Czech Technical University in Prague
Faculty of Nuclear Sciences and Physical Engineering

Department of mathematics
Study major: BMATIMIN

String attractors

BACHELOR THESIS

Author:	Veronika Hendrychová
Supervisor:	doc. Ing. Ľubomíra Dvořáková, Ph.D.
Consultant:	Ing. Karel Břinda, Ph.D.
Year:	2023

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Hendrychová** Jméno: **Veronika** Osobní číslo: **502475**

Fakulta/ústav: **Fakulta jaderná a fyzikálně inženýrská**

Zadávající katedra/ústav: **Katedra matematiky**

Studijní program: **Matematické inženýrství**

Specializace: **Matematická informatika**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Atraktory řetězců

Název bakalářské práce anglicky:

String attractors

Pokyny pro vypracování:

1. Seznamte se se základními pojmy kombinatoriky na slovech, speciálně s pojmem atraktor řetězce a motivací pro jeho zavedení [1], [2].
2. Nastudujte definici Burrows-Wheelerovy transformace [3] a známé výsledky o ní ve vztahu k atraktorům řetězců (viz např. review [4]).
3. Seznamte se s dosavadními výsledky pro atraktory faktorů nekonečných slov, zejména pak slov sturmovských [5]–[6].
4. Zkoumejte atraktory prefixů nekonečných slov získaných palindromickými uzávěry a posléze antipalindromickými uzávěry. Navrhněte algoritmus, který pro nekonečné slovo získané pomocí uzávěrů vygeneruje prefix zadané délky a nalezne jeho minimální atraktor. Implementujte tento algoritmus v některém z běžných programovacích jazyků a určete jeho složitost.
5. S pomocí této implementace zkuste na další vhodně zvolené třídě nekonečných slov odvodit velikost minimálního atraktoru a exaktně navržené vztahy dokázat.

Seznam doporučené literatury:

[1] N. Prezza, String Attractors. arXiv [cs.DS], Sep. 15, 2017. [Online]. Available: http://arxiv.org/abs/1709.05314
[2] D. Kempa, N. Prezza, At the roots of dictionary compression: string attractors. In 'Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing', Los Angeles, CA, USA, Jun. 2018, 827–840.
[3] M. Burrows, D. J. Wheeler, A Block-sorting Lossless Data Compression Algorithm. SRC Research Report, Digital Equipment Corporation, California, 1994.
[4] G. Navarro, Indexing Highly Repetitive String Collections, Part I: Repetitiveness Measures. ACM Comput. Surv. 54(2), 2021, 1–31.
[5] S. Mantaci, A. Restivo, G. Romana, G. Rosone, M. Sciortino, A combinatorial view on string attractors. Theor. Comput. Sci. 850, 2021, 236–248.
[6] L. Schaeffer, J. Shallit, String Attractors for Automatic Sequences. arXiv [cs.FL], Dec. 12, 2020. [Online]. Available: http://arxiv.org/abs/2012.06840

Jméno a pracoviště vedoucí(ho) bakalářské práce:

doc. Ing. Lubomíra Dvořáková, Ph.D. katedra matematiky FJFI

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Ing. Karel Břinda, Ph.D. INRIA, the French National Research Institute for Computer Science and Automation

Datum zadání bakalářské práce: **31.10.2022** Termín odevzdání bakalářské práce: **02.08.2023**

Platnost zadání bakalářské práce: 31.10.2024

[illegible] [illegible] [illegible]

doc. Ing. Lubomíra Dvořáková, Ph.D.	prof. Ing. Zuzana Masáková, Ph.D.	doc. Ing. Václav Čuba, Ph.D.
podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Studentka bere na vědomí, že je povinna vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

2. 11. 2022	Hendrychová
Datum převzetí zadání	Podpis studentky

Declaration

I declare that I have independently prepared the bachelor thesis on the topic of String attractors and have used only the literature and sources stated in bibliography.

Prague, 2. 8. 2023 Veronika Hendrychová

Acknowledgement

I would like to thank my supervisor, Ľubomíra Dvořáková, for her professional guidance in research and writing of my bachelor thesis. I would also like to thank my consultant, Karel Břinda, for his valuable advice and assistance.

Veronika Hendrychová

Title:
String attractors

Author: Veronika Hendrychová

Study major: BMATIMIN
Type: Bachelor thesis

Supervisor: doc. Ing. Ľubomíra Dvořáková, Ph.D.
FNSPE, CTU
Consultant: Ing. Karel Břinda, Ph.D.
INRIA, the French National Research Institute for Computer Science and Automation

Abstract: String attractor, a relatively new combinatorial notion, is closely related to measuring the complexity of words and offers a unified approach to the repetitiveness measures induced by dictionary compressors. However, attractors have been only little studied in the context of combinatorics on words, particularly for classes of low complexity, including pseudostandard, and complementary-symmetric Rote sequences. In this work, we determine the form of attractors of pseudopalindromic prefixes of these sequences and prove their minimality. We design, implement and experimentally evaluate Python programs to generate these words and their attractors, validate a presumed attractor of a given word, and identify a minimal attractor of a given word using a brute-force approach.
Keywords: string attractors, Sturmian sequences, episturmian sequences, pseudostandard sequences, dictionary compression

Contents

Introduction

Efficient handling of vast and repetitive datasets holds significant importance in numerous research fields, including biology, physics and computer science. Examples of large repetitive datasets include software repositories such as GitHub, versioned documents such as Wikipedia, and genomic databases such as the datasets of the 100,000 Genomes Project [1]. Especially in the field of bioinformatics, it is crucial not only to compress data but also to perform complex operations on them, such as to search them or provide random access.

To efficiently store repetitive data, a wide class of methods called dictionary compressors has been developed. To achieve more efficient compression, these methods employ the repeated sections in the input. They enable not only to reduce storage requirements, but also provide theoretical and practical tools to quantify the underlying complexity of the data. However, these measures tend to be specific to the compression process and do not provide a universal view of the input complexity. Additionally, the effectiveness of the methods, which often involve multiple algorithms, is hard to compare.

A recently introduced notion of string attractors represents a new approach to this problematics. The concept was initially presented by Nicola Prezza [2] and further studied by Prezza and Dominik Kempa [3]. They demonstrated that string attractors, despite being relatively simple combinatorial objects, are naturally induced by dictionary compressors. They represent a way to unify the compressors' measures, thus offering a new approach to compare compression methods and providing a more comprehensive understanding of data complexity. The compressors can be viewed as approximation algorithms to identify an attractor of minimum size. Kempa and Prezza also showed that the problem of finding the minimum size attractor is NP-complete, which means that the computational effort to obtain it grows exponentially with the input size, although verifying the correct solution can be done efficiently.

Subsequently, properties of string attractors and optimization of obtaining them have been studied. These include for instance the relation to the sensitivity of a string compression algorithm [4], and obtaining a minimal attractor by reduction to MAX-SAT formulations [5]. A modification to a k-attractor has been introduced and studied [6, 7], as well as a circular variant of attractors [8].

In the field of combinatorics on words, string attractors can also be examined in the context of infinite words. This essentially means study of finite factors or prefixes of any length, which was formalized in terms of string attractor profile function [9]. Considerable attention has been dedicated to studying attractors of specific classes of infinite words that follow certain structural regulations, which makes the problem more tractable. The attractors of minimum size for factors and prefixes of various sequences have been determined so far. These include standard Sturmian sequences [8, 10], the Tribonacci sequence [11], episturmian sequences [10], the Thue-Morse sequence [12, 11, 13], the period-doubling sequence [11], and the powers of two sequence [11, 14]. More recent works include string attractors of fixed points of k-bonacci-like morphisms [15].

In this work, we study string attractors in the context of previously unstudied sequences obtained by pseudopalindromic closures. First, we focus on pseudostandard sequences, which are obtained using antipalindromic closures. Second, we study the complementary-symmetric Rote sequences generated by a special combination of pseudopalindromic closures. They form a subclass in the family of generalized pseudostandard sequences, which also includes for example the well-known Thue-Morse sequence. Building upon the known form of attractors of standard Sturmian and episturmian sequences, we determine the form of the attractors of pseudopalindromic prefixes of these infinite words and formally prove their minimality. To reach our results, we implement several algorithms, which are included.

1 String attractors and compression methods

1.1 Preliminaries

An *alphabet* $\mathcal{A}$ is a finite non-empty set of symbols called *letters*. A *word* over $\mathcal{A}$ of *length* n is a string $u = u_0u_1 \cdots u_{n-1}$, where $u_i \in \mathcal{A}$ for all $i \in \{0, 1, \ldots, n-1\}$. We let $|u|$ denote the length of u. The concatenation of two words $u = u_0u_1 \cdots u_{n-1}$ and $v = v_1v_2 \cdots v_{m-1}$ is the word $uv = u_0 \cdots u_{n-1}v_0 \cdots v_{m-1}$. The neutral element for concatenation of words is the *empty word* ε and its length is set to $|\varepsilon| := 0$. The set of all finite words over $\mathcal{A}$ together with the operation of concatenation forms a monoid, denoted $\mathcal{A}^*$, and we write $\mathcal{A}^+ = \mathcal{A}^* \setminus \{\varepsilon\}$. If $u = xyz$ for $x, y, z \in \mathcal{A}^*$, then x is a *prefix* of u, z is a *suffix* of u and y is a *factor* of u.

A *sequence*, also called an *infinite word*, over $\mathcal{A}$ is an infinite string $\mathbf{u} = u_0u_1u_2 \cdots$, where $u_i \in \mathcal{A}$ for all $i \in \mathbb{N}$. We always denote sequences by bold letters. A sequence $\mathbf{u}$ is *eventually periodic* if $\mathbf{u} = vwww \cdots := v(w)^\omega$ for some $v \in \mathcal{A}^*$ and $w \in \mathcal{A}^+$. If $\mathbf{u}$ is not eventually periodic, then it is *aperiodic*. A *factor* of $\mathbf{u} = u_0u_1u_2 \cdots$ is a word y such that $y = u_iu_{i+1}u_{i+2} \cdots u_{j-1}$ for some $i, j \in \mathbb{N}$, $i \leq j$. If $i = j$, then $y = \varepsilon$. In the context of string attractors, the set $\{i, i+1, \ldots, j-1\}$ is called an *occurrence* of the factor y in $\mathbf{u}$. (Usually, only the number i is called an occurrence of y in $\mathbf{u}$.) If $i = 0$, the factor y is a *prefix* of $\mathbf{u}$.

1.2 String attractors and their basic properties

Definition 1.1. A *string attractor* (or *attractor* for short) of a word $w = w_0w_1 \cdots w_{n-1}$, where $w_i \in \mathcal{A}$ for all $i \in \{0, 1, \ldots, n-1\}$, is a set $\Gamma \subset \{0, 1, \ldots, n-1\}$ such that every factor of w has an occurrence containing at least one element of Γ.

Example 1.2. Consider the word $w = \texttt{1324321324}$. One example of its attractor is the following set Γ (see the corresponding positions in the word underlined):

$$\Gamma = \{2, 3, 4, 6, 9\} \leftrightarrow w = \texttt{13}\underline{\texttt{243}}\texttt{2}\underline{\texttt{1}}\texttt{32}\underline{\texttt{4}}\,.$$

Let us verify that Γ is a valid attractor of w. From Definition 1.1, all substrings of w must have an occurrence crossing some position from the attractor. Verifying the substrings that contain any of the underlined positions is trivial, as they already cross some position from Γ. The remaining substrings in between the underlined positions (the substrings are `1`, `3`, `2`, `13`, and `32`) must have another occurrence in the word that crosses some of the positions from Γ. We can observe that this is satisfied.
Another attractor of w is Γ^*:

$$\Gamma^* = \{2, 3, 4, 6\} \leftrightarrow w = \texttt{13}\underline{\texttt{243}}\texttt{2}\underline{\texttt{1}}\texttt{324}\,.$$

The * symbol denotes an attractor of the smallest possible size. We will denote this minimum size as $\gamma^* := |\Gamma^*|$. The size of the above attractor is apparently minimal since it is equal to the number of distinct letters in the word.

Minimal attractors are not uniquely determined; a single string may have multiple as demonstrated by the following example of another minimal attractor

$$\Gamma^* = \{0, 4, 5, 9\} \leftrightarrow w = \underline{\texttt{1}}\texttt{324}\underline{\texttt{32}}\texttt{132}\underline{\texttt{4}}\,.$$

From Definition 1.1 and Example 1.2, we can observe several fundamental properties of attractors. The set $\{0, 1, \ldots, n-1\}$ always forms an attractor. All attractors must satisfy $|\Gamma| \geq |\mathcal{A}|$ (assuming all letters appear in the word). Any superset of an attractor is an attractor, too.

1.3 Dictionary compression

Dictionary compressors are algorithms used to losslessly compress highly repetitive data. These methods exploit the string repetitiveness and are able to achieve better results than other methods, e.g. the widely used entropy compression. There are several groups of these algorithms, connected by the underlying idea of referencing the repeated substrings to a dictionary of strings. Most of these methods induce a measure of complexity of the compressed string, which is determined by the individual method's effect on the input. Below we will describe the underlying algorithms of two widely used compressors – gzip and bzip2.

Gzip is based on the LZ77 algorithm, which we will describe in detail below. This algorithm belongs to a broader family of Lempel-Ziv (LZ) compressors [16], which employ a parsing technique to divide the input into phrases. These phrases are then encoded using pointers to previously encountered occurrences of the same phrases. The measure of the input repetitiveness is represented by the number of phrases into which it is parsed.

Bzip2 uses the Burrows-Wheeler transform [17], which is a reversible textual transform. It is based on sorting the rows of a matrix of the string's circular permutations, and compressing the last column of this matrix using some other method, e.g. run-length encoding. Its measure is the number of equal-letter runs in the last column of the sorted matrix.

There are many other groups of dictionary compressors. One family based on context-free grammar includes *straight-line programs* (SLP), *run-length encoded SLP* (RLSLP), and *collage systems* [18], the latter being the richest generalization of the previous. The measure here is denoted by the size of a generic collage system. Among other known compressors we can include the *compact directed acyclic word graph* (CDAWG) [19]. It is based on building a compact automaton for recognizing the string's suffixes, and its measure is the number of its edges.

All of these methods can be viewed as an approximation algorithms of a minimal attractor of the input. Attractors have a close relation to their measures of complexity, namely using the minimum size γ^*, and Kempa and Prezza proved exact relations between them [3]. On top of that, attractors and their way to represent a complexity measure are not dependent on any particular compression method and are therefore universal. Below we will describe in greater detail the connection between string attractors and LZ and BWT methods.

Attractors and the methods based on Lempel-Ziv compression

Lempel-Ziv methods represent one of the largest groups of dictionary compressors. Their basic principle is to replace repeated factors with pointers to previously seen instances of those factors. There are many variants of the algorithm that were developed throughout the years – the most used ones include LZ77, LZ78, and LZW [20]. In the example below, we will work with the LZ77 variant, however, in the context of string attractors, the differences between the variants are not of much relevance.

The algorithm implements greedy, left-to-right parsing of the given word into the longest previous factors. After this partition, the word is encoded by linking these phrases to their previous occurrences. The estimate of the word's repetitiveness is expressed by the measure z, which denotes the number of phrases into which the word is parsed.

Example 1.3. Let us have a finite word $w = \texttt{0101101101}$ to be compressed by the LZ77 algorithm. We will use the LZ77 variant where the overlaps between phrases are allowed. The word will be parsed into phrases as

$$\begin{array}{r|c|c|c|c}
 & 0 & 1 & 2\;\;3 & 4\;\;5\;\;6\;\;7\;\;8\;\;9 \\
\hline
\mathrm{LZ77}(w): & \texttt{0} & \texttt{1} & \texttt{0}\;\;\texttt{1} & \texttt{1}\;\;\texttt{0}\;\;\texttt{1}\;\;\texttt{1}\;\;\texttt{0}\;\;\texttt{1}
\end{array}.$$

The two-letter factor `01` has an occurrence in the word's prefix, and the factor `101101` has an occurrence starting at the position 1. After this parsing, we can encode the word as a set of pairs (length, letter/position):

$$\mathrm{LZ77}(w) = (1, '\texttt{0}'), (1, '\texttt{1}'), (2, 0), (6, 1)\,.$$

In this case, the size of the Lempel-Ziv parse (i.e. the number of factors) is 4, therefore the measure $z = 4$.

String attractors are closely connected to the final parsing. Firstly, we can observe that the first positions in the individual phrases always form an attractor of the given word (in Example 1.3 this would mean $\underline{\texttt{0}}\underline{\texttt{1}}\underline{\texttt{0}}\texttt{1}\underline{\texttt{1}}\texttt{01101}$). This is due to the fact that for any factor there either must exist its previous occurrence of this phrase, or it is the first occurrence and then its position is in the attractor. However, it is clearly not minimal – consider $\texttt{01}\underline{\texttt{0}}\texttt{1}\underline{\texttt{1}}\texttt{01101}$ in Example 1.3. Overall, the size γ^* of the smallest string attractor gives us both lower and upper bounds for the measure z as summarized in the following theorems. Here the number n denotes the length of the string.

Theorem 1.4 (Lemma 3.7 in [3]). *Let z be the number of factors of the Lempel-Ziv factorization of a string w. Then, w has an attractor of size z.*

Theorem 1.5 (Corollary 3.15 in [3]). *The following bound holds between the size z of the Lempel-Ziv parse and the size γ^* of the minimal string attractor: $z \in \mathcal{O}(\gamma^* \log^2(n/\gamma^*))$.*

Attractors and the methods based on the Burrows-Wheeler transform

The Burrows-Wheeler transform is an invertible string transform that rearranges text into runs of identical letters. The transformation itself is not a compression method, but used on highly repetitive data, it creates much more compressible content for e.g. run-length encoding (RLBWT). Importantly, in the context of data compression and bioinformatics, BWT can be further equipped with additional data tables to form a highly efficient fulltext index called the FM-index [21].

The computation of the BWT of a given string starts by appending the \$ character to the string to ensure reversibility, and creating a matrix containing all the string's circular rotations as the rows. The rows are then lexicographically sorted, after which the last column of the matrix contains the resulting BWT-transformed string. Note that in practice more memory- and time-efficient algorithms are used.

The obtained transformed string can be then compressed using relatively simple methods, such as the run-length encoding (RLE), as in the example below. The measure r of the words complexity is the number of runs of equal letters in the transformed string. We note that for simplicity, many theoretical papers about string attractors that focus primarily on their asymptotic properties use a slightly different, non-reversible, variant of the BWT, which results also in slightly different values of r.

Example 1.6. Let us again take the word $w = \mathtt{0101101101}$, this time to be compressed by RLBWT. Firstly, we append the sign \$, then we create the word's rotation matrix and sort it.

$$\begin{array}{ccccccccccc}
0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & \$ \\
1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & \$ & 0 \\
0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & \$ & 0 & 1 \\
1 & 1 & 0 & 1 & 1 & 0 & 1 & \$ & 0 & 1 & 0 \\
1 & 0 & 1 & 1 & 0 & 1 & \$ & 0 & 1 & 0 & 1 \\
0 & 1 & 1 & 0 & 1 & \$ & 0 & 1 & 0 & 1 & 1 \\
1 & 1 & 0 & 1 & \$ & 0 & 1 & 0 & 1 & 1 & 0 \\
1 & 0 & 1 & \$ & 0 & 1 & 0 & 1 & 1 & 0 & 1 \\
0 & 1 & \$ & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 \\
1 & \$ & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 \\
\$ & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1
\end{array}
\qquad
\begin{array}{cccccccccc|c|}
\$ & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 \\
0 & 1 & \$ & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 \\
0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & \$ \\
0 & 1 & 1 & 0 & 1 & \$ & 0 & 1 & 0 & 1 & 1 \\
0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & \$ & 0 & 1 \\
1 & \$ & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 \\
1 & 0 & 1 & \$ & 0 & 1 & 0 & 1 & 1 & 0 & 1 \\
1 & 0 & 1 & 1 & 0 & 1 & \$ & 0 & 1 & 0 & 1 \\
1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & \$ & 0 \\
1 & 1 & 0 & 1 & \$ & 0 & 1 & 0 & 1 & 1 & 0 \\
1 & 1 & 0 & 1 & 1 & 0 & 1 & \$ & 0 & 1 & 0
\end{array}$$

Matrix with word's rotations as rows   Lexicographically sorted rows

The last column of the sorted matrix, $\mathrm{BWT}(w) = \mathtt{11\$11011000}$ is then encoded by the run-length algorithm and we obtain $\mathrm{RLBWT}(w) = (2, \mathtt{1}), (1, \$), (2, \mathtt{1}), (1, \mathtt{0}), (2, \mathtt{1}), (3, \mathtt{0})$. In this case, the number of equal-letter runs is 6, therefore $r = 6$.

Similarly to the LZ example, we can again observe the attractors' trail in the encoded word – the first positions in the runs always form an attractor. Thanks to the fact that the BWT preserves the relative order of equal letters, we are able to trace the letters in the first positions in $\mathrm{BWT}(w)$ back in the original word. In this example, we underlined them in $\mathrm{BWT}(w) = \mathtt{\underline{1}1\$\underline{1}1\underline{01}1\underline{0}00}$. Indexing from 0 in the original word $w = \mathtt{0101101101}$, we can determine that the underlined positions in $\mathrm{BWT}(w)$ are reflected to positions 1, 4, 0, 7 and 2, respectively, i.e., $w = \mathtt{\underline{010}1\underline{1}01\underline{1}01}$. It can be easily recognized that the underlined positions form an attractor of w. However, we can observe that it is not minimal.

In general, the size of a minimal attractor represents the lower bound for the measure r.

Theorem 1.7 (Theorem 3.9 in [3]). *Let r be the number of equal-letter runs in the Burrows-Wheeler transform of w. Then, w has an attractor of size r.*

Attractors and comparison of the methods

Besides providing upper and lower bounds, string attractors are also useful for comparing individual dictionary compressors to each other. For illustration, we give an example of the relationship between LZ measure z and BWT measure r.

Theorem 1.8 (Corollary 3.16 in [3])**.** *The following bound holds between the number r of equal-letter runs in the BWT, the size z of the Lempel-Ziv parse, and the length n of the input:* $z \in \mathcal{O}(r \log^2 \frac{n}{r})$.

In the next chapter, we will study string attractors in the context of several classes of sequences studied in combinatorics on words.

2 Episturmian, pseudostandard, and Rote sequences

In this section, we will study three families of sequences and methods to generate them. We will describe attractors of their prefixes and formally prove their minimality.

2.1 Palindromic closures and episturmian sequences

The *language* $\mathcal{L}(\mathbf{u})$ of a sequence $\mathbf{u}$ is the set of all its factors. A factor w of $\mathbf{u}$ is left special if aw, bw are in $\mathcal{L}(\mathbf{u})$ for at least two distinct letters $a, b \in \mathcal{A}$. The *factor complexity* of a sequence $\mathbf{u}$ is the mapping $\mathcal{C}_{\mathbf{u}} : \mathbb{N} \to \mathbb{N}$ defined by $\mathcal{C}_{\mathbf{u}}(n) = \#\{u \in \mathcal{L}(\mathbf{u}) : |u| = n\}$. The factor complexity of an aperiodic sequence $\mathbf{u}$ satisfies $\mathcal{C}_{\mathbf{u}}(n) \geq n + 1$ for all $n \in \mathbb{N}$ [22].

The aperiodic sequences with the lowest possible factor complexity $\mathcal{C}_{\mathbf{u}}(n) = n + 1$ are called *Sturmian sequences*. Sturmian sequence is *standard* if all left special factors are prefixes.

Clearly, all Sturmian sequences are defined over a binary alphabet, e.g., $\{\mathtt{0}, \mathtt{1}\}$. It is a well-known fact that for each Sturmian sequence there exists a unique standard Sturmian sequence with the same language [23]. Since we are interested in the language of Sturmian sequences, it suffices to consider only standard Sturmian sequences. They can be generated using a method called *palindromic closures*.

Definition 2.1. The map $R : \mathcal{A}^* \to \mathcal{A}^*$, called *reversal*, associates with each word its mirror image, i.e., $R(u_0 u_1 \cdots u_{n-1}) = u_{n-1} \cdots u_1 u_0$, where $u_i \in \mathcal{A}$ for each $i \in \{0, 1, \ldots, n-1\}$. A word u is a *palindrome* if $u = R(u)$.

Definition 2.2. Let u be a word, then by *palindromic closure* $(u)^R$ we denote the shortest palindrome having u as prefix.

Example 2.3. Let $u = \mathtt{00}$, then $(u)^R = u = \mathtt{00}$ and $(u\mathtt{1})^R = (\mathtt{001})^R = \mathtt{00100}$. For $v = \mathtt{110110}$, we have $(v\mathtt{0})^R = \mathtt{110110011011}$ and $(v\mathtt{1})^R = \mathtt{11011011}$.

Definition 2.4. Let $\Delta = \delta_1 \delta_2 \delta_3 \cdots$ with $\delta_i \in \mathcal{A}$ and define $u_0 = \varepsilon$ and $u_{n+1} = (u_n \delta_{n+1})^R$ for all $n \in \mathbb{N}$. Then we denote $\mathbf{u}(\Delta) = \lim_{n\to\infty} u_n$, i.e., $\mathbf{u}(\Delta)$ is a unique sequence having u_n as prefix for each $n \in \mathbb{N}$, and we call Δ the *directive sequence* of $\mathbf{u}(\Delta)$.

Theorem 2.5 (Theorem 5 in [10] restricted to Sturmian sequences)**.** *Let* $\mathbf{u}$ *be a standard Sturmian sequence over* $\mathcal{A} = \{\mathtt{0}, \mathtt{1}\}$*. Then* $\mathbf{u} = \mathbf{u}(\Delta)$ *for a unique sequence* $\Delta = \delta_1 \delta_2 \cdots$ *with* $\delta_i \in \mathcal{A}$*.*

Generation of Sturmian sequences by palindromic closures gives a direct view on minimal attractors of their prefixes. The proof from [10] restricted to Sturmian sequences is recalled here as a similar technique will be applied later for describing the attractors of pseudostandard sequences.

Theorem 2.6 (Theorem 7 in [10] restricted to Sturmian sequences)**.** *Let* $\mathbf{u}$ *be a standard Sturmian sequence* $\mathbf{u} = \mathbf{u}(\Delta)$*. Then each* u_n *containing two letters (both* $\mathtt{0}$ *and* $\mathtt{1}$*) has an attractor of size 2 in the form* $\Gamma = \{m_0, m_1\}$*, where* $m_a = \max\{|p| : p \textit{ is a palindrome and } pa \textit{ is a prefix of } \mathbf{u}\}$*.*

Example 2.7. Let us have a directive sequence $\Delta = \mathtt{01001}\cdots$. Then the generation of prefixes unfolds as follows (attractors are underlined in accordance with Theorem 2.6):

$$\begin{aligned}
u_1 &= \mathtt{0} \\
u_2 &= \underline{\mathtt{0}}\underline{\mathtt{1}}\mathtt{0} \\
u_3 &= \mathtt{0}\underline{\mathtt{1}}\mathtt{0}\underline{\mathtt{0}}\mathtt{10} \\
u_4 &= \mathtt{0}\underline{\mathtt{1}}\mathtt{00100}\underline{\mathtt{0}}\mathtt{10} \\
u_5 &= \mathtt{0100100}\underline{\mathtt{0}}\mathtt{10}\underline{\mathtt{1}}\mathtt{0010010}
\end{aligned}$$

Proof. Let us recall that we index the positions in words from 0, i.e., $u = u_0 u_1 \cdots u_{|u|-1}$. We will make the proof by mathematical induction on n. Let u_k be the first word to contain both letters and without loss of generality, let the directive sequence be in the form $\Delta = \mathtt{0}^{k-1}\mathtt{1}\cdots$, where $k > 1$.

- For $n = k$, the word is in the form $u_k = \mathtt{0}^{k-1}\mathtt{1}\mathtt{0}^{k-1}$, the longest palindromic prefix followed by 0 is $m_0 = \mathtt{0}^{k-2}$ and the longest palindromic prefix followed by 1 is $m_1 = \mathtt{0}^{k-1}$. Indeed, the set $\{k-2, k-3\}$ forms an attractor in $u_k = \mathtt{0}^{k-2}\underline{\mathtt{0}}\underline{\mathtt{1}}\mathtt{0}^{k-1}$.

- For $n > k$, let us assume that the next letter in the directive sequence is 0 (if we chose 1, the proof would be analogous). The word is then $u_n = (u_{n-1}\mathtt{0})^R = w\mathtt{0}m_0\mathtt{0}R(w)$ where $u_{n-1} = w\mathtt{0}m_0 = m_0\mathtt{0}R(w)$ and m_0 is the longest palindromic prefix followed by 0 in u_{n-1}. From this we can observe that the two occurrences of u_{n-1} overlap in the word u_n. The case $u_n = u_{n-1}\mathtt{0}u_{n-1}$ cannot occur as $u_{n-1} \neq \mathtt{111}\cdots\mathtt{1}$.

  We want to show that the underlined positions in $u_n = w\mathtt{0}m_0\underline{\mathtt{0}}R(w) = m_1\underline{\mathtt{1}}\cdots$ is an attractor. Note that m_1 remains the same for both u_n and u_{n-1} because in this step we added $\mathtt{0}$.

  Let us take any factor in $u_n = w\mathtt{0}m_0\underline{\mathtt{0}}R(w) = m_1\underline{\mathtt{1}}\cdots$. If it crosses the position of the underlined zero, the proof is finished. If it does not, then it must be wholly contained in u_{n-1} and cross the attractor position of $\mathtt{1}$. This is due to the fact that u_{n-1} forms both the prefix and suffix of u_n. Using the induction assumption, the positions of the $\mathtt{0}$ that is now in the suffix form of u_{n-1} and the $\mathtt{1}$ that is found in the prefix form of u_{n-1} formed the attractor for u_{n-1}, therefore any attractor contained in u_{n-1} crosses at least one of them.

□

A similar statement holds not only for palindromic prefixes of standard Sturmian sequences, but also for any factors.

Theorem 2.8 (Theorem 10 in [10] restricted to Sturmian sequences)**.** *Each factor of a Sturmian sequence has an attractor of size 2.*

Sturmian sequences form a subclass of another family of strings with low complexity, called episturmian sequences. In fact, Sturmian sequences correspond to aperiodic binary episturmian sequences. Again, we can generate these infinite words using palindromic closures and obtain a minimal attractor of their prefixes, this time using any finite alphabet.

Definition 2.9 (Definition 4 in [10])**.** Let $\mathbf{u}$ be a sequence whose language is closed under reversal and such that for each length n it contains at most one left special factor. Then $\mathbf{u}$ is called an *episturmian sequence*. An episturmian sequence is *standard* if all left special factors are prefixes.

In the study of attractors, we are interested in the language of the infinite words. Similarly to Sturmian sequences, for each episturmian sequence there exists a standard episturmian sequence with the same language. Therefore, we will consider only standard episturmian sequences.

Theorem 2.10 (Theorem 5 in [10])**.** *Let* $\mathbf{u}$ *be a standard episturmian sequence over* $\mathcal{A}$*. Then* $\mathbf{u} = \mathbf{u}(\Delta)$ *for a unique sequence* $\Delta = \delta_1\delta_2\cdots$ *with* $\delta_i \in \mathcal{A}$*.*

Theorem 2.11 (Theorem 7 in [10])**.** *Let u be a non-empty palindromic prefix of a standard episturmian sequence. For every letter a occurring in u, denote*

$$m_a = \max\{|p| \,:\, p \text{ is a palindrome and } pa \text{ is a prefix of } u\}.$$

Then $\Gamma = \{m_a : a \text{ occurs in } u\}$ *is an attractor of u and its size is minimal.*

Theorem 2.12 (Theorem 10 in [10])**.** *Let* $\mathbf{u}$ *be an episturmian sequence. Each factor of* $\mathbf{u}$ *containing d distinct letters has an attractor of size d.*

The algorithm to generate episturmian words and their attractors in accordance with Theorem 2.11 is described as Algorithm 1.

2.2 Antipalindromic closures and pseudostandard sequences

Another studied group of infinite words are pseudostandard sequences, which are generated using antipalindromic closures. Due to the definition of antipalindromes, we are now restricted to binary alphabet.

Definition 2.13. The map $E : \{\mathtt{0},\mathtt{1}\}^* \to \{\mathtt{0},\mathtt{1}\}^*$, called *exchange antimorphism*, is a composition of reversal and letter exchange, i.e., $E(w_0w_1\cdots w_{n-1}) = \overline{w_{n-1}}\cdots\overline{w_1}\ \overline{w_0}$, where $w_i \in \{\mathtt{0},\mathtt{1}\}$ and $\overline{w_i} = 1 - w_i$ for each $i \in \{0,1,\ldots,n-1\}$. A word w is an *antipalindrome* if $w = E(w)$.

Definition 2.14. Let w be a word, then by *antipalindromic closure* $(w)^E$ we denote the shortest antipalindrome having w as prefix.

Example 2.15. Let $u = \mathtt{01}$, then $(u)^E = u = \mathtt{01}$ and $(u\mathtt{1})^E = (\mathtt{011})^E = \mathtt{011001}$. For $v = \mathtt{1101}$, we have $(v\mathtt{0})^E = \mathtt{110100}$ and $(v\mathtt{1})^E = \mathtt{1101100100}$.

Definition 2.16. Let $\Delta = \delta_1\delta_2\ldots$, where $\delta_i \in \{\mathtt{0}, \mathtt{1}\}$ for all $i \in \mathbb{N}$. The infinite word $\mathbf{w}(\Delta, E^\omega)$, called *pseudostandard sequence*, is the sequence having prefixes w_n obtained from the recurrence relation $w_{n+1} = (w_n\delta_{n+1})^E$, $w_0 = \varepsilon$. The sequence Δ is called the *directive sequence* of the infinite word $\mathbf{w}(\Delta, E^\omega)$.

Theorem 2.17. *Let w_n be a non-empty antipalindromic prefix of $\mathbf{w}(\Delta, E^\omega)$, where the prefix of Δ of length n contains at least two $\mathtt{0}$'s and one $\mathtt{1}$, and $\Delta = \mathtt{0}\cdots$. Then, when indexing from 0, a minimum size attractor is equal to $\Gamma = \{m_0, m_1, |w_n| - m_1 - 1\}$,*
where $m_a = max\{|q| : q$ is an antipalindrome and qa is a prefix of $w_n\}$.

Remark 2.18. For Δ starting with $\mathtt{1}$ the statement and the proof are analogous.

Example 2.19. Let us have a directive sequence $\Delta = \mathtt{01001}\cdots$. Then the generation of the pseudostandard sequence unfolds as follows (attractors are underlined):

$$
\begin{aligned}
w_0 &= \varepsilon\\
w_1 &= \underline{\mathtt{01}}\\
w_2 &= \underline{\mathtt{0}}\mathtt{1}\underline{\mathtt{10}}\mathtt{01}\\
w_3 &= \mathtt{01}\underline{\mathtt{1}}\mathtt{001}\underline{\mathtt{0}}\mathtt{11}\underline{\mathtt{0}}\mathtt{01}\\
w_4 &= \mathtt{01}\underline{\mathtt{1}}\mathtt{001011001}\underline{\mathtt{0}}\mathtt{11}\underline{\mathtt{0}}\mathtt{01}\\
w_5 &= \mathtt{011001011001}\underline{\mathtt{0}}\mathtt{11}\underline{\mathtt{0}}\mathtt{01}\underline{\mathtt{1}}\mathtt{001011001011001}
\end{aligned}
$$

Note that the attractor form of w_i corresponds with the theorem statement for $i \geq 3$, where both letters have already appeared in the directive sequence Δ.

Proof. Firstly, using the mathematical induction on n, we will show that the set $\{m_0, m_1, |w_n| - m_1 - 1\}$ forms an attractor. Secondly, we will explain its minimality.

- The initial step is for $n = k$, where $\Delta = \mathtt{0}^{k-1}\mathtt{1}\cdots$ for $k > 1$. Then the word is of the form $w_k = (\mathtt{01})^{k-1}\mathtt{10}(\mathtt{01})^{k-1} = \mathtt{0101}\cdots\mathtt{01}\underline{\mathtt{0}}\mathtt{1}\underline{\mathtt{10}}\mathtt{0101}\cdots\mathtt{0101}$. The underlined positions clearly form an attractor.

  Another initial step complying with the requirements from the statement could be $\Delta = \mathtt{010}\cdots$. In that case, the word is $w_3 = \mathtt{01}\underline{\mathtt{1}}\mathtt{001}\underline{\mathtt{0}}\mathtt{11}\underline{\mathtt{0}}\mathtt{01}$ and the underlined positions form an attractor.

- For the next generation step $n - 1 \to n$, two cases may occur:

  1. The next letter is $\mathtt{1}$ and we obtain

$$
w_n = (w_{n-1}\mathtt{1})^E = \underbrace{v\mathtt{0}\overbrace{w_{l_1}}}_{w_{n-1}}{}^{\!\!\!\!\!\!\!\!\!\!\!\!\overbrace{\phantom{w_{l_1}\mathtt{1}E(v)}}^{w_{n-1}}}\mathtt{1}E(v)\,, \tag{1}
$$

  where $w_{l_1} \neq \varepsilon$ is the longest antipalindromic prefix followed by $\mathtt{1}$ in w_{n-1}, and v is the prefix of the word. Let $w_{l_0} \neq \varepsilon$ denote the longest antipalindromic prefix followed by $\mathtt{0}$ in w_{n-1}. Notice that the longest antipalindromic prefix of w_n followed by $\mathtt{1}$ is equal to $w_{n-1} = v\mathtt{0}w_{l_1}$ and the longest antipalindromic prefix of w_n followed by $\mathtt{0}$ is the same as for w_{n-1}, i.e., $m_0 = |w_{l_0}|$. Consequently, we want to show that the attractor of w_n is formed by the underlined positions $w_n = v\underline{\mathtt{0}}w_{l_1}\underline{\mathtt{1}}E(v) = w_{l_0}\underline{\mathtt{0}}\cdots$.

  Let us take an arbitrary factor in the word w_n. It either crosses the underlined $\mathtt{1}$, i.e., the position $|w_{n-1}|$, or it does not cross the position $|w_{n-1}|$ and is entirely contained in the prefix w_{n-1}. Using the induction hypothesis, the attractor of w_{n-1} is formed by the underlined positions $w_{n-1} = v\underline{\mathtt{0}}w_{l_1} = w_{l_0}\underline{\mathtt{0}}\cdots = w_{l_1}\underline{\mathtt{1}}\cdots$. The first two cases coincide with two positions of the expected attractor of w_n. If the factor crosses the position $|w_{l_1}|$, then observing (1), the factor has also an occurrence in w_n crossing $|w_{n-1}|$, which is a contradiction.

2. The next letter is `0` and we have

$$w_n = (w_{n-1}\texttt{0})^E = \underbrace{v\texttt{1}\overbrace{w_{l_0}}}_{w_{n-1}}\!\overbrace{\texttt{0}E(v)}^{w_{n-1}}\,.$$

   Again the longest antipalindromic prefix followed by `1` in w_{n-1} must comply $w_{l_1} \neq \varepsilon$, similarly $w_{l_0} \neq \varepsilon$. Notice that the longest antipalindromic prefix of w_n followed by `0` is equal to w_{n-1} and the longest antipalindromic prefix of w_n followed by `1` is the same as for w_{n-1}, i.e., $m_1 = |w_{l_1}|$. Therefore, we want to show that the attractor of w_n is formed by the underlined positions $w_n = w_{l_1}\underline{\texttt{1}}z\underline{\texttt{0}}E(v) = v\texttt{1}E(z)\underline{\texttt{0}}w_{l_1}$, where z is a factor of w_n.

   The rest of the proof is analogous to the part 1.

Now let us comment on the attractor's minimality.

- As soon as $\texttt{0}^k\texttt{1}$ for $k \geq 2$ is a prefix of Δ, then $w_{k+1} = (\texttt{01})^{k-1}\texttt{011001}(\texttt{01})^{k-1}$. It follows from Definition 2.21 that for each $n \geq k+1$, the factor `00`, resp. `11`, only occurs as a factor of `011001` in w_n.

  An attractor of size two does not exist any more. Let us explain why. All factors of length two, i.e., $\texttt{00}, \texttt{01}, \texttt{10}, \texttt{11}$, have to cross the attractor. We underline below the possible positions for an attractor of size two containing $\texttt{00}, \texttt{01}, \texttt{10}, \texttt{11}$:

  1. $w_n = \cdots\texttt{01}\underline{\texttt{1}}\texttt{0}\underline{\texttt{0}}\texttt{1}\cdots$
  2. $w_n = \cdots\texttt{0}\underline{\texttt{1}}\texttt{1}\underline{\texttt{0}}\texttt{01}\cdots$
  3. $w_n = \cdots\texttt{0110}\underline{\texttt{0}}\texttt{1}\cdots\texttt{01}\underline{\texttt{1}}\texttt{001}\cdots$
  4. $w_n = \cdots\texttt{011}\underline{\texttt{0}}\texttt{01}\cdots\texttt{0}\underline{\texttt{1}}\texttt{1001}\cdots$
  5. $w_n = \cdots\texttt{01}\underline{\texttt{1}}\texttt{001}\cdots\texttt{0110}\underline{\texttt{0}}\texttt{1}\cdots$
  6. $w_n = \cdots\texttt{0}\underline{\texttt{1}}\texttt{1001}\cdots\texttt{011}\underline{\texttt{0}}\texttt{01}\cdots$

  However, in all the above cases, either `010`, or `101` does not cross the attractor.

- If $\texttt{01}^k\texttt{0}$ for $k \geq 1$ is a prefix of Δ, then $w_{k+2} = \texttt{01}(\texttt{1001})^k(\texttt{0110})^k\texttt{01}$. It follows from Definition 2.21 that for each $n \geq k+2$, the factor `00`, resp. `11`, only occurs as a factor of `011001` in w_n. An attractor of size two does not exist. All factors of length two, i.e., $\texttt{00}, \texttt{01}, \texttt{10}, \texttt{11}$, have to cross the attractor. Using the same arguments as in the first case, we can then show that either the factor `010` or `101` does not cross the attractor.

□

The algorithm to generate pseudostandard words and their attractors in accordance with Theorem 2.17 is described as Algorithm 2.

Remark 2.20. In the assumption of Theorem 2.17, the prefix of the directive sequence Δ is restricted to contain at least two `0`'s and one `1` (and without loss of generality starting with `0`). Let us explain that when a prefix of length n of Δ does not comply with this rule, the corresponding prefix w_n has an attractor of size two. Two situations may occur.

- If $\texttt{01}^{n-1}$ for $n \geq 2$ is a prefix of Δ, then $w_n = \texttt{01}\underline{\texttt{1}}\texttt{0}\underline{\texttt{0}}\texttt{1}(\texttt{1001})^{n-2}$, where we underlined the positions of an attractor of size two. For comparison, here we underlined the position of Γ as obtained from the theorem statement for $n \geq 3$: $w_n = \underline{\texttt{0}}\texttt{11}\underline{\texttt{0}}\texttt{01}(\texttt{1001})^{n-3}\underline{\texttt{1}}\texttt{001}$ and for $n = 2$: $w_n = \underline{\texttt{0}}\texttt{1}\underline{\texttt{1}}\underline{\texttt{0}}\texttt{01}$.

- If $\texttt{0}^n$ is a prefix of Δ, then $w_n = (\texttt{01})^n$ and any two positions of `0` and `1` (except for the set $\{0, n-1\}$) form an attractor.

2.3 Combination of closures and Rote sequences

We have introduced sequences generated by palindromic and antipalindromic closures and their attractors, so it is only natural to study attractors of infinite words generated by a combination of these two methods. We call a word (or a closure) pseudopalindromic if it is either palindromic or antipalindromic. In this section, we will describe the minimal attractors of pseudopalindromic prefixes of complementary-symmetric Rote sequences, which form a subclass of generalized pseudostandard sequences. Such sequences are obtained by expanding a directive sequence by pseudopalindromic closures.

Definition 2.21. Let $\Delta = \delta_1\delta_2\ldots$ and $\Theta = \vartheta_1\vartheta_2\ldots$, where $\delta_i \in \{\mathtt{0},\mathtt{1}\}$ and $\vartheta_i \in \{E, R\}$ for all $i \in \mathbb{N}$. The sequence $\mathbf{w}(\Delta,\Theta)$, called *generalized pseudostandard sequence*, is the sequence having prefixes w_n obtained from the recurrence relation

$$w_{n+1} = (w_n\delta_{n+1})^{\vartheta_{n+1}},$$

$$w_0 = \varepsilon.$$

The bi-sequence (Δ,Θ) is called the *directive bi-sequence* of the infinite word $\mathbf{w}(\Delta,\Theta)$.

Example 2.22. Consider $\mathbf{w} = \mathbf{w}(\Delta,\Theta)$ with $\Delta = \mathtt{0}\mathtt{1}^\omega$ and $\Theta = (RE)^\omega$, then $\mathbf{w}$ is the Thue-Morse sequence. Here are the first six pseudopalindromic prefixes:

$$\begin{aligned} w_0 &= \varepsilon \\ w_1 &= \mathtt{0} \\ w_2 &= \mathtt{01} \\ w_3 &= \mathtt{0110} \\ w_4 &= \mathtt{01101001} \\ w_5 &= \mathtt{0110100110010110} \end{aligned}$$

Definition 2.23. *Complementary-symmetric (CS) Rote sequences* are binary sequences having complexity $2n$ and such that their language is closed under letter exchange.

Besides being generalized pseudostandard sequences, the CS Rote sequences are also closely connected to Sturmian sequences by a structural theorem.

Let $u = u_0u_1\cdots u_n$ be a binary word on $\{\mathtt{0},\mathtt{1}\}$ of length at least two. The *sum of* u, denoted by $S(u)$, is the word $v = v_0v_1\cdots v_{n-2}$ defined by

$$v_i = (u_{i+1} + u_i) \bmod 2, \quad \text{for } i \in \{0,1,\ldots,n-2\}$$

For example, if $u = \mathtt{0011010}$, then $S(u) = \mathtt{010111}$.

Theorem 2.24 (Theorem 3 in [24]). *A binary sequence* $\mathbf{w}$ *is a CS Rote sequence if and only if the sequence* $S(\mathbf{w})$ *is a Sturmian sequence.*

We say that a CS Rote sequence $\mathbf{w}$ is *standard* if both $\mathtt{0}\mathbf{w}$ and $\mathtt{1}\mathbf{w}$ are CS Rote sequences. Equivalently, a sequence $\mathbf{w}$ is standard CS Rote if and only if $S(\mathbf{w})$ is standard Sturmian.

The connection between the CS Rote sequences and the Sturmian sequences includes also their prefixes. The relation between pseudopalindromic prefixes of a standard CS Rote sequence $\mathbf{w}$ and palindromic prefixes of a standard Sturmian sequence $S(\mathbf{w})$ is as follows.

Lemma 2.25 (Lemma 37 in [25]). *Let* $\mathbf{w}$ *be a standard CS Rote sequence starting with* $\mathtt{0}$. *Let* $u_0 = \varepsilon, u_1, u_2, \ldots,$ *and* $w_0 = \varepsilon, w_1, w_2, \ldots$ *be the pseudopalindromic prefixes of* $S(\mathbf{w})$ *and* $\mathbf{w}$, *respectively, ordered by length. Then* $S(w_{n+1}) = u_n$ *for all* $n \in \mathbb{N}, n \geq 1$.

Remark 2.26. Let us explain that it is not possible to use known attractors of palindromic prefixes of standard Sturmian sequences to obtain attractors of pseudopalindromic prefixes of CS Rote sequences.

Consider the following palindromic prefix $u = \mathtt{010010010}$ of a standard Sturmian sequence. The corresponding standard CS Rote sequence starting in $\mathtt{0}$ has the antipalindromic prefix $w = \mathtt{0011100011}$, i.e., $S(w) = u$. Let us underline the positions of the attractor of u from Theorem 2.6: $u = \mathtt{0}\underline{\mathtt{1}}\mathtt{0010}\underline{\mathtt{0}}\mathtt{10}$ and also from [8] (Theorem 22): $u = \mathtt{010010}\underline{\mathtt{01}}\mathtt{0}$. Now, the factor $\mathtt{10}$ has a unique occurrence in w, therefore each attractor of w has to contain either the position 4 or 5. However, there is no straightforward way how to obtain these positions from the underlined attractors of u (or their mirror image from Observation 2.30).

Blondin-Massé at al. [25] showed that the standard CS Rote sequences form a subclass of binary generalized pseudostandard sequences. Moreover, they described precisely the form of the corresponding directive bi-sequence.

Theorem 2.27 (Corollary 42 in [25])**.** *Let (Δ, Θ) be a directive bi-sequence. Then $\mathbf{w} = \mathbf{w}(\Delta, \Theta)$ is a standard CS Rote sequence if and only if $\mathbf{w}$ is aperiodic and no factor of length two of the directive bi-sequence is in the set*

$$\{(ab, EE) : a, b \in \{\mathtt{0}, \mathtt{1}\}\} \cup \{(a\overline{a}, RR) : a \in \{\mathtt{0}, \mathtt{1}\}\} \cup \{(aa, RE) : a \in \{\mathtt{0}, \mathtt{1}\}\} .$$

Moreover, if Θ does not start in E, then the prefixes w_n from Definition 2.21 coincide with all pseudopalindromic prefixes of $\mathbf{w}$.

The aperiodicity of a binary generalized pseudostandard sequence may be recognized easily.

Theorem 2.28 (Remark 12 in [26])**.** *Let (Δ, Θ) be a directive bi-sequence. Then $\mathbf{w} = \mathbf{w}(\Delta, \Theta)$ is aperiodic if and only if there is no bijection $\pi : \{E, R\} \to \{\mathtt{0}, \mathtt{1}\}$ such that $\pi(\vartheta_n) = \delta_{n+1}$ for all sufficiently large n.*

In the proof of the main theorem on string attractors of pseudopalindromic prefixes of standard CS Rote sequences, the following statements will be useful.

Proposition 2.29 (Proposition 7 [27])**.** *Let $\mathbf{u}$ be a standard Sturmian sequence and let $(u_n)_{n=0}^{\infty}$ be the sequence of palindromic prefixes of $\mathbf{u}$ ordered by length. If u_n contains both letters, then for some $a \in \{\mathtt{0}, \mathtt{1}\}$*

$$u_n = u_{n-1} a\overline{a} u_i \, ,$$

where u_i is the longest palindromic prefix of u_n followed by $\overline{a}$.

Observation 2.30. *If w is a palindrome with an attractor Γ, the mirror image $\overline{\Gamma} = \{|w| - 1 - \gamma \ : \gamma \in \Gamma\}$ is an attractor of w, too.*

Lemma 2.31. *Let $\mathbf{w}$ be a standard CS Rote sequence starting in $\mathtt{0}$. Then for $n \geq 2$, using notation from Definition 2.21 and denoting $a := \delta_n$, we have:*

1. *If $w_{n-1} = R(w_{n-1})$ and $w_n = R(w_n)$, then $w_n = w_{n-1} a \overline{w_i}$, where w_i is the longest antipalindromic prefix followed by a.*
2. *If $w_{n-1} = R(w_{n-1})$ and $w_n = E(w_n)$, then $w_n = w_{n-1} a \overline{w_i}$, where w_i is the longest palindromic prefix followed by $\overline{a}$.*
3. *If $w_{n-1} = E(w_{n-1})$ and $w_n = R(w_n)$, then $w_n = w_{n-1} a w_i$, where w_i is the longest palindromic prefix followed by a.*

Proof. Assume $S(w_n)$ contains both letters. The reader is invited to check the cases, where $S(w_n)$ contains only one letter. The possible prefixes of (Δ, Θ) are given in Theorem 2.27.

1. Since $w_n = w_{n-1} a \cdots$ and w_{n-1} as a palindrome has $\mathtt{0}$ as both the first and the last letter, then by Lemma 2.25, Theorem 2.24, and Proposition 2.29, we obtain $S(w_n) = u_{n-1} a \overline{a} u_i$, where u_i is the longest palindromic prefix of u_n followed by $\overline{a}$. Consequently, w_n is equal to $w_{n-1} a w$, where w starts in $\mathtt{1}$ and is a pseudopalindrome by Lemma 2.25. Therefore, since w is a suffix of $w_n = R(w_n)$, we get $w = \overline{w_i}$, where w_i is the longest antipalindromic prefix of w_n followed by a.

2. The proof is the same as before, just the last sentence changes: Therefore, since w is a suffix of $w_n = E(w_n)$, we get $w = \overline{w_i}$, where w_i is the longest palindromic prefix of w_n followed by $\overline{a}$.

3. Since $w_n = w_{n-1} a \cdots$ and w_{n-1} as an antipalindrome has $\mathtt{1}$ as the last letter, then by Lemma 2.25, Theorem 2.24, and Proposition 2.29, we obtain $S(w_n) = u_{n-1} \overline{a} a u_i$, where u_i is the longest palindromic prefix of u_n followed by a. Consequently, w_n is equal to $w_{n-1} a w$, where w starts in $\mathtt{0}$ and is a pseudopalindrome by Lemma 2.25. Therefore, since w is a suffix of $w_n = R(w_n)$, we get $w = w_i$, where w_i is the longest palindromic prefix of w_n followed by a.

□

Let us state the main theorem describing the minimal string attractors of pseudopalindromic prefixes of standard CS Rote sequences.

Theorem 2.32. *Let* $\mathbf{w}$ *be a standard CS Rote sequence, then the size of the minimal attractor of any pseudopalindromic prefix equals the number of letters contained in the prefix. More precisely, if the directive bi-sequence* (Δ, Θ) *has* $(\mathtt{0}, R)$ *as the first element, then the minimal attractors of the pseudopalindromic prefixes of* $\mathbf{w}$ *containing at least two letters are of the following form:*

1. *If* $w_n = E(w_n)$, $\delta_n = a$, *and* w_i *is the longest antipalindromic prefix of* w_n *followed by* $\overline{a}$, *then*
$$\begin{aligned} \Gamma_1 &= \{|w_i|, |w_{n-1}|\}\,; \\ \Gamma_2 &= \{|w_{n-1}| - |w_i| - 1, |w_n| - |w_i| - 1\} \end{aligned}$$
are attractors of w_n.

2. *If* $w_n = R(w_n)$, $\delta_n = a$, $\vartheta_{n-1} = E$, *and* w_j *is the longest palindromic prefix of* w_n *followed by* $\overline{a}$, *then*
$$\Gamma = \{|w_j|, |w_{n-1}|\}$$
is an attractor of w_n.

3. *If* $w_n = R(w_n)$, $\delta_n = a$, $\vartheta_{n-1} = R$, *and* m *is the minimum index satisfying that* $\vartheta_i = R$ *for all* $i \in \{m, \ldots, n\}$, *then the attractor of* w_m *from Item 2 is simultaneously an attractor of* w_n.

Proof. First of all, Theorem 2.27 describes the form of (Δ, Θ) guaranteeing that pseudopalindromic prefixes coincide with the prefixes w_n. It follows that Θ has to start with R. The assumption that $(\mathtt{0}, R)$ is the first element of (Δ, Θ) is thus without loss of generality. If a pseudopalindromic prefix contains one letter, then any position is its attractor. Further on, let us consider pseudopalindromic prefixes w_n containing two letters. Let us proceed by induction on n.

Consider the first pseudopalindromic prefix w_{k+1} containing both letters $\mathtt{0}$ and $\mathtt{1}$. Then by Theorem 2.27 $(\mathtt{0}^k\mathtt{1}, R^kE)$ is a prefix of (Δ, Θ) and $k \geq 1$. Then $w_{k+1} = \mathtt{0}^k\mathtt{1}^k$ and the longest antipalindromic prefix of w_{k+1} followed by $\mathtt{0}$ is $w_0 = \varepsilon$. Hence $\Gamma_1 = \{0, k\}$ and $\Gamma_2 = \{k-1, 2k-1\}$ are clearly attractors of $w_{k+1} = \underline{\mathtt{0}}\mathtt{0}^{k-1}\underline{\mathtt{1}}\mathtt{1}^{k-1} = \mathtt{0}^{k-1}\underline{\mathtt{0}}\mathtt{1}^{k-1}\underline{\mathtt{1}}$ (we underlined the positions of Γ_1, resp. Γ_2).

Now, for $n \geq k+1$, let us denote $a := \delta_n$ and assume $\vartheta_n = \vartheta_{n+m+1} = E$ (by Theorem 2.27 $m \geq 1$), while $\vartheta_i = R$ for all $i \in \{n+1, \ldots, n+m\}$. We assume that w_n has the attractors

$$\begin{aligned} \Gamma_1 &= \{|w_i|, |w_{n-1}|\}\,; \\ \Gamma_2 &= \{|w_{n-1}| - |w_i| - 1, |w_n| - |w_i| - 1\}\,, \end{aligned}$$

where w_i is the longest antipalindromic prefix of w_n followed by $\overline{a}$ (it may be an empty word). We will show that under this assumption, w_{n+1} up to w_{n+m+1} have also the attractors as described in Theorem 2.32. This will prove the theorem completely.

There are four situations to be considered according to Theorem 2.27. We will treat the first two of them. The remaining ones are analogous.

1. $m = 1$ and $(\delta_{n-1}\delta_n\delta_{n+1}\delta_{n+2}, \vartheta_{n-1}\vartheta_n\vartheta_{n+1}\vartheta_{n+2}) = (\overline{a}aa\overline{a}, RERE)$: By Lemma 2.31
$$w_{n-1} = w_j\overline{aw_i} = w_i\overline{a}w_j \quad \text{and} \quad w_n = w_j\overline{aw_i}a\overline{w_j} = w_j\overline{aw_{n-1}}\,, \tag{2}$$
where w_j is the longest palindromic prefix of w_n followed by $\overline{a}$ (it may be ε). Using the different forms of w_{n-1}, we can also rewrite w_n as
$$w_n = w_i\underline{\overline{a}}w_j\underline{a}\overline{w_j} = w_j\overline{\underline{a}w_j}\underline{a}w_i\,, \tag{3}$$
where we underlined the positions from Γ_1 and Γ_2, respectively.
   - Using palindromic closure and Lemma 2.31, we obtain
$$w_{n+1} = (w_na)^R = w_{n-1}a\overline{w_j}aw_{n-1}\,. \tag{4}$$

We will show that $\Gamma = \{|w_j|, |w_n|\}$ is an attractor of w_{n+1} – see the corresponding positions underlined:

$$w_{n+1} = w_j\overline{\underline{a}w_{n-1}}\underline{a}w_{n-1}\,. \tag{5}$$

By (4), each factor f of w_{n+1} either crosses $|w_n|$, i.e., the underlined a in (5), or is entirely contained in w_n. In that case, consider the attractor $\Gamma_2 = \{|w_{n-1}| - |w_i| - 1, |w_n| - |w_i| - 1\} = \{|w_j|, |w_n| - |w_i| - 1\}$ of w_n. Using (2) we can rewrite w_{n+1} as

$$w_{n+1} = \underbrace{w_j\overline{\underline{a}}\overbrace{\overline{w_j}aw_i}^{\overline{w_{n-1}}}}_{w_n}\underline{a}w_j\overline{aw_i} = w_j\overline{\underline{a}w_i}a\overbrace{\overline{w_j}\underline{a}w_i}^{\overline{w_{n-1}}}\overline{a}w_j\,. \tag{6}$$

From this form, we can see that f contained in w_n either crosses $|w_j|$, i.e., the underlined $\overline{a}$, or f crosses the position $|w_n| - |w_i| - 1$ in (6) and is contained in the word $\overline{w_{n-1}} = \overline{w_j}\underline{a}w_i$, where we underlined the position crossed by f. Observing the right-hand form of w_{n+1} in (6), the factor f has then an occurrence containing the position $|w_n|$, i.e., the underlined a.

- Next, using antipalindromic closure, we obtain

$$w_{n+2} = (w_{n+1}\overline{a})^E = \overbrace{w_{n-1}a\overline{w_n}}^{w_{n+1}}\overline{aw_{n-1}}\,. \tag{7}$$

We will show that the attractors of w_{n+2} are:

(a) $\{|w_n|, |w_{n+1}|\}$: Each factor f of $w_{n+2} = w_n\underline{a}w_{n-1}\overline{\underline{a}w_{n-1}}$ (we underlined the positions of the expected attractor) either crosses $|w_{n+1}|$, i.e., the underlined $\overline{a}$, or is entirely contained in $w_{n+1} = w_naw_{n-1}$. In this case, we can write w_{n+2} as

$$w_{n+2} = \overbrace{\underbrace{w_j\overline{aw_j}aw_i}_{w_n}\underline{a}w_i\overline{a}}^{w_{n+1}}\underbrace{w_j\overline{\underline{a}w_j}aw_i}_{w_n}\,. \tag{8}$$

By (5), f then either crosses $|w_n|$, i.e., the underlined a, or f is contained in $w_n = w_j\overline{aw_{n-1}}$ and crosses $|w_j|$. However, since w_n forms also the suffix of w_{n+2}, the factor f has an occurrence containing the position $|w_{n+1}|$, i.e., the underlined $\overline{a}$.

(b) $\{|w_{n+1}| - |w_n| - 1, |w_{n+2}| - |w_n| - 1\} = \{|w_{n-1}|, |w_{n+2}| - |w_n| - 1\}$: Each factor f of $w_{n+2} = w_{n-1}\underline{a}\overline{w_{n-1}\underline{a}}w_n$ (we underlined the positions of the expected attractor) either crosses $|w_{n-1}|$, i.e., the underlined a, or f is entirely contained in $\overline{w_{n+1}} = \overline{w_{n-1}a}w_n$ and, by Observation 2.30, crosses the mirror image of the attractor of w_{n+1}, i.e., $\overline{\Gamma} = \{|w_{n+1}| - |w_n| - 1, |w_{n+1}| - |w_j| - 1\}$ (taking the indices only in $\overline{w_{n+1}}$). The situation can be depicted as

$$w_{n+2} = \overbrace{w_j\overline{aw_i}\underline{a}\overline{w_j}}^{w_n}aw_i\overline{\underline{a}}\overbrace{w_j\overline{aw_i}a\overline{w_j}}^{w_n}\,,\qquad \underbrace{\overline{w_j}aw_i\overline{\underline{a}}w_j\overline{aw_i}a\overline{w_j}}_{\overline{w_{n+1}}}\,. \tag{9}$$

We can observe that if f crosses the position $|w_{n+1}| - |w_n| - 1$ in $\overline{w_{n+1}}$, then f crosses $|w_{n+2}| - |w_n| - 1$ in w_{n+2}, i.e., the underlined $\overline{a}$. Or f is entirely contained in $w_n = w_j\overline{aw_i}\underline{a}\overline{w_j}$ and crosses the underlined a. Since w_n is a prefix of w_{n+2}, it follows that the factor f has then an occurrence containing the position $|w_{n-1}|$, i.e., the underlined a.

2. $m \geq 2$ and $(\delta_{n-1}\delta_n\delta_{n+1}\cdots\delta_{n+m}\delta_{n+m+1}, \vartheta_{n-1}\vartheta_n\vartheta_{n+1}\cdots\vartheta_{n+m}\vartheta_{n+m+1}) = (\overline{a}a^{m+1}\overline{a}, RER^mE)$: The proof that $\Gamma = \{|w_j|, |w_n|\}$ is an attractor of w_{n+1} stays the same as above.

- Using palindromic closure, since w_{n-1} is the longest palindromic prefix of w_{n+1} followed by a, we obtain

$$w_{n+2} = (w_{n+1}a)^R = w_naw_{n-1}a\overline{w_n} = w_naw_{n+1}\,. \tag{10}$$

We will show that the attractor of w_{n+2} is the same as the attractor $\Gamma = \{|w_j|, |w_n|\}$ of w_{n+1}. Indeed, each factor f either crosses $|w_n|$ or by (10) f is entirely contained in

the prefix w_{n+1} of w_{n+2} and crosses an element of Γ.
Similarly, for $k \in \{3, \dots, m\}$, we have $w_{n+k} = w_n a w_{n+k-1}$. The attractor of w_{n+k} is again equal to Γ: each factor f either crosses $|w_n|$, or f is entirely contained in the prefix w_{n+k-1} of w_{n+k} and the attractor of w_{n+k-1} is by induction assumption $\Gamma = \{|w_j|, |w_n|\}$.

- Using antipalindromic closure, since w_n is the longest antipalindromic prefix followed by a, we have
$$w_{n+m+1} = (w_{n+m}\overline{a})^E = w_{n+m-1}a\overline{w_n}a\overline{w_{n+m-1}}\,. \tag{11}$$
We will show that the attractors of w_{n+m+1} are:
(a) $\{|w_n|, |w_{n+m}|\}$: Each factor f of $w_{n+m+1} = w_n\underline{a}w_{n+m-1}\underline{\overline{a}}\overline{w_{n+m-1}}$ (we underlined the positions of the expected attractor) either crosses $|w_{n+m}|$, i.e., the underlined $\overline{a}$, or by (11) f is entirely contained in w_{n+m} or in $\overline{w_{n+m}}$ (a suffix of w_{n+m+1}).
Using the previous expressions, the word w_{n+m+1} can be expressed as
$$w_{n+m+1} = \underbrace{w_n\underline{a}w_{n+m-2}a\overline{w_{n-1}}\rlap{\overbrace{\phantom{\overline{a}w_j\underline{\overline{a}}}}^{w_n}}\overline{a}w_j}_{w_{n+m}}\underbrace{\underline{\overline{a}}\overline{w_{n-1}}aw_j\overline{aw_{n+m-2}}}_{\overline{w_{n+m-1}}} = \tag{12}$$
$$= \overbrace{w_{n-1}a\rlap{\underbrace{\phantom{\overline{w_j}\underline{a}w_{n-1}}}_{\overline{w_n}}}\overline{w_j}}^{w_n}\underline{a}w_{n-1}a\overline{w_j}aw_{n+m-2}\overline{\underline{a}w_{n+m-1}}\,. \tag{13}$$
If f is contained in w_{n+m}, then f crosses the attractor $\Gamma = \{|w_j|, |w_n|\}$ of w_{n+m}. It means that either f crosses $|w_n|$, i.e., the underlined a, or f is contained in $w_n = w_j\underline{\overline{a}}\overline{w_{n-1}}$, where we underlined the position crossed by f. Then from (12) we can see that the factor w_n has an occurrence such that the a is in the attractor of w_{n+m} and it is therefore covered.
If f is contained in the suffix $\overline{w_{n+m-1}}$ of w_{n+m+1}, then f crosses its attractor $\Gamma = \{|w_j|, |w_n|\}$. By (11) the factor f either crosses $|w_n|$ in $\overline{w_{n+m}}$, which means that f crosses $|w_{n+m}|$ in w_{n+m+1}, or f is contained in the prefix $\overline{w_n} = \overline{w_j}\underline{a}w_{n-1}$, where we underlined the position crossed by f. However, then f has an occurrence in w_{n+m+1} containing $|w_n|$ since w_naw_{n-1} is a prefix of w_{n+m+1} and $\overline{w_j}$ is a suffix of w_n, as can be observed from (13).
(b) $\{|w_{n+m}| - |w_n| - 1, |w_{n+m+1}| - |w_n| - 1\} = \{|w_{n+m-1}|, |w_{n+m+1}| - |w_n| - 1\}$: Each factor f of w_{n+m+1} either crosses $|w_{n+m-1}|$, or by (11) f is entirely contained in w_{n+m} (prefix of w_{n+m+1}) or in $\overline{w_{n+m}}$. The word can be expressed as
$$w_{n+m+1} = w_{n+m-1}\underline{a}\underbrace{\overline{w_{n+m-2}a}w_j\overbrace{\overline{aw_{n-1}\underline{a}}w_j}^{\overline{w_n}}\overline{aw_{n-1}}}_{\overline{w_{n+m}}} \tag{14}$$
$$= \underbrace{w_{n+m-2}a\overline{w_j}\overbrace{aw_{n-1}\underline{a}\overline{w_j}}^{w_n}aw_{n-1}}_{w_{n+m}}\overline{aw_{n+m-2}\underline{a}}w_n \tag{15}$$
If f is contained in $w_{n+m} = w_{n+m-1}a\overline{w_n}$, then by Observation 2.30, f crosses the mirror image of the attractor of w_{n+m}, i.e., $\overline{\Gamma} = \{|w_{n+m}|-|w_n|-1, |w_{n+m}|-|w_j|-1\}$. Then f either crosses $|w_{n+m}| - |w_n| - 1 = |w_{n+m-1}|$ or f is contained in the suffix $\overline{w_n} = \overline{w_{n-1}\underline{a}}w_j$ of w_{n+m}, where the position crossed by f is underlined. Then we can observe from (14) that f has an occurrence containing the position $|w_{n+m+1}| - |w_n| - 1$, i.e., the underlined $\overline{a}$.
If f is contained in $\overline{w_{n+m}} = \overline{w_{n+m-1}a}w_n$, then f crosses the mirror image of the attractor of $\overline{w_{n+m}}$, i.e., $\overline{\Gamma} = \{|w_{n+m}| - |w_n| - 1, |w_{n+m}| - |w_j| - 1\}$ (taking the indices in $\overline{w_{n+m}}$). The factor is either covered by the position $|w_{n+m-1}|$ in $\overline{w_{n+m}}$, which corresponds with the position $|w_{n+m+1}| - |w_n| - 1$ in the attractor, or f is entirely contained in $w_n = w_{n-1}\underline{a}\overline{w_j}$, where we underlined the position crossed by f. By (15), the factor f has then an occurrence containing $|w_{n+m}| - |w_n| - 1$, i.e., the underlined a.

3. $m = 1$ and $(\delta_{n-1}\delta_n\delta_{n+1}\delta_{n+2}, \vartheta_{n-1}\vartheta_n\vartheta_{n+1}\vartheta_{n+2}) = (\overline{a}a\overline{a}a, RERE)$

4. $m \geq 2$ and $(\delta_{n-1}\delta_n\delta_{n+1}\cdots\delta_{n+m}\delta_{n+m+1}, \vartheta_{n-1}\vartheta_n\vartheta_{n+1}\cdots\vartheta_{n+m}\vartheta_{n+m+1}) = (\overline{a}a\overline{a}^m a, RER^m E)$

□

Example 2.33. To illustrate the case of $m = 1$ we will generate prefixes of a Rote sequence from the bi-sequence starting in $(\mathtt{0011001}, RRERERE)$. The attractors from Theorem 2.32 are underlined.

$$
\begin{aligned}
w_1 &= \underline{\mathtt{0}}\\
w_2 &= \mathtt{0}\underline{\mathtt{0}}\\
w_3 &= \overbrace{\underline{\mathtt{0}}\mathtt{0}}^{w_2}\ \underline{\mathtt{1}}\mathtt{1} = \mathtt{0}\underline{\mathtt{0}}\ \overbrace{\mathtt{1}\underline{\mathtt{1}}}^{\overline{w_2}}\\
w_4 &= \underbrace{\overbrace{\ \mathtt{0}\ }^{w_1}\underline{\mathtt{0}}\mathtt{11}}_{w_3}\underline{\mathtt{1}}\mathtt{00}\\
w_5 &= \underbrace{\overbrace{\mathtt{0011}}^{w_3}\underline{\mathtt{1}}\mathtt{00}}_{w_4}\underline{\mathtt{0}}\mathtt{11} = \mathtt{00}\underline{\mathtt{1}}\underbrace{\mathtt{11}\underline{\mathtt{0}}\overbrace{\mathtt{0011}}^{w_3}}_{\overline{w_4}}\\
w_6 &= \underbrace{\overbrace{\ \mathtt{00}\ }^{w_2}\underline{\mathtt{1}}\mathtt{1100011}}_{w_5}\underline{\mathtt{0}}\mathtt{0011100}\\
w_7 &= \underbrace{\overbrace{\mathtt{0011100011}}^{w_5}\underline{\mathtt{0}}\mathtt{0011100}}_{w_6}\underline{\mathtt{1}}\mathtt{1100011} = \mathtt{0011100}\underline{\mathtt{0}}\underbrace{\mathtt{1100011}\underline{\mathtt{1}}\overbrace{\mathtt{0011100011}}^{w_5}}_{\overline{w_6}}
\end{aligned}
$$

For illustration of the case $m > 1$ we will use the bi-sequence starting in $(\mathtt{001100001}, RRERERRRE)$. The steps for $n \leq 5$ are identical with the previous example.

$$
\begin{aligned}
w_1 &= \underline{\mathtt{0}}\\
w_2 &= \mathtt{0}\underline{\mathtt{0}}\\
w_3 &= \underline{\mathtt{0}}\mathtt{0}\underline{\mathtt{1}}\mathtt{1} = \mathtt{0}\underline{\mathtt{0}}\mathtt{1}\underline{\mathtt{1}}\\
w_4 &= \mathtt{0}\underline{\mathtt{0}}\mathtt{11}\underline{\mathtt{1}}\mathtt{00}\\
w_5 &= \mathtt{00}\underline{\mathtt{1}}\mathtt{11}\underline{\mathtt{0}}\mathtt{0011} = \mathtt{0011}\underline{\mathtt{1}}\mathtt{00}\underline{\mathtt{0}}\mathtt{11}\\
w_6 &= \underbrace{\overbrace{\ \mathtt{00}\ }^{w_2}\underline{\mathtt{1}}\mathtt{1100011}}_{w_5}\underline{\mathtt{0}}\mathtt{0011100}\\
w_7 &= \underbrace{\overbrace{\ \mathtt{00}\ }^{w_2}\underline{\mathtt{1}}\mathtt{1100011}}_{w_5}\underline{\mathtt{0}}\mathtt{001110001100011100}\\
w_8 &= \underbrace{\overbrace{\ \mathtt{00}\ }^{w_2}\underline{\mathtt{1}}\mathtt{1100011}}_{w_5}\underline{\mathtt{0}}\mathtt{00111000110001110001100011100}\\
w_9 &= \underbrace{\overbrace{\mathtt{0011100011}}^{w_5}\underline{\mathtt{0}}\mathtt{00111000110001110001100011100}}_{w_8}\underline{\mathtt{1}}\mathtt{11000111001110001110011100011}\\
&= \mathtt{00111000110001110001100011100}\underline{\mathtt{0}}\underbrace{\mathtt{11000111001110001110011100011}\underline{\mathtt{1}}\overbrace{\mathtt{0011100011}}^{w_5}}_{\overline{w_8}}
\end{aligned}
$$

3 Algorithms and software implementations

To study minimal attractors of episturmian and pseudostandard words on practical examples, we designed a series of algorithms to generate the words and obtain their attractors, and implemented them in Python. The programs described below were also used to generate examples in this work and to apply the theorems on larger datasets of words that would be hard to manage manually.

3.1 Episturmian words

First, we designed and implemented an algorithm to generate a palindromic prefix of the episturmian sequence and its minimal attractor in accordance with Theorem 2.11 (Algorithm 1). For a user-provided prefix of a directive sequence `dir_seq`, the function uses `prefix` to store the generated prefix, which is then one of the outputs. It uses dictionary `lpp_index` to store the distinct letters as keys, with the starting indexes of the longest palindromic prefixes followed by these letters as corresponding values. The attractor is stored in the list `attractor`; it is exactly the set of positions stored in `lpp_index`, which is in accordance with Theorem 2.11.

The algorithm then iterates through the directive sequence, appending letters to the prefix. The palindromic closure is performed using the knowledge of the longest palindromic prefixes followed by distinct letters, as we did in the proof of Theorem 2.6. When creating the palindromic closure, we add one letter and then look for the longest palindromic suffix formed with the added letter, and then copy the rest of the word in reversed order. However, this process is equivalent to searching for the longest palindromic prefix followed by the same letter that we added. We have this information stored in `lpp_index`, which means that we only have to copy the letters from the index `len(prefix) - lpp_index[dir_seq[i]] - 2` to the beginning, where `dir_seq[i]` is the letter that we added from the directive sequence, `lpp_index[dir_seq[i]]` is the starting index of the longest palindromic prefix followed by the letter `dir_seq[i]`, and `len(prefix)` is the current length of the prefix. The copy process is done by the function `complete_from(prefix, position)`.

Algorithm 1: Function to generate an episturmian word and compute its associated minimum attractor from a user-provided directive sequence

```python
def palindromic_attractor(dir_seq):
    attractor = set()  # set to store the attractor
    prefix = []  # prefix of the generated word
    lpp_index = {}  # indexes of the longest palindromic prefixes
        followed by distinct letters

    for i in range(len(dir_seq)):
        prefix.append(dir_seq[i])  # add letter to prefix
        to_index = len(prefix)-1
        if dir_seq[i] in lpp_index:  # the letter has already appeared
            _complete_from(prefix, len(prefix)-lpp_index[dir_seq[i]]-2)
        else:
            _complete_from(prefix, len(prefix)-1)
        lpp_index[dir_seq[i]] = to_index  # update the list of prefixes
        attractor = set(lpp_index.values())
    return prefix, attractor

# function to compute the palindromic closure of a given string
def _complete_from(prefix, position):
    index = position - 1
    while index != -1:
        prefix.append(prefix[index])
        index -= 1
    return prefix
```

3.2 Pseudostandard words

Second, we designed and implemented an algorithm to generate antipalindromic prefixes of the pseudostandard sequence based on the given prefix of the directive sequence, and its minimal attractor in accordance with Theorem 2.17 (Algorithm 2). The input is again the finite prefix of the directive sequence `dir_seq`, the output is the generated word `prefix` and its attractor stored in `attractor`. To generate the prefix, we use the longest antipalindromic prefixes followed by distinct letters (stored in `lap_index`) and the function `anticomplete_from(prefix, position)` that generates the antipalindromic closure analogously to the previous program. After the copy process, we add the third, mirrored, position to the attractor.

Algorithm 2: Function to generate a pseudostandard word and compute its associated minimum attractor from a user-provided directive sequence

```python
def antipalindromic_attractor(dir_seq):
    attractor = set()  # set to store the attractor
    prefix = []  # prefix of the generated word
    lap_index = {}  # indexes of the longest antipalindromic prefixes
        followed by distinct letters

    for i in range(len(dir_seq)):
        prefix.append(dir_seq[i])  # add letter to prefix
        to_index = len(prefix)-1
        if dir_seq[i] in lap_index:  # the letter has already appeared
            _anticomplete_from(prefix, len(prefix)-lap_index[dir_seq[i
                ]]-2)
        else:
            _anticomplete_from(prefix, len(prefix))
        lap_index[dir_seq[i]] = to_index
        attractor = set(lap_index.values())
        # adding the third position to attractor
        if(len(lap_index.values()) == 1):
            attractor.add(len(prefix)-lap_index[dir_seq[0]]-1)
        else:
            attractor.add(len(prefix)-lap_index[abs(1-dir_seq[0])]-1)
    return prefix, attractor

# function to compute the antipalindromic closure of a given string
def _anticomplete_from(prefix, position):
    index = position-1
    while index > -1:
        prefix.append(abs(1-prefix[index]))
        index -= 1
    return prefix
```

3.3 General attractor verifier

We then developed and implemented an algorithm to verify whether the given set of positions forms an attractor of the given word (Algorithm 3). This program takes a list `word` and a set of positions `attractor` as an input and verifies if the positions form an attractor of the given word. The output is the shortest factor that does not have an occurrence crossed by any of the positions from `attractor`. If all the factors are covered, it returns `None`.

The algorithm processes the sections in between the given positions one by one and verifies the subfactors in the same way as in Example 1.2. It starts from individual letters and gradually increases the verified factors' length. Once an occurrence of the given factor is found crossing a position from the presumed attractor, it is stored in a dictionary, with the given position as value and the verified factor as key. The program then does not have to look through the word for each factor, but it verifies first whether it is contained in the dictionary. In case the factor is not in the dictionary and it does not have an occurrence crossed by any position from the expected attractor, the factor is returned and the algorithm ends. Otherwise, it ends after processing all the factors with the conclusion that the given positions form an attractor for the given word.

Algorithm 3: Verifying whether a given set of positions forms an attractor

```
def verify_attractor(word, attractor):
    attractor = list(sorted(attractor))  # sorts positions in increasing order
    start_of_section = 0
    verified_factors = {}  # stores factors crossed by positions from attractor

    # verifying sections in between the positions from attractor
    for i in range(len(attractor)):
        if start_of_section != attractor[i]:
            missing_factor = _verify_subfactors(start_of_section, attractor[i]-1, word, attractor, verified_factors)
            if missing_factor:  # some factor is not covered
                return missing_factor
        start_of_section = attractor[i]+1  # move to the next section

    # verifying the last section separately
    if len(word)-1 != attractor[-1]:
        missing_factor = _verify_subfactors(attractor[-1]+1, len(word)-1, word, attractor, verified_factors)
        if missing_factor:
            return missing_factor
    return None

# returns subfactor of the word that is not covered
def _verify_subfactors(start, end, word, attractor, factors):
    # iterates through different lengths of factors
    for i in range(1, end-start+2):
        for j in range(start, end+2-i):
            current = word[j:j+i]
            # if it has not been already verified before
            if str(current) not in factors.keys():
                result = _crosses_attractor(current, word, attractor)
                if result is not None:  # suitable occurrence was found
                    factors[str(current)] = result
                else:  # this factor is not covered
                    return word[j:j+i]
    return None

# returns which position from attractor is crossed by the given factor
def _crosses_attractor(factor, word, attractor):
    for position in attractor:
        # iterating through possible crossings of factor and position
        for i in range(max(0, position-len(word)+len(factor)), min(len(factor)-1, position)+1):
            is_in = True
            # verifying individual letters
            for j in range(len(factor)):
                if factor[j] != word[position-i+j]:
                    is_in = False
                    break
            if is_in:
                return position
    return None
```

3.4 General attractor generator

In case there are no structural assumptions on the word for which we try to find an attractor, we are tackling an NP-complete problem. The brute-force approach to this is to try all the combinations of positions and for each verify whether it forms an attractor. We used this simple solution to test our conjectures (e.g. to disprove pseudostandard words having a minimal attractor of size 2).

The input is a list `word` and an integer `limit`, which sets the maximum number of positions that will be verified whether they form an attractor. The output is the smallest possible list of positions that form an attractor of the given word. If such combination does not exist (within the range of the `limit`), the output is `None`.

The program uses the attractor verifier from Algorithm 3 and the package `itertools` to generate the possible combinations of the given number of positions.

Algorithm 4: Brute-force approach to find a minimum attractor of a given word

```python
def find_attractor(word, limit):
    indexes = [i for i in range(len(word))]  # all indexes in the word
    min_attr_size = len(set(word))  # number of distinct letters
    for attr_size in range(min_attr_size, limit):
        for combination in itertools.combinations(indexes, attr_size):
            missing_factor = verify_attractor(word, list(combination))
            # if the combination forms an attractor
            if missing_factor is None:
                return list(combination)
    return None
```

3.5 Implementation and experimental evaluation

Algorithms 1-4 were implemented in Python 3. The Algorithm 4 uses an external library `itertools`. The programs for attractor verification and generation run in exponential time.

We also evaluated the obtained implementation on a standard laptop computer. We found that the running time of the attractor generator is approximately 1 second to either find an attractor of standard Sturmian word of length 90, or of a pseudostandard word of length 80, or of a randomly generated word over 4-letter alphabet of length 20 (although the computation time of the latter varies from 0.5 second to 6 seconds). The limits of the program are very tight, as the generation of the minimal attractor of a randomly generated word over 2-letter alphabet of length 100 takes several minutes, and for 3-letter alphabet it already takes more than an hour. For longer words, the program is practically unusable.

We used the algorithms to obtain valuable insights into the form of attractors of episturmian and pseudostandard words. We were able to test our conjectures on larger datasets and alternatively provide counter-examples.

Conclusion

Our thesis studied string attractors, a recently introduced concept unifying the theory of dictionary compressors. String attractors provide a unified framework to rigorously study and compare the power of various classes of compression methods, including those based on Lempel-Ziv or the Burrows-Wheeler transform.

The primary focus of our work regarded combinatorics on words – to identify the form of an attractor within a specific class of sequences and prove its minimality. We have newly discovered such results for two distinct categories of infinite aperiodic sequences, both of which were generated using pseudopalindromic closures.

Firstly, we discovered the smallest attractor of size 3 for the prefixes of pseudostandard sequences. We worked with antipalindromic closures and formally proved the attractor's minimality.

Secondly, we determined a minimal attractor of size 2 of the pseudopalindromic prefixes of standard complementary-symmetric Rote sequences. This subset of generalized pseudostandard sequences is generated through a specific combination of palindromic and antipalindromic closures, and is connected to standard Sturmian sequences through modular letter difference. We formally proved the attractor's minimality, highlighting that its existence does not trivially follow from the previously known attractors of Sturmian sequences.

Another major contribution of the thesis is the design, implementation, and experimental evaluation of several algorithms to obtain valuable observations to help us determine the form of the attractors, and then to test hypotheses on larger datasets. Firstly, we have presented programs capable of generating the prefixes of episturmian and pseudostandard sequences and their attractors. Both programs use pseudostandard closures, and generate attractors in the form outlined in this work. Additionally, we constructed an attractor verifier that uses a dictionary of factors to assess whether a given set forms an attractor for a given word. Lastly, the latter program is used as a part of a general attractor generator, which essentially utilizes a brute force approach to test which smallest set forms an attractor of a given word. While the software was developed primarily with the goal of building our intuition and examples for the theoretical section, its applicability goes beyond the context of this thesis.

The thesis opens a series of questions for further investigation. Building upon our current understanding of the minimum attractor for the given classes of infinite words, we can ask how the dictionary compression algorithms will behave for the prefixes of these sequences. Specifically, if the size of the attractor remains constant as the prefix length increases, can we expect the compression measure of a given method to also remain constant? What structure will the compressed words have?

Another compelling open problem are the attractors of generalized pseudostandard sequences, which are generated by an arbitrary sequence of pseudopalindromic closures. These infinite words encompass various sequences, including the well-known Thue-Morse sequence, whose attractors are already known. However, the construction of these attractors remains unrelated to closures.

Results concerning attractors in the field of combinatorics on words are rather isolated. This work contributes as one of the building blocks to construct a more comprehensive theory of attractors of infinite words, with potential applications extending to the broader field of bioinformatics.

References

1. 100,000 Genomes Project. [N.d.]. Available also from: `https://www.genomicsengland.co.uk/initiatives/100000-genomes-project`.
2. PREZZA, N. String attractors. 2017. Available from DOI: `10.48550/arXiv.1709.05314`.
3. KEMPA, D.; PREZZA, N. At the Roots of Dictionary Compression: String Attractors. *STOC'18*. 2018. Available from DOI: `10.48550/arXiv.1710.10964`.
4. AKAGI, T.; FUNAKOSHI, M.; INENAGA, S. Sensitivity of string compressors and repetitiveness measures. *Information and Computation*. 2023, vol. 291. Available from DOI: `10.1016/j.ic.2022.104999`.
5. BANNAI, H.; GOTO, K.; ISHIHATA, M.; KANDA, S.; KÖPPL, D.; NISHIMOTO, T. Computing NP-hard Repetitiveness Measures via MAX-SAT. *ESA*. 2022. Available from DOI: `10.48550/arXiv.2207.02571`.
6. KEMPA, D.; POLICRITI, A.; PREZZA, N.; ROTENBERG, E. String Attractors: Verification and Optimization. *ESA*. 2018. Available from DOI: `10.4230/LIPICS.ESA.2018.52`.
7. FUCHS, J.; WHITTINGTON, P. The 2-Attractor Problem is NP-Complete. 2023.
8. MANTACI, S.; RESTIVO, A.; ROMANA, G.; ROSONE, G.; SCIORTINO, M. A combinatorial view on string attractors. *Theoretical Computer Science*. 2021, vol. 850, pp. 236–248.
9. RESTIVO, A.; ROMANA, G.; SCIORTINO, M. String Attractors and Infinite Words. 2022. Available from DOI: `10.48550/arXiv.2206.00376`.
10. DVOŘÁKOVÁ, L. String attractors of episturmian sequences. 2022. Available from DOI: `10.48550/arXiv.2211.01660`.
11. SCHAEFFER, L.; SHALLIT, J. String attractors of automatic sequences. 2020. Available from DOI: `10.48550/ARXIV.2012.06840`.
12. KUTSUKAKE, K.; MATSUMOTO, T.; NAKASHIMA, Y.; INENAGA, S.; BANNAI, H.; TAKEDA, M. On repetitiveness measures of Thue-Morse words. *SPIRE, LNCS, vol. 12303, Springer*. 2020, pp. 213–220.
13. DOLCE, F. String attractors for factors of the Thue-Morse word. *Frid, A., Mercaş, R. (eds) Combinatorics on Words. WORDS 2023. Lecture Notes in Computer Science, vol 13899. Springer, Cham.* 2023.
14. KOCIUMAKA, T.; NAVARRO, G.; PREZZA, N. Towards a definitive measure of repetitiveness. *LATIN, LNCS, vol. 12118, Springer*. 2020, pp. 207–219.
15. GHEERAERT, F.; ROMANA, G.; STIPULANTI, M. String attractors of fixed points of k-bonacci-like morphisms. *WORDS*. 2023. Available from DOI: `10.48550/arXiv.2302.13647`.
16. ZIV, J.; LEMPEL, A. A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory*. 1977, vol. 23, no. 3, pp. 337–343. Available from DOI: `10.1109/TIT.1977.1055714`.
17. BURROWS, M.; WHEELER, D. J. A block sorting lossless data compression algorithm. *SRC Research Report, Digital Equipment Corporation, California*. 1994.
18. KIDA, Takuya; MATSUMOTO, Tetsuya; SHIBATA, Yusuke; TAKEDA, Masayuki; SHINOHARA, Ayumi; ARIKAWA, Setsuo. Collage system: A unifying framework for compressed pattern matching. *Theoretical Computer Science*. 2003, vol. 298(1), pp. 253–272. Available from DOI: `10.1016/S0304-3975(02)00426-7`.
19. CROCHEMORE, M.; VÉRIN, R. On compact directed acyclic word graphs. *Structures in Logic and Computer Science*. 1997, pp. 192–211. Available from DOI: `10.1007/3-540-63246-8_12`.
20. SALOMON, D.; MOTTA, G. *Handbook of Data Compression*. Springer, 2010. ISBN 978-1-84882-902-2.
21. FERRAGINA, P.; MANZINI, G. Opportunistic data structures with applications. 2000. Available from DOI: `10.1109/SFCS.2000.892127`.
22. MORSE, M.; HEDLUND, G. A. Symbolic Dynamics II. Sturmian trajectories. *American Journal of Mathematics*. 1940, vol. 62, pp. 1–42.

23. DROUBAY, X.; JUSTIN, J.; PIRILLO, G. Episturmian words and some constructions of de Luca and Rauzy. *Theoretical Computer Science*. 2001, vol. 255, pp. 539–553.

24. ROTE, G. Sequences with subword complexity $2n$. *Number Theory*. 1994, vol. 46, pp. 196–213. Available from DOI: `10.1006/jnth.1994.1012`.

25. MASSÉ, A. Blondin; PAQUIN, G.; TREMBLAY, H.; VUILLON, L. On generalized pseudostandard words over binary alphabet. *Integer Seq.* 2013, vol. 16, Article 13.2.11.

26. DVOŘÁKOVÁ, L.; FLORIAN, J. On periodicity of generalized pseudostandard words. *Electron. J. Comb.* 2016, vol. 23(1), p. 9.

27. DE LUCA, A.; MIGNOSI, F. Some combinatorial properties of Sturmian words. *Theoretical Computer Science*. 1994, vol. 136, pp. 361–385.