Ben J.M. Smeets. Estimation of escape probabilities for PPM based on universal source coding theory. In *Proceedings of IEEE International Symposium on Information Theory*, page 65, 1997. doi:10.1109/ISIT.1997.612980.

[19] Jean Berstel. Axel Thue's papers on repetitions in words: a translation. Publications du Laboratoire de Combinatoire et d'Informatique Mathématique, Université du Québec à Montréal, Canada, 1995.

[20] Laurent Bienvenu, Valentino Delle Rose, and Wolfgang Merkle. Relativized depth. *CoRR*, 2021. arXiv:2112.04451.

[21] Charles Bloom. Solving the problems of context modelling, 1998. (informal publication, Accessed: May 11, 2021). URL: https://www.cbloom.com/papers/ppmz.pdf.

[22] Manuel Blum and Carl Hewitt. Automata on a 2-dimensional tape. In *8th Annual Symposium on Switching and Automata Theory, Austin, Texas, USA, October 18-20, 1967*, pages 155–160. IEEE Computer Society, 1967. doi:10.1109/FOCS.1967.6.

[23] Mikołaj Bojańczyk. Polyregular functions. *CoRR*, 2018. arXiv:1810.08760.

[24] Mikołaj Bojańczyk, Sandra Kiefer, and Nathan Lhote. String-to-string interpretations with polynomial-size output. In Christel Baier, Ioannis Chatzigiannakis, Paola Flocchini, and Stefano Leonardi, editors, *46th International Colloquium on Automata, Languages, and Programming, ICALP 2019, July 9-12, 2019, Patras, Greece*, volume 132 of *LIPIcs*, pages 106:1–106:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.ICALP.2019.106.

[25] Émile Borel. Les probabilités dénombrables et leurs applications arithmétiques. *Rendiconti del Circolo Matematico di Palermo*, 27(1):247–271, 1909. doi:10.1007/BF03019651.

[26] Chris Bourke, John M. Hitchcock, and N. V. Vinodchandran. Entropy rates and finite-state dimension. *Theor. Comput. Sci.*, 349(3):392–406, 2005. doi:10.1016/j.tcs.2005.09.040.

[27] Allen H. Brady. The determination of the value of Radó's noncomputable function $\sum(k)$ for four-state turing machines. *Math. Comput.*, 40(162):647–665, 1983. doi:10.2307/2007539.

[28] Michael Burrows and David Wheeler. A block-sorting lossless data compression algorithm, 1994.

[29] Cristian S. Calude, Sanjay Jain, Wolfgang Merkle, and Frank Stephan. Searching for shortest and least programs. *Theor. Comput. Sci.*, 807:114–127, 2020. doi:10.1016/j.tcs.2019.10.011.

[30] Cristian S. Calude and André Nies. Chaitin Ω numbers and strong reducibilities. *J. Univers. Comput. Sci.*, 3(11):1162–1166, 1997. doi:10.3217/jucs-003-11-1162.

[31] Cristian S. Calude, Kai Salomaa, and Tania Roblot. Finite state complexity. *Theor. Comput. Sci.*, 412(41):5668–5677, 2011. doi:10.1016/j.tcs.2011.06.021.

[32] Cristian S. Calude, Kai Salomaa, and Tania Roblot. State-size hierarchy for finite-state complexity. *Int. J. Found. Comput. Sci.*, 23(1):37–50, 2012. doi:10.1142/S0129054112400035.

[33] Cristian S. Calude and Ludwig Staiger. Liouville, computable, borel normal and martin-löf random numbers. *Theory Comput. Syst.*, 62(7):1573–1585, 2018. doi:10.1007/s00224-017-9767-8.

[34] Cristian S. Calude, Ludwig Staiger, and Frank Stephan. Finite state incompressible infinite sequences. *Inf. Comput.*, 247:23–36, 2016. doi:10.1016/j.ic.2015.11.003.

[35] Olivier Carton and Pablo Ariel Heiber. Normality and two-way automata. *Inf. Comput.*, 241:264–276, 2015. doi:10.1016/j.ic.2015.02.001.

[36] Gregory J. Chaitin. On the length of programs for computing finite binary sequences. *J. ACM*, 13(4):547–569, 1966. doi:10.1145/321356.321363.

[37] Gregory J. Chaitin. On the length of programs for computing finite binary sequences: statistical considerations. *J. ACM*, 16(1):145–159, 1969. doi:10.1145/321495.321506.

[38] Gregory J. Chaitin. A theory of program size formally identical to information theory. *J. ACM*, 22(3):329–340, 1975. doi:10.1145/321892.321894.

[39] Gregory J. Chaitin, Asat Arslanov, and Cristian S. Calude. Program-size complexity computes the halting problem. *Bull. EATCS*, 57, 1995.

[40] D. G. Champernowne. The construction of decimals normal in the scale of ten. *J. Lond. Math. Soc.*, s1-8(4):254–260, 1933. doi:10.1112/jlms/s1-8.4.254.

[41] Moses Charikar, Eric Lehman, Ding Liu, Rina Panigrahy, Manoj Prabhakaran, April Rasala, Amit Sahai, and Abhi Shelat. Approximating the smallest grammar: Kolmogorov complexity in natural models. In John H. Reif, editor, *Proceedings on 34th Annual ACM Symposium on Theory of Computing, May 19-21, 2002, Montréal, Québec, Canada*, pages 792–801. ACM, 2002. doi:10.1145/509907.510021.

[42] Xin Chen, Brent Francia, Ming Li, Brian McKinnon, and Amit Seker. Shared information and program plagiarism detection. *IEEE Trans. Inf. Theory*, 50(7):1545–1551, 2004. doi:10.1109/TIT.2004.830793.

[43] Rudi Cilibrasi and Paul M. B. Vitányi. Clustering by compression. *IEEE Trans. Inf. Theory*, 51(4):1523–1545, 2005. doi:10.1109/TIT.2005.844059.

[44] Rudi Cilibrasi, Paul M. B. Vitányi, and Ronald de Wolf. Algorithmic clustering of music based on string compression. *Comput. Music. J.*, 28(4):49–67, 2004. doi:10.1162/0148926042728449.

[45] John G. Cleary and Ian H. Witten. Data compression using adaptive coding and partial string matching. *IEEE Trans. Commun.*, 32(4):396–402, 1984. doi:10.1109/TCOM.1984.1096090.

[46] John G. Cleary, Ian H. Witten, and William J. Teahan. Unbounded length contexts for PPM. *Comput. J.*, 40(2/3):67–75, 1997. doi:10.1109/DCC.1995.515495.

[47] Arthur H. Copeland and Paul Erdös. Note on normal numbers. *Bull. Amer. Math. Soc*, 52(10):857–861, October 1946. doi:10.1090/s0002-9904-1946-08657-7.

[48] Cristina Costa-Santos, João Bernardes, Paul M. B. Vitányi, and Luis Filipe Coelho Antunes. Clustering fetal heart rate tracings by compression. In *19th IEEE International Symposium on Computer-Based Medical Systems (CBMS 2006), 22-23 June 2006, Salt Lake City, Utah, USA*, pages 685–690. IEEE Computer Society, 2006. doi:10.1109/CBMS.2006.68.

[49] Jack Jie Dai, James I. Lathrop, Jack H. Lutz, and Elvira Mayordomo. Finite-state dimension. *Theor. Comput. Sci.*, 310(1-3):1–33, 2004. doi:10.1016/S0304-3975(03)00244-5.

[50] Robert P. Daley. Noncomplex sequences: Characterizations and examples. *J. Symbolic Logic*, 41(3):626–638, 1976. doi:10.2307/2272040.

[51] Nicolaas G. de Bruijn. A combinatorial problem. In *Proc. Koninklijke Nederlandse Academie van Wetenschappen*, volume 49, pages 758–764, 1946.

[52] Nicolaas G. de Bruijn. Acknowledgement of priority to C. Flye Sainte-Marie on the counting of circular arrangements of 2^n zeros and ones that show each n-letter word exactly once. T.H.-Report 75-WSK-06, Department of Mathematics, Technological University Eindhoven, The Netherlands, 1975. URL: `https://research.tue.nl/en/publications/acknowledgement-of-priority-to-c-flye-sainte-marie-on-the-countin`.

[53] A. de Rivière. Question nr. 48. In *L'Intermédiaire des Mathématiciens*, volume 1, pages 19–20, 1894.

[54] Jean-Paul Delahaye and Hector Zenil. Numerical evaluation of algorithmic complexity for short strings: A glance into the innermost structure of randomness. *Appl. Math. Comput.*, 219(1):63–77, 2012. `doi:10.1016/j.amc.2011.10.006`.

[55] David Doty, Jack H. Lutz, and Satyadev Nandakumar. Finite-state dimension and real arithmetic. *Inf. Comput.*, 205(11):1640–1651, 2007. `doi:10.1016/j.ic.2007.05.003`.

[56] David Doty and Philippe Moser. Finite-state dimension and lossy decompressors. *CoRR*, 2006. `arXiv:cs/0609096`.

[57] David Doty and Philippe Moser. Feasible depth. In *Computation and Logic in the Real World, Third Conference on Computability in Europe, CiE 2007, Siena, Italy, June 18-23, 2007, Proceedings*, volume 4497 of *Lecture Notes in Computer Science*, pages 228–237. Springer, 2007. `doi:10.1007/978-3-540-73001-9\_24`.

[58] David Doty and Jared Nichols. Pushdown dimension. *Theor. Comput. Sci.*, 381(1-3):105–123, 2007. `doi:10.1016/j.tcs.2007.04.005`.

[59] Gaëtan Douéneau-Tabot. Pebble transducers with unary output. In *46th International Symposium on Mathematical Foundations of Computer Science, MFCS 2021, August 23-27, 2021, Tallinn, Estonia*, volume 202 of *LIPIcs*, pages 40:1–40:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.MFCS.2021.40.

[60] Gaëtan Douéneau-Tabot, Emmanuel Filiot, and Paul Gastin. Register transducers are marble transducers. In Javier Esparza and Daniel Král', editors, *45th International Symposium on Mathematical Foundations of Computer Science, MFCS 2020, August 24-28, 2020, Prague, Czech Republic*, volume 170 of *LIPIcs*, pages 29:1–29:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.MFCS.2020.29.

[61] Rodney G. Downey and Denis Roman Hirschfeldt. *Algorithmic Randomness and Complexity*. Springer, New York ; London, 2010. ISBN: 978-0-387-95567-4. doi:10.1007/978-0-387-68441-3.

[62] Rodney G. Downey, Michael McInerney, and Keng Meng Ng. Lowness and logical depth. *Theor. Comput. Sci.*, 702:23–33, 2017. doi:10.1016/j.tcs.2017.08.010.

[63] Joost Engelfriet. Two-way pebble transducers for partial functions and their composition. *Acta Informatica*, 52(7-8):559–571, 2015. doi:10.1007/s00236-015-0224-3.

[64] Joost Engelfriet, Hendrik Jan Hoogeboom, and Bart Samwel. XML navigation and transformation by tree-walking automata and transducers with visible and invisible pebbles. *Theor. Comput. Sci.*, 850:40–97, 2021. doi:10.1016/j.tcs.2020.10.030.

[65] Joost Engelfriet and Sebastian Maneth. Two-way finite state transducers with nested pebbles. In Krzysztof Diks and Wojciech, editors, *Mathemati-*

cal Foundations of Computer Science 2002, 27th International Symposium, MFCS 2002, Warsaw, Poland, August 26-30, 2002, Proceedings, volume 2420 of *Lecture Notes in Computer Science*, pages 234–244. Springer, 2002. doi:10.1007/3-540-45687-2_19.

[66] Stephen A. Fenner, Jack H. Lutz, Elvira Mayordomo, and Patrick Reardon. Weakly useful sequences. *Inf. Comput.*, 197(1-2):41–54, 2005. doi:10.1016/j.ic.2005.01.001.

[67] Lester R. Ford Jr. A cyclic arrangement of m-tuples, report no. *Reprot P-1071, RAND Corp*, 1957.

[68] Harold Fredricksen. A survey of full length nonlinear shift register cycle algorithms. *SIAM review*, 24(2):195–221, 1982. doi:10.1137/1024041.

[69] Harold Fredricksen and Irving J. Kessler. An algorithm for generating necklaces of beads in two colors. *Discret. Math.*, 61(2-3):181–188, 1986. doi:10.1016/0012-365X(86)90089-0.

[70] Harold Fredricksen and James Maiorana. Necklaces of beads in k colors and k-ary de Bruijn sequences. *Discret. Math.*, 23(3):207–210, 1978. doi:10.1016/0012-365X(78)90002-X.

[71] Cédric Gaucherel. Ecosystem complexity through the lens of logical depth: Capturing ecosystem individuality. *Biol. Theory*, 9(4):440–451, March 2014. doi:10.1007/s13752-014-0162-2.

[72] Viliam Geffert and L'ubomíra Ištoňová. Translation from classical two-way automata to pebble two-way automata. *RAIRO Theor. Informatics Appl.*, 44(4):507–523, 2010. doi:10.1051/ita/2011001.

[73] Ian Glaister and Jeffrey O. Shallit. Automaticity III: polynomial automaticity and context-free languages. *Comput. Complex.*, 7(4):371–387, 1998. doi:10.1007/s000370050016.

[74] Noa Globerman and David Harel. Complexity results for two-way and multi-pebble automata and their logics. *Theor. Comput. Sci.*, 169(2):161–184, 1996. doi:10.1016/S0304-3975(96)00119-3.

[75] I.J. Good. Normal recurring decimals. *LMS*, s1-21(3):167–169, 1946. doi:10.1112/jlms/s1-21.3.167.

[76] John M. Hitchcock. Fractal dimension and logarithmic loss unpredictability. *Theor. Comput. Sci.*, 304(1-3):431–441, 2003. doi:10.1016/S0304-3975(03)00138-5.

[77] Rupert Hölzl, Thorsten Kräling, and Wolfgang Merkle. Time-bounded Kolmogorov complexity and Solovay functions. *Theory Comput. Syst.*, 52(1):80–94, 2013. doi:10.1007/s00224-012-9413-4.

[78] Paul G. Howard and Jeffrey Scott Vitter. Analysis of arithmetic coding for data compression. *Inf. Process. Manag.*, 28(6):749–764, 1992. doi:10.1016/0306-4573(92)90066-9.

[79] Paul G. Howard and Jeffrey Scott Vitter. Practical implementations of arithmetic coding. *Image and Text Compression - The Kluwer International Series in Engineering and Computer Science (Communication and Information Theory)*, pages 85–112, 1992. doi:10.1007/978-1-4615-3596-6_4.

[80] Paul G. Howard and Jeffrey Scott Vitter. Arithmetic coding for data compression. In *Encyclopedia of Algorithms*, pages 145–150. 2016. doi:10.1007/978-1-4939-2864-4_34.

[81] David. A. Huffman. A method for the construction of minimum-redundancy codes. *Proceedings of the IRE*, 40(9):1098–1101, 1952. doi:10.1109/JRPROC.1952.273898.

[82] David A. Huffman. Canonical forms for information-lossless finite-state logical machines. *IRE Trans. Inf. Theory*, 5(5):41–59, 1959. doi:10.1109/TIT.1959.1057537.

[83] Kayleigh Hyde. Nondeterministic finite state complexity. Master's thesis, University of Hawai'i at Mānoa, 2013. Accessed: April 20, 2021. URL: http://hdl.handle.net/10125/29507.

[84] Kayleigh Hyde and Bjørn Kjos-Hanssen. Nondeterministic automatic complexity of overlap-free and almost square-free words. *Electron. J. Comb.*, 22(3):P3.22, 2015. doi:10.37236/4851.

[85] Glen G. Langdon Jr. An introduction to arithmetic coding. *IBM J. Res. Dev.*, 28(2):135–149, 1984. doi:10.1147/rd.282.0135.

[86] David W. Juedes, James I. Lathrop, and Jack H. Lutz. Computational depth and reducibility. *Theor. Comput. Sci.*, 132(2):37–70, 1994. doi:10.1016/0304-3975(94)00014-X.

[87] Bjørn Kjos-Hanssen. Automatic complexity of Fibonacci and Tribonacci words. *Discret. Appl. Math.*, 289:446 – 454, 2021. doi:10.1016/j.dam.2020.10.014.

[88] Bjørn Kjos-Hanssen. Automatic complexity of shift register sequences. *Discret. Math.*, 341(9):2409–2417, 2018. doi:10.1016/j.disc.2018.05.015.

[89] Bjørn Kjos-Hanssen. An incompressibility theorem for automatic complexity. *Forum of Mathematics, Sigma*, 9:E62, 2021. doi:10.1017/fms.2021.58.

[90] Donald E. Knuth. *The Art of Computer Programming: Volume 4A, Combinatorial Algorithms, Part* 1. Addison-Wesley, Upper Saddle River, New Jersey, USA, 2011. ISBN: 0-201-03804-8.

[91] Ker-I Ko. On the notion of infinite pseudorandom sequences. *Theor. Comput. Sci.*, 48(3):9–33, 1986. doi:10.1016/0304-3975(86)90081-2.

[92] Zvi Kohavi. *Switching and Finite Automata Theory, Second Edition.* McGraw-Hill, New York, 1978.

[93] A.N. Kolmogorov. Three approaches to the quantitative definition of information. *Problems Inform. Transmission*, 1(1):1–7, 1965. doi:10.1080/00207166808803030.

[94] S. Rao Kosaraju and Giovanni Manzini. Compression of low entropy strings with Lempel-Ziv algorithms. *SIAM J. Comput.*, 29(3):893–911, 1999. doi:10.1137/S0097539797331105.

[95] Alexander Kozachinskiy and Alexander Shen. Automatic Kolmogorov complexity, normality, and finite-state dimension revisited. *J. Comput. Syst. Sci.*, 118:75–107, 2021. doi:10.1016/j.jcss.2020.12.003.

[96] James I. Lathrop and Jack H. Lutz. Recursive computational depth. *Inf. Comput.*, 153(1):139–172, 1999. doi:10.1006/inco.1999.2794.

[97] James I. Lathrop and Martin Strauss. A universal upper bound on the performance of the Lempel-Ziv algorithm on maliciously-constructed data. In Bruno Carpentieri, Alfredo De Santis, Ugo Vaccaro, and James A. Storer, editors, *Compression and Complexity of Sequences 1997, Positano, Amalfitan Coast, Salerno, Italy, June 11-13, 1997, Proceedings*, pages 123–135. IEEE, 1997. doi:10.1109/SEQUEN.1997.666909.

[98] Abraham Lempel and Jacob Ziv. On the complexity of finite sequences. *IEEE Trans. Inf. Theory*, 22(1):75–81, 1976. doi:10.1109/TIT.1976.1055501.

[99] Leonid A. Levin. On the notion of a random sequence. *Soviet Math. Dokl*, 14(5):1413–1416, 1973.

[100] Leonid A. Levin. Laws of information conservation (non-growth) and aspects of the foundation of probability theory. *Problems Inform. Transmission*, 10(3):206–210, 1974. URL: http://mi.mathnet.ru/ppi1039.

[101] Leonid A. Levin. Randomness conservation inequalities; information and independence in mathematical theories. *Inf. Control.*, 61(1):15–37, 1984. doi:10.1016/S0019-9958(84)80060-1.

[102] Nathan Lhote. Pebble minimization of polyregular functions. In *LICS '20: 35th Annual ACM/IEEE Symposium on Logic in Computer Science, Saarbrücken, Germany, July 8-11, 2020*, pages 703–712. ACM, 2020. doi:10.1145/3373718.3394804.

[103] Ming Li, Xin Chen, Xin Li, Bin Ma, and Paul M. B. Vitányi. The similarity metric. *IEEE Trans. Inf. Theory*, 50(12):3250–3264, 2004. doi:10.1109/TIT.2004.838101.

[104] Ming Li and Paul M. B. Vitányi. *An Introduction to Kolmogorov Complexity and Its Applications, Third Edition*. Texts in Computer Science. Springer, 2008. ISBN: 978-0-387-33998-6. doi:10.1007/978-0-387-49820-1.

[105] Jack H. Lutz. Almost everywhere high nonuniform complexity. *J. Comput. Syst. Sci.*, 44(2):220–258, 1992. doi:10.1016/0022-0000(92)90020-J.

[106] Jack H. Lutz. Resource bounded measure. *CoRR*, 2011. arXiv:1101.5455.

[107] Monroe H. Martin. A problem in arrangements. *Bull. Amer. Math. Soc.*, 40(12):859–864, 1934. doi:10.1090/S0002-9904-1934-05988-3.

[108] Per Martin-Löf. The definition of random sequences. *Inform. Control*, 9(6):602–619, 1966. doi:10.1016/S0019-9958(66)80018-9.

[109] Elvira Mayordomo, Philippe Moser, and Sylvain Perifel. Polylog space compression, pushdown compression, and Lempel-Ziv are incomparable. *Theory Comput. Syst.*, 48(4):731–766, 2011. doi:10.1007/s00224-010-9267-6.

[110] Tova Milo, Dan Suciu, and Victor Vianu. Typechecking for XML transformers. *J. Comput. Syst. Sci.*, 66(1):66–97, 2003. doi:10.1016/S0022-0000(02)00030-2.

[111] Alistair Moffat. Implementing the PPM data compression scheme. *IEEE Trans. Commun.*, 38(11):1917–1921, 1990. doi:10.1109/26.61469.

[112] Philippe Moser. On the polynomial depth of various sets of random strings. *Theor. Comput. Sci.*, 477:96–108, 2013. doi:10.1016/j.tcs.2012.10.045.

[113] Philippe Moser. Polylog depth, highness and lowness for E. *Inf. Comput.*, 271:104483, 2020. doi:10.1016/j.ic.2019.104483.

[114] Philippe Moser and Frank Stephan. Depth, highness and DNR degrees. *Discret. Math. Theor. Comput. Sci.*, 19(4), 2017. doi:10.23638/DMTCS-19-4-2.

[115] Philippe Moser and Frank Stephan. Limit-depth and DNR degrees. *Inf. Process. Lett.*, 135:36–40, 2018. doi:10.1016/j.ipl.2018.02.015.

[116] Satyadev Nandakumar and Santhosh Kumar Vangapelli. Normality and finite-state dimension of Liouville numbers. *Theory Comput. Syst.*, 58(3):392–402, 2016. doi:10.1007/s00224-014-9554-8.

[117] Lê Thành Dung Nguyên, Camille Noûs, and Pierre Pradic. Comparison-free polyregular functions. In Nikhil Bansal, Emanuela Merelli, and James Worrell, editors, *48th International Colloquium on Automata, Languages, and Programming, ICALP 2021, July 12-16, 2021, Glasgow, Scotland (Virtual Conference)*, volume 198 of *LIPIcs*, pages 139:1–139:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.ICALP.2021.139.

[118] André Nies. *Computability and Randomness*. Oxford logic guides. Oxford University Press, Oxford ; New York, 2009. ISBN: 978-0-19-923076-1. doi:10.1093/acprof:oso/9780199230761.001.0001.

[119] Larry A. Pierce and Paul C. Shields. Sequences incompressible by SLZ (LZW), yet fully compressible by ULZ. In Ingo Althöfer, Ning Cai, Gunter Dueck, Levon Khachatrian, Mark S. Pinsker, Andras Sárközy, Ingo Wegener, and Zhen Zhang, editors, *Numbers, Information and Complexity*, pages 385–390. Springer, 2000. doi:10.1007/978-1-4757-6048-4_32.

[120] Carl Pomerance, John Michael Robson, and Jeffrey O. Shallit. Automaticity II: descriptional complexity in the unary case. *Theor. Comput. Sci.*, 180(1-2):181–201, 1997. doi:10.1016/S0304-3975(96)00189-2.

[121] Michael O. Rabin and Dana S. Scott. Finite automata and their decision problems. *IBM J. Res. Dev.*, 3(2):114–125, 1959. doi:10.1147/rd.32.0114.

[122] Tibor Radó. On non-computable functions. *Bell Syst. Tech. J.*, 41(3):877–884, May 1962. doi:10.1002/j.1538-7305.1962.tb00480.x.

[123] Anthony Ralston. de Bruijn sequences-a model example of the interaction of discrete mathematics and computer science. *Mathematics Magazine*, 55(3):131, May 1982. doi:10.2307/2690079.

[124] Joseph J. Rotman. *An Introduction to the Theory of Groups*. Graduate Texts in Mathematics. Springer, New York, 1995. ISBN: 978-1-4612-8686-8. doi:10.1007/978-1-4612-4176-8.

[125] Camille Flye Sainte-Marie. Solution to question nr. 48. In *L'Intermédiaire des Mathématiciens*, volume 1, pages 107–110, 1894.

[126] Claus-Peter Schnorr. A unified approach to the definition of random sequences. *Math. Syst. Theory*, 5(3):246–258, 1971. doi:10.1007/BF01694181.

[127] Claus-Peter Schnorr. Process complexity and effective random tests. *J. Comput. Syst. Sci.*, 7(4):376–388, 1973. doi:10.1016/S0022-0000(73)80030-3.

[128] Claus-Peter Schnorr and H. Stimm. Endliche Automaten und Zufallsfolgen. *Acta Inf.*, 1:345–359, 1972. doi:10.1007/BF00289514.

[129] Jeffrey O. Shallit and Yuri Breitbart. Automaticity I: properties of a measure of descriptional complexity. *J. Comput. Syst. Sci.*, 53(1):10–25, 1996. doi:10.1006/jcss.1996.0046.

[130] Jeffrey O. Shallit and Ming-Wei Wang. Automatic complexity of strings. *J. Autom. Lang. Comb.*, 6(4):537–554, 2001. doi:10.25596/jalc-2001-537.

[131] Dafna Sheinwald. On the Ziv-Lempel proof and related topics. *Proceedings of the IEEE*, 82(6):866–871, 1994. doi:10.1109/5.286190.

[132] Dafna Sheinwald, Abraham Lempel, and Jacob Ziv. On encoding and decoding with two-way head machines. *Inf. Comput.*, 116(1):128–133, 1995. doi:10.1006/inco.1995.1009.

[133] John C. Shepherdson. The reduction of two-way automata to one-way automata. *IBM J. Res. Dev.*, 3(2):198–200, 1959. doi:10.1147/rd.32.0198.

[134] Michael Sipser. A complexity theoretic approach to randomness. In *Proceedings of the 15th Annual ACM Symposium on Theory of Computing, 1983, Boston, Massachusetts, USA*, pages 330–335. ACM, 1983. doi:10.1145/800061.808762.

[135] Robert I. Soare. Computability and recursion. *Bull. Symb. Log.*, 2(3):284–321, 1996. doi:10.2307/420992.

[136] Fernando Soler-Toscano, Hector Zenil, Jean-Paul Delahaye, and Nicolas Gauvrit. Calculating Kolmogorov complexity from the output frequency distributions of small turing machines. *PLoS ONE*, 9(5):e96223, May 2014. doi:10.1371/journal.pone.0096223.

[137] R.J. Solomonoff. A preliminary report on a general theory of inductive inference. Technical Report ZTB-138, Zator Company, Cambridge, Mass., November 1960. Accessed: June 14, 2021. URL: http://raysolomonoff.com/publications/z138.pdf.

[138] R.J. Solomonoff. A formal theory of inductive inference. part i. *Inform. Control.*, 7(1):1–22, 1964. doi:https://doi.org/10.1016/S0019-9958(64)90223-2.

[139] R.J. Solomonoff. A formal theory of inductive inference. part ii. *Inform. Control*, 7(2):224–254, 1964. doi:https://doi.org/10.1016/S0019-9958(64)90131-7.

[140] Christian Steinruecken. *Lossless Data Compression*. PhD thesis, University of Cambridge, 2014. (Accessed: May 11, 2021). URL: https://q4.github.io/thesis.pdf.

[141] Jason Teutsch. Short lists for shortest descriptions in short time. *Comput. Complex.*, 23(4):565–583, September 2014. doi:10.1007/s00037-014-0090-3.

[142] Alan M. Turing. On computable numbers, with an application to the Entscheidungsproblem. *P. Lond. Math. Soc.*, s2-42(1):230–265, 01 1937. doi:10.1112/plms/s2-42.1.230.

[143] Tatyana van Aardenne-Ehrenfest and Nicolaas Govert de Bruijn. Circuits and trees in oriented linear graphs. *Simon Stevin : Wis- en Natuurkundig Tijdschrift*, 28:203–217, 1951. URL: https://research.tue.nl/en/publications/circuits-and-trees-in-oriented-linear-graphs-3.

[144] Paul M. B. Vitányi. Corrigendum to [6]. *Theor. Comput. Sci.*, 770:101, 2019. doi:10.1016/j.tcs.2018.07.009.

[145] Paul M. B. Vitányi. Logical depth for reversible turing machines with an application to the rate of decrease in logical depth for general turing machines. *Theor. Comput. Sci.*, 778:78–80, 2019. doi:10.1016/j.tcs.2019.01.031.

[146] Donal Dines Wall. *Normal Numbers*. PhD thesis, University of California, Berkeley, 1948.

[147] Terry A. Welch. A technique for high-performance data compression. *Computer*, 17(6):8–19, 1984. doi:10.1109/MC.1984.1659158.

[148] Ian H. Witten and Timothy C. Bell. The zero-frequency problem: Estimating the probabilities of novel events in adaptive text compression. *IEEE Trans. Inf. Theory*, 37(4):1085–1094, 1991. doi:10.1109/18.87000.

[149] Ian H. Witten, Radford M. Neal, and John G. Cleary. Arithmetic coding for data compression. *Commun. ACM*, 30(6):520–540, June 1987. doi:10.1145/214762.214771.

[150] Aaron D. Wyner and Abraham J. Wyner. Improved redundancy of a version of the Lempel-Ziv algorithm. *IEEE Trans. Inf. Theory*, 41(3):723–731, 1995. doi:10.1109/18.382018.

[151] Aaron D. Wyner and Jacob Ziv. Fixed data base version of the Lempel-Ziv data compression algorithm. *IEEE Trans. Inf. Theory*, 37(3):878–880, 1991. doi:10.1109/18.79955.

[152] Aaron D. Wyner and Jacob Ziv. The sliding-window Lempel-Ziv algorithm is asymptotically optimal. *Proceedings of the IEEE*, 82(6):872–877, 1994. doi:10.1109/5.286191.

[153] Hector Zenil, Jean-Paul Delahaye, and Cédric Gaucherel. Image characterization and classification by physical complexity. *Complex.*, 17(3):26–42, 2012. doi:10.1002/cplx.20388.

[154] Hector Zenil, Santiago Hernández-Orozco, Narsis A. Kiani, Fernando Soler-Toscano, Antonio Rueda-Toicen, and Jesper Tegnér. A decomposition method for global evaluation of shannon entropy and local estimations of algorithmic complexity. *Entropy*, 20(8):605, 2018. doi:10.3390/e20080605.

[155] Marius Zimand. Short lists with short programs in short time - A short proof. In Arnold Beckmann, Erzsébet Csuhaj-Varjú, and Klaus Meer, editors, *Language, Life, Limits - 10th Conference on Computability in Europe, CiE 2014, Budapest, Hungary, June 23-27, 2014. Proceedings*, volume 8493 of *Lecture Notes in Computer Science*, pages 403–408. Springer, 2014. doi:10.1007/978-3-319-08019-2_42.

[156] Jacob Ziv and Abraham Lempel. A universal algorithm for sequential data compression. *IEEE Trans. Inf. Theory*, 23(3):337–343, 1977. doi:10.1109/TIT.1977.1055714.

[157] Jacob Ziv and Abraham Lempel. Compression of individual sequences via variable-rate coding. *IEEE Trans. Inf. Theory*, 24(5):530–536, 1978. doi:10.1109/TIT.1978.1055934.

[158] A.K. Zvonkin and Leonid A. Levin. The complexity of finite objects and the development of the concepts of information and randomness by means of the theory of algorithms. *Russ. Math. Survey*, 25(6):83–124, December 1970. doi:10.1070/rm1970v025n06abeh001269.