arXiv:2403.15215v1 [cs.DM] 22 Mar 2024

Exploring the Crochemore and Ziv-Lempel factorizations of some automatic sequences with the software `Walnut`

Marieh Jahannia[1][0000−0001−8510−2599] and Manon Stipulanti[2][0000−0002−2805−2465]

[1] School of Mathematics, Statistics and Computer Science, College of Science
University of Tehran, Tehran, Iran
mjahannia@ut.ac.ir
[2] Department of Mathematics, University of Liège, Liège, Belgium,
m.stipulanti@uliege.be

Abstract. We explore the Ziv-Lempel and Crochemore factorizations of some classical automatic sequences making an extensive use of the theorem prover `Walnut`.

Keywords: Combinatorics on words · Crochemore factorization · Ziv-Lempel factorization · Automatic sequences · `Walnut` theorem prover.

2020 Mathematics Subject Classification: 11B85, 68R15

1 Introduction

In the expansive variety of tools at the heart of combinatorics on words, factorizations break down a given sequence into simpler components to provide valuable insights about its properties and behavior. Crochemore [6,7] on the one hand and Lempel and Ziv [14,21] on the other introduced two such distinguished factorizations, taking after their respective names. The first was aimed for algorithm design and builds on repetitive and non-repetitive aspects of sequences. The second has remained a cornerstone of data compression and string processing algorithms, with ongoing discoveries revealing new applications for its use. In the case of infinite words, Berstel and Savelli [2] characterized the Crochemore factorization of Sturmian words, the Thue-Morse sequence and its generalizations, and the period-doubling sequence. Then Ghareghani et al. [10] examined both the Crochemore and Ziv-Lempel factorizations for standard episturmian words. Constantinescu and Ilie [5] characterized the general behavior of the Ziv-Lempel factorization of morphic sequences, depending on the periodicity of the sequence and the growth function of the morphism. Jahannia et al. [11,12] introduced two variations of these factorizations where the factors are required to satisfy an additional property and studied them for the m-bonacci words.

Within combinatorics on words, the study of automatic sequences [1,16,18] has turned out to be a fascinating journey, showing complex structures and patterns defined by fundamental mathematical principles. As their name points it out, these sequences are produced by finite automata with output, but also as fixed points of morphisms. Their generation rules often lead to connections to various branches of mathematics. The Thue-Morse sequence is one of the most famous –if not the most famous– examples and encapsulates the binary encoding of the occurrences of 0's and 1's in binary representations, revealing a self-replicating and non-repetitive structure [1]. The period-doubling sequence, born from the logistic map, exhibits a chaotic behavior and contributes to the field of dynamical systems [8]. The Rudin-Shapiro sequence, recognized for its statistical relationship or mutual dependence to the Golay sequence, unfolds a binary pattern influenced by the existence of certain arithmetic progressions [4]. Lastly, the paper-folding sequence, a classic representation of a fractal, illustrates how simple operations can generate infinitely long sequences [1].

Recent developments in the field of sequence analysis have seen the emergence of systematic and automated decision procedures designed to autonomously determine the validity of a given property for specified sequences, altering the need of human work in proofs. Notably, in the case of automatic sequences, Mousavi [15] and Shallit [17] have made significant contributions by developing the software called `Walnut`. The latter works by representing a sequence as a finite automaton and expressing properties as first-order logic predicates. The decision procedure then translates these predicates into automata, facilitating the identification of representations for which the predicate holds true.

In this paper, we make use of `Walnut` to obtain a precise description of the Crochemore and Ziv-Lempel factorizations of some classical automatic se-

quences. More precisely, our main direction of investigation is the following general question (see Section 2 for precise definitions):

Question. Given an abstract numeration system S and an S-automatic sequence $\mathbf{x}$, is it possible to use `Walnut` to show that the starting positions and lengths of the factors in both the Crochemore and Ziv-Lempel factorizations of $\mathbf{x}$ only depend on the numeration system S?

Using Fici's [9] nice survey of factorizations of the Fibonacci word, we produce a detailed `Walnut` code in Section 3 to answer this question about the Fibonacci word. In Section 4, we apply this code to other classical automatic sequences. We end the paper with Section 5 where we discuss the scope of our method.

2 Background

Combinatorics on words. We let Σ denote a finite set of symbols, called the *letters*, referred to as an *alphabet*. A *word* over Σ is a finite or infinite sequence of letters chosen from Σ. In this paper, to differentiate finite and infinite words, we write the latter in bold. As usual, we let Σ^* represent the set of finite words over Σ, and ε denote the *empty* word. For a word $w \in \Sigma^*$, we let $|w|$ denote its length. For all $i \in \{0, \ldots, |w| - 1\}$, we use $w[i]$ to refer to the i-th letter of w, starting from position 0. If we write $w = w_0 w_1 \cdots w_{|w|-1}$, then we let $\widetilde{w}$ denote its *reversal* or *mirror*, defined as $\widetilde{w} = w_{|w|-1} \cdots w_1 w_0$. A *factor* of w is a contiguous block of letters within w; we write $w[i..j]$ to represent the factor occupying positions $i, i+1, \ldots, j$. For instance, if $w = \texttt{computer}$, then $v = \texttt{comp}$ is a factor of w where $v = w[0..3]$. A *prefix* (resp., *suffix*) of w is a word x such that $w = xy$ (resp., $w = yx$) for some word y. For instance, `comp` is a prefix and `er` is a suffix of `computer`. If $w = xy$, we write $x^{-1}w = y$ and $wy^{-1} = x$.

A *morphism* is a mapping $\psi\colon \Sigma^* \to \Sigma^*$ such that for all $u, v \in \Sigma^*$, $\psi(uv) = \psi(u)\psi(v)$. A morphism ψ is *k-uniform* if there exists an integer k such that $|\psi(a)| = k$ for all $a \in \Sigma$. A *coding* is a 1-uniform morphism. The morphism ψ is *prolongable* on the letter $a \in \Sigma$ if $\psi(a) = au$ and $\psi^n(u) \neq \varepsilon$ for all $n \geq 0$. A *fixed point* of ψ is given by $\psi^\omega(a) = au\psi(u)\psi^2(u)\cdots$. For example, the morphism $\mu\colon a \mapsto ab, 1 \mapsto ba$ is prolongable on both letters a and b. The infinite word $t = \mu^\omega(a) = abbabaabbaababbabaababba\cdots$ is a fixed point called the *Thue-Morse* sequence.

Factorizations. For a finite word w, a *factorization* of w is a sequence $(x_0, x_1, \ldots, x_m)$ of finite words such that w can be expressed as the concatenation of the elements of the sequence, i.e., $w = x_0 x_1 \cdots x_m$. Similarly, in the case of an infinite word $\mathbf{w}$, the factorization is a sequence $(x_0, x_1, \ldots)$ of finite words such that $\mathbf{w} = x_0 x_1 \cdots$. For example, a factorization of the word $w = \texttt{abracadabra}$ is $(\texttt{ab}, \texttt{ra}, \texttt{ca}, \texttt{da}, \texttt{bra})$. We now introduce two distinguished factorizations of words. More formally, given an infinite word $\mathbf{w}$, its *Ziv-Lempel factorization*, in short *z-factorization*, is given by $z(\mathbf{w}) = (z_0, z_1, \ldots)$, where z_m is the shortest prefix of $z_m z_{m+1} \cdots$ occurring only once in the word $z_0 \cdots z_m$. The *Crochemore factorization*, in short *c-factorization*, of $\mathbf{w}$ is given by $c(\mathbf{w}) =$

$(c_0, c_1, \ldots)$, where c_m is either the longest prefix of $c_m c_{m+1} \cdots$ occurring twice in $c_0 \cdots c_m$ or a letter not present in $c_0 \cdots c_{m-1}$. Roughly, the Ziv-Lempel factorization breaks the sequence into minimal never seen before factors, while the Crochemore one splits the sequence into maximal already seen factors. We consider similar definitions on finite words. For instance, for $w = abbabaabbaababb$, its z- and c-factorizations are respectively $z(w) = (a, b, ba, baa, bbaa, babb)$ and $c(w) = (a, b, b, ab, a, abba, aba, bb)$.

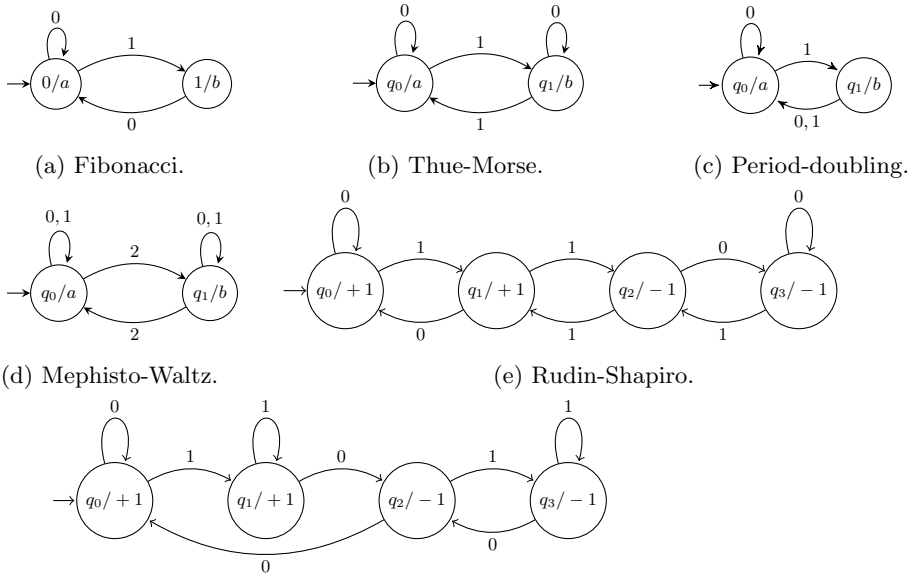
Abstract numeration systems. Such numeration systems were introduced at the beginning of the century by Lecomte and Rigo [13]; see also [3, Chap. 3] for a general presentation. An abstract numeration system (ANS) is defined by a triple $S = (L, \Sigma, <)$ where Σ is an alphabet ordered by the total order $<$ and L is an infinite regular language over Σ, i.e., accepted by a deterministic finite automaton. We say that L is the *numeration language* of S. When we genealogically order the words of L, we obtain a one-to-one correspondence rep_S between $\mathbb{N}$ and L. Then, the *S-representation* of the non-negative integer n is the $(n+1)$st word of L, and the inverse map, called the *(e)valuation map*, is denoted by val_S. For instance, consider the ANS S built on the language a^*b^* over the ordered alphabet $\{a, b : a < b\}$. The first few words in the language are $\varepsilon, a, b, aa, ab, bb, aaa$, and we have $\mathrm{rep}_S(5) = bb$ and $\mathrm{val}_S(aaa) = 6$.

Automatic sequences. As their name indicates it, automatic sequences are defined through automata. A *deterministic finite automaton with output (DFAO)* is defined by a 6-tuple $\mathcal{M} = (Q, \Sigma, \delta, q_0, \Delta, \tau)$, where Q is a finite set of *states*, Σ is a finite *input alphabet*, $\delta\colon Q \times \Sigma \to Q$ is the *transition function*, q_0 is the *initial state*, Δ is a finite *output alphabet*, and $\tau\colon Q \to \Delta$ is the *output function*. The output of M on the finite word $w \in \Sigma^*$, denoted $M(w)$, is defined as $M(w) = \tau(\delta(q_0, w)) \in \Delta$.

Let $S = (L, \Sigma, <)$ be an ANS. An infinite word $\mathbf{x}$ is *S-automatic* if there exists a DFAO $\mathcal{M}$ such that, for all $n \geq 0$, the nth term $\mathbf{x}[n]$ of $\mathbf{x}$ is given by the output $\mathcal{M}(\mathrm{rep}_S(n))$ of $\mathcal{M}$. In this case, we say that the DFAO $\mathcal{M}$ *generates* or *produces* the sequence $\mathbf{x}$. In particular, for an integer $k \geq 2$, if $\mathcal{M}$ is fed with the genealogically ordered language $\{\varepsilon\} \cup \{1, \ldots, k-1\}\{0, \ldots, k-1\}^*$, then $\mathbf{x}$ is said to be *k-automatic*. For the case of integer base numeration systems, a classical reference on automatic sequences is [1], while [16,18] treat the case of more exotic numeration systems. A well-known characterization of automatic sequences states that they are morphic, i.e., obtained as the image under a coding of a fixed point of a morphism [16]. In particular, a sequence is k-automatic if and only if the morphism producing it is k-uniform morphism [1].

3 The Fibonacci word

The infinite Fibonacci word $\mathbf{f} = abaababa\ldots$ is the fixed point, starting with a, of the morphism $\phi\colon a \to ab, b \to a$. It is automatic in a specific ANS based on Fibonacci numbers defined by $F_0 = 1$, $F_1 = 2$, and $F_n = F_{n-1} + F_{n-2}$ for all $n \geq 2$. Zeckendorf [20] demonstrated a remarkable theorem stating that every non-negative integer can be represented as a sum of distinct non-consecutive



(a) Fibonacci. (b) Thue-Morse. (c) Period-doubling.

(d) Mephisto-Waltz. (e) Rudin-Shapiro.

(f) Paper-folding.

Fig. 1: DFAOs generating the automatic sequences of the paper.

Fibonacci numbers. Given an integer n, its *canonical Fibonacci representation*, denoted as $\mathrm{rep}_F(n)$, is defined as $\mathrm{rep}_F(n) = \sum_{i=0}^{m} c_i F_i$ where the coefficients c_i are in $\{0, 1\}$ and obtained using the greedy algorithm. This gives rise to the *Fibonacci* or *Zeckendorf numeration system* and the corresponding *Fibonacci-automatic* sequences. For instance, the automaton in Figure 1a generates $\mathbf{f}$.

Related to our concern, Fici [9] showed that the z-factorization of the Fibonacci word $\mathbf{f}$ is the concatenation of its singular words, i.e.,

$$\mathbf{f} = \prod_{n \geq -1} w_n = a \cdot b \cdot aa \cdot bab \cdot aabaa \cdot babaabab \cdots . \tag{1}$$

The *singular words* of the Fibonacci word $\mathbf{f}$, introduced by Wen and Wen [19], are defined as follows: $w_{-1} = a$, $w_0 = b$, and for $n \geq 1$, $w_n = x \cdot \phi^n(a) \cdot y^{-1}$, where $xy \in \{ab, ba\}$ is the length-2 suffix of $\phi^n(a)$. The first few singular words are a, b, aa, bab, $aabaa$, $babaabab$, $aabaababaabaa$. Notably, $|w_n| = |\phi^n(a)| = F_n$ for all $n \geq 0$.

The idea behind our approach is that we can express the z-factorization of $\mathbf{f}$ as a first-order logic formula in Walnut. We define the two following predicates:

```
def fibfactoreq "?msd_fib At t<n => F[i+t]=F[j+t]":
def fibzfactor "?msd_fib (Aj j<i => ~$fibfactoreq(i,j,n))
    & (At t<n => (El l<i => $fibfactoreq(i,l,t)))":
```

The first formula takes the triple (i, j, n) as input and checks whether the length-n factors $\mathbf{f}[i..i+n-1]$ and $\mathbf{f}[j..j+n-1]$ are equal. The second formula, on input

(i, n), verifies whether the factor $\mathbf{f}[i..i+n-1]$ does not appear before position i and each of its prefixes appears before. In other words, it tests whether $\mathbf{f}[i..i+n-1]$ is the shortest prefix of $(\mathbf{f}[0...i-1])^{-1}\mathbf{f}$ occurring only once in $\mathbf{f}[0...i+n-1]$. In particular, the variables i, j indicate positions within $\mathbf{f}$ and n is a measure of length. Running `Walnut` on these predicates yields the automaton in Figure 2. Now observe from Identity (1) that the pairs (i, n) of position and length of the

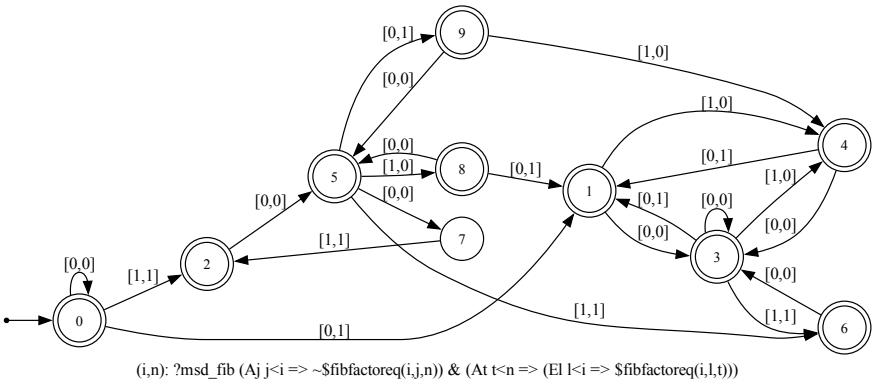


Fig. 2: An automaton accepting, among others, the base-2 representations of the pairs (i, n) giving the position and length of factors of the z-factorization of the Fibonacci word.

factors of the z-factorization of $\mathbf{f}$ are given by $(0, 1)$ and $(F_{n+1} - 1, F_n)$ for all $n \geq 0$. This gives rise to the regular expression in `Walnut`:

```
reg fibzgoodrep msd_fib msd_fib "[0,0]*[0,1] | [0,0]*[1,1]
    | [0,0]*[1,1][0,0]([1,0][0,0])*
    | [0,0]*[1,1][0,0]([1,0][0,0])*[1,0]":
```

Now we check that the pairs (i, n) guessed before indeed give factors having the desired property `fibzfactor`, so we write the following check in `Walnut`:

```
eval fibzcheck "?msd_fib Ai An $fibzgoodrep(i,n) => $fibzfactor(i,n) ":
```

and `Walnut` returns `TRUE`. Note that previous check is not a bi-implication. Indeed, for instance, we see from Figure 2 that other pairs (i, n) satisfy `fibzfactor`. This is for instance the case of $(3, 4)$ since $\mathbf{f}[3..6] = abab$ is the shortest prefix of $(\mathbf{f}[0...2])^{-1}\mathbf{f} = ababaabaababaaba\cdots$ that occurs only once in $\mathbf{f}[0...6] = abaabab$. Therefore, to obtain the z-factorization of $\mathbf{f}$, we need the final check `fibcheck` to be true, but also to have consecutive positions i that cover all $\mathbb{N}$.

Fici [9] showed the following other factorization of the Fibonacci word:

$$\mathbf{f} = \prod_{n \geq 1} \widetilde{\phi^n(a)} = a \cdot ba \cdot aba \cdot baaba \cdot ababaaba \cdots . \tag{2}$$

In fact, the latter is almost the c-factorization of $\mathbf{f}$, with the difference that the c-factorization starts with a, b, a and then coincides with Identity (2). See [2]. It is not difficult to modify the formula `fibzfactor` in `Walnut` to deal with the c-factorization instead:

```
def fibcfactor "?msd_fib (Ej j<i => $fibfactoreq(i,j,n))
        & (Al l<i => ~$fibfactoreq(i,l,n+1))":
```

As before, examining Identity (2), the pairs (i, n) for the factors of the c-factorization are given by the following regular expression in `Walnut`:

```
reg fibcgoodrep msd_fib msd_fib "[0,0]*[0,1] | [0,0]*[1,1]
    | [0,0]*[1,0][0,1] | [0,0]*[1,1][0,0][0,0]
    | [0,0]*[1,1][0,0][0,0][1,0]
    | [0,0]*[1,1][0,0]([1,0][0,0])*[1,0][0,0][0,0]
    | [0,0]*[1,1][0,0]([1,0][0,0])*[1,0][0,0][0,0][1,0]":
```

The last check

```
eval fibccheck "?msd_fib Ai An $fibcgoodrep(i,n) => $fibcfactor(i,n)":
```

returns `TRUE`.

Remark 1. The palindromic (resp., closed) version of the z- and c-factorizations, defined in [11] (resp., [12]), requires that each factor is palindromic (resp., closed, i.e., each factor has a proper factor that occurs exactly twice, as a prefix and as a suffix). One can tweak the `Walnut` code presented in this section to obtain these. See [17, Sec. 8.6.3 and 8.8.3] for related `Walnut` code.

4 The z- and c- factorizations of some classical automatic sequences

The conclusion of the previous section is the following: when a candidate is known for the z- or c-factorization of an infinite word, then it is not difficult to check with `Walnut` that it is indeed the right factorization. This is the purpose of the current section, and we use the same techniques as in Section 3. Given an infinite word $\mathbf{x}$, the `Walnut` code for its z-factorization (resp., c-factorization) is summed up in Code 1 (resp., Code 2). Also see Table 1 where we give the first few factors of $z(\mathbf{x})$ and $c(\mathbf{x})$ for some words $\mathbf{x}$.

Code 1. Given the k-automatic sequence $\mathbf{x}$ coded by `X` in `Walnut`, we use the following predicates to find its z-factorization, where `LX` is some specific guessed regular expression:

$\mathbf{x}$	$z(\mathbf{x}) = (z_0, z_1, \ldots)$
f	$(a, b, aa, bab, aabaa, babaabab, aabaababaabaa, babaababaabaababaabab, \ldots)$
t	$(a, b, ba, baa, bbaa, babb, abaaba, bbaabb, abaabbaababbaa, bbabaababbab, \ldots)$
pd	$(a, b, aa, abab, abaaabaa, abaaababababaaabab, abaaababababaaabaaabaaabababaaabaa, \ldots)$
rs	$(1, 11(-1), 11(-1)111, 1(-1)(-1), (-1)1(-1), 111(-1)11(-1)1(-1), (-1)(-1)11, \ldots)$
pf	$(1, 1(-1), 11(-1)(-1), 111, (-1)(-1)1(-1), (-1)111(-1)1, 1(-1)(-1)(-1), \ldots)$
mw	$(a, ab, aabb, baaa, baabbbab, babbaa, abbbaaabaabbbabbabbaaabb, \ldots)$
$\mathbf{x}$	$c(\mathbf{x}) = (c_0, c_1, \ldots)$
f	$(a, b, a, aba, baaba, ababaaba, baabaababaaba, ababaababaabaababaaba, \ldots)$
t	$(a, b, b, ab, a, abba, aba, bbabaab, abbaab, babaabbaababba, abbabaababba, \ldots)$
pd	$(a, b, a, aa, ba, baba, aaba, aabaaaba, babaaaba, babaaababababaaaba, \ldots)$
rs	$(1, 11, (-1), 11(-1)11, 11(-1), (-1)(-1), 1(-1)111, (-1)11(-1)1, (-1)(-1)(-1)1, \ldots)$
pf	$(1, 1, (-1), 11(-1), (-1)11, 1(-1)(-1)1, (-1)(-1)111(-1), 11(-1)(-1), \ldots)$
mw	$(a, a, b, aab, bb, aa, abaabbba, bbabba, aabaab, aabbbaaabaabbbabbabbaaab, \ldots)$

Table 1: The first few factors of the z- and c-factorizations of the automatic sequences considered in this paper.

```
def xfactoreq "?msd_k At t<n => X[i+t]=X[j+t]":
def xzfactor "?msd_k (Aj j<i => ~$xfactoreq(i,j,n)) &
    (At t<n => (El l<i => $xfactoreq(i,l,t)))":
reg xzgoodrep msd_k msd_k "LX":
eval xzcheck "?msd_k Ai An $xzgoodrep(i,n) => $xzfactor(i,n)":
```

Code 2. Given the k-automatic sequence $\mathbf{x}$ coded by `X` in `Walnut`, we use the following predicates to find its c-factorization, where `LX` is some specific guessed regular expression:

```
def xfactoreq "?msd_k At t<n => X[i+t]=X[j+t]":
def xcfactor "?msd_k (Ej j<i => $xfactoreq(i,j,n))
    & (Al l<i => ~$xfactoreq(i,l,n+1))":
reg xcgoodcrep msd_k msd_k "LX":
eval xccheck "?msd_k Ai An $xcgoodrep(i,n) => $xcfactor(i,n)":
```

4.1 The Thue-Morse sequence

The most famous example among 2-automatic sequences is the *Thue-Morse* sequence $\mathbf{t}$ which is the fixed point of the morphism $\mu\colon a \mapsto ab, b \mapsto ba$ starting with a. This sequence is generated by the automaton in Figure 1b.

Theorem 1. *Let $z(\mathbf{t}) = (z_0, z_1, \ldots)$ be the z-factorization of the Thue-Morse sequence $\mathbf{t}$. Then, for all $m \in \{0, \ldots, 6\}$, z_m is given in Table 1 and, for all $m \geq 7$, $z_m = \mathbf{t}[i..i+n-1]$ where*

$$(i, n) = \begin{cases} (13 \cdot 2^{m/2-3} + 1, 7 \cdot 2^{m/2-3}), & \text{if } m \text{ is even;} \\ (5 \cdot 2^{(m-1)/2-1} + 1, 3 \cdot 2^{(m-1)/2-2}), & \text{if } m \text{ is odd.} \end{cases}$$

Proof. In Code 1, replace X by T, k by 2, and LX by

```
[0,0]*[0,1] | [0,0]*[1,1] | [0,0]*[1,0][1,0]
| [0,0]*[1,0][0,1][0,1] | [0,0]*[1,0][1,0][1,0]
| [0,0]*[1,0][0,1][1,0][1,0] | [0,0]*[1,0][1,0][1,1][1,0]
| [0,0]*[1,0][1,1][0,1][1,1][0,0]*[1,0]
| [0,0]*[1,0][0,0][1,1][0,1][0,0]*[1,0]
```

Then running Code 1 in Walnut returns TRUE. □

The next result gives back [2, Thm. 2].

Theorem 2. *Let $c(\mathbf{t}) = (c_0, c_1, \ldots)$ be the c-factorization of the Thue-Morse sequence $\mathbf{t}$. Then, for all $m \in \{0, \ldots, 5\}$, c_m is given in Table 1 and, for all $m \geq 6$, $c_m = \mathbf{t}[i..i+n-1]$ where*

$$(i, n) = \begin{cases} (5 \cdot 2^{m/2-2}, 3 \cdot 2^{m/2-3}), & \text{if } m \text{ is even;} \\ (13 \cdot 2^{(m-1)/2-3}, 7 \cdot 2^{(m-1)/2-3}), & \text{if } m \text{ is odd.} \end{cases}$$

Proof. In Code 2, replace X by T, k by 2, and LX by

```
[0,0]*[0,1] | [0,0]*[1,1] | [0,0]*[1,0][0,1] | [0,0]*[1,1][1,0]
| [0,0]*[1,0][0,0][1,1] | [0,0]*[1,1][1,0][0,0]
| [0,0]*[1,0][0,0][1,1][0,1][0,0]* | [0,0]*[1,0][1,1][0,1][1,1][0,0]*
```

Then running Code 2 in Walnut returns TRUE. □

4.2 The period-doubling sequence

The *period-doubling* sequence $\mathbf{pd} = abaaababababaaabaa\cdots$ is also a 2-automatic sequence. It is closely related to the Thue-Morse sequence as it is defined for all $n \geq 0$ by $\mathbf{pd}[n] = b$ if $\mathbf{t}[n] = \mathbf{t}[n+1]$, $\mathbf{pd}[n] = a$ otherwise. Furthermore, $\mathbf{pd}$ is the fixed point of the morphism $h\colon a \mapsto ab, b \mapsto aa$, starting with a and is generated by the automaton in Figure 1c.

Theorem 3. *Let $z(\mathbf{pd}) = (z_0, z_1, \ldots)$ be the z-factorization of the period-doubling sequence $\mathbf{pd}$. Then, $z_0 = a$ and, for all $m \geq 1$, $z_m = \mathbf{pd}[i..i+n-1]$ where $i = n = 2^{m-1}$.*

Proof. In Code 1, replace X by PD, k by 2, and LX by

```
[0,0]*[0,1] | [0,0]*[1,1][0,0]*
```

Then running Code 1 in Walnut returns TRUE. □

The next result gives back [2, Thm. 4].

Theorem 4. *Let $c(\mathbf{pd}) = (c_0, c_1, \ldots)$ be the c-factorization of the period-doubling sequence $\mathbf{pd}$. Then $c_0 = a$ and, for all $m \geq 1$, $c_m = \mathbf{pd}[i..i+n-1]$ where*

$$(i, n) = \begin{cases} (3 \cdot 2^{m/2-1} - 1, 2^{m/2-1}), & \text{if } m \text{ is even;} \\ (2^{(m-1)/2+1} - 1, 2^{(m-1)/2}), & \text{if } m \text{ is odd.} \end{cases}$$

Proof. In Code 2, replace `X` by `PD`, `k` by 2, and `LX` by

```
[0,0]*[0,1] | [0,0]*[1,1][1,0]* | [0,0]*[1,1][0,0][1,0]*
```

Then running Code 2 in `Walnut` returns `TRUE`. □

4.3 The Rudin-Shapiro sequence

The *Rudin-Shapiro* sequence $\mathbf{rs} = 111(-1)11(-1)1\cdots$ is defined as follows: for all $n \geq 0$, the nth letter $\mathbf{rs}[n]$ is given by 1 or -1 according to the parity of the number of (possibly overlapping) occurrences of the block 11 is the base-2 representation of n. The sequence is 2-automatic as the automaton in Figure 1e generates it. In addition, $\mathbf{rs}$ can be written as $\tau(\rho^\omega(a))$, where $\rho\colon a \mapsto ab, b \mapsto ac, c \mapsto db, d \mapsto dc$ and $\tau\colon a, b \mapsto 1, c, d \mapsto -1$.

Theorem 5. *Let $z(\mathbf{rs}) = (z_0, z_1, \ldots)$ be the z-factorization of the Rudin-Shapiro sequence $\mathbf{rs}$. Then $z_m = \mathbf{rs}[i..i+n-1]$ where, for $0 \leq m \leq 10$, (i, n) belongs to*

$$\{(0,1),(1,3),(4,6),(10,3),(13,3),(16,9),(25,4),(29,8),(37,12),(49,6),(55,6)\},$$

and for all $m \geq 11$ with $p = \lfloor \frac{m}{4} \rfloor$,

$$(i,n) = \begin{cases} (9\cdot 2^p + 1, 3\cdot 2^p), & \text{if } m \equiv 0 \bmod 4; \\ (3\cdot 2^{p+2} + 1, 2^p), & \text{if } m \equiv 1 \bmod 4; \\ (13\cdot 2^p + 1, 2^{p+1}), & \text{if } m \equiv 2 \bmod 4; \\ (15\cdot 2^p + 1, 3\cdot 2^p), & \text{if } m \equiv 3 \bmod 4. \end{cases}$$

Proof. In Code 1, replace `X` by `RS`, `k` by 2, and `LX` by

```
[0,0]*[0,1] | [0,0]*[1,0][1,1] | [0,0]*[1,1][0,1][0,0]
| [0,0]*[1,0][0,0][1,1][0,1] | [0,0]*[1,0][1,0][1,1][1,1]
| [0,0]*[1,0][0,1][0,0][0,0][0,1] | [0,0]*[1,0][1,0][0,1][0,0][1,0]
| [0,0]*[1,0][1,1][1,0][0,0][1,0] | [0,0]*[1,0][0,0][0,1][1,1][0,0][1,0]
| [0,0]*[1,0][1,0][0,0][0,1][0,1][1,0]
| [0,0]*[1,0][1,0][0,0][1,1][1,1][1,0]
| [0,0]*[1,0][1,0][1,1][1,1][0,0][0,0]*[1,0]
| [0,0]*[1,0][0,0][0,1][1,1][0,0][0,0][0,0]*[1,0]
| [0,0]*[1,0][1,0][0,0][0,1][0,0][0,0][0,0]*[1,0]
| [0,0]*[1,0][1,0][0,1][1,0][0,0][0,0][0,0]*[1,0]
```

Then running Code 1 in `Walnut` returns `TRUE`. □

Theorem 6. *Let $c(\mathbf{rs}) = (c_0, c_1, \ldots)$ be the c-factorization of the Rudin-Shapiro sequence $\mathbf{rs}$. Then $c_m = \mathbf{rs}[i..i+n-1]$ where, for $0 \leq m \leq 12$, (i, n) belongs to*

$$\{(0,1),(1,2),(3,1),(4,5),(9,3),(12,2),(14,5),(19,5),(24,4),(28,8),(36,12),(48,6),(54,6)\},$$

and for all $m \geq 13$ *with* $p = \lfloor \frac{m}{4} \rfloor$,

$$(i,n) = \begin{cases} (13 \cdot 2^{p-1}, 2^p), & \text{if } m \equiv 0 \bmod 4; \\ (15 \cdot 2^{p-1}, 3 \cdot 2^{p-1}), & \text{if } m \equiv 1 \bmod 4; \\ (9 \cdot 2^p, 3 \cdot 2^p), & \text{if } m \equiv 2 \bmod 4; \\ (12 \cdot 2^p, 2^p), & \text{if } m \equiv 3 \bmod 4. \end{cases}$$

Proof. In Code 2, replace `X` by `RS`, `k` by 2, and `LX` by

```
[0,0]*[0,1] | [0,0]*[0,1][1,0] | [0,0]*[1,0][1,1]
| [0,0]*[1,1][0,0][0,1] | [0,0]*[1,0][0,0][0,1][1,1]
| [0,0]*[1,0][1,0][0,1][0,0] | [0,0]*[1,0][1,1][1,0][0,1]
| [0,0]*[1,0][0,0][0,1][1,0][1,1] | [0,0]*[1,0][1,0][0,1][0,0][0,0]
| [0,0]*[1,0][1,1][1,0][0,0][0,0]
| [0,0]*[1,0][0,0][0,1][1,1][0,0][0,0]
| [0,0]*[1,0][1,0][0,0][0,1][0,1][0,0]
| [0,0]*[1,0][1,0][0,0][1,1][1,1][0,0]
| [0,0]*[1,0][1,0][0,0][0,1][0,0][0,0][0,0]*[1,0]
| [0,0]*[1,0][1,0][0,1][1,0][0,0][0,0][0,0]*[1,0]
| [0,0]*[1,0][1,0][1,1][1,1][0,0][0,0][0,0]*[1,0]
| [0,0]*[1,0][0,0][0,1][1,1][0,0][0,0][0,0][0,0]*[1,0]
```

Then running Code 2 in `Walnut` returns `TRUE`. □

4.4 The paper-folding sequence

The *paper-folding* sequence arises from the iterative folding of a piece of paper. As the paper is folded repeatedly to the right and then unfolded, the sequence of turns is recorded. For each folding action, a corresponding binary digit is assigned: right turns are coded by 1 and left turns by -1. This systematic recording process generates the infinite sequence

$$\mathbf{pf} = 11(-1)11(-1)(-1)111(-1)(-1)1(-1)(-1)111(-1)11\cdots .$$

It is 2-automatic and generated by the automaton in Figure 1f. Finally, it can be written as $\nu(h^\omega(a))$, where $h\colon a \mapsto ab, b \mapsto cb, c \mapsto ad, d \mapsto cd$ and $\nu\colon a, b \mapsto 1, c, d \mapsto -1$.

Theorem 7. *Let* $z(\mathbf{pf}) = (z_0, z_1, \ldots)$ *be the z-factorization of the paper-folding sequence* $\mathbf{pf}$*. Then, for all* $m \in \{0, \ldots, 5\}$*,* z_m *is given in Table 1 and, for all* $m \geq 6$*,* $z_m = \mathbf{pf}[i..i+n-1]$ *where*

$$(i,n) = \begin{cases} (5 \cdot 2^{m/2-1}, 2^{m/2-1}), & \text{if } m \text{ is even;} \\ (3 \cdot 2^{(m-1)/2}, 2^{(m-1)/2+1}), & \text{if } m \text{ is odd.} \end{cases}$$

Proof. In Code 1, replace `X` by `RS`, `k` by 2, and `LX` by

```
[0,0]*[0,1] | [0,0]*[0,1][1,0] | [0,0]*[0,1][1,0][1,0]
| [0,0]*[1,0][1,1][1,1] | [0,0]*[1,0][0,1][1,0][0,0]
| [0,0]*[1,0][1,1][1,1][0,0] | [0,0]*[1,0][0,0][1,1][0,0][0,0]*[0,0]
| [0,0]*[1,1][1,0][0,0][0,0][0,0]*[0,0]
```

Then running Code 1 in `Walnut` returns `TRUE`. □

Theorem 8. *Let $c(\mathbf{pf}) = (c_0, c_1, \ldots)$ be the c-factorization of the paper-folding sequence $\mathbf{pf}$. Then $c_m = \mathbf{pf}[i..i+n-1]$ where, for $0 \leq m \leq 9$, (i, n) belongs to*

$$\{(0,1),(1,1),(2,1),(3,3),(6,3),(9,4),(13,6),(19,4),(23,6),(29,10)\},$$

and for all $m \geq 10$ with $p = \lfloor \frac{m}{3} \rfloor$,

$$(i,n) = \begin{cases} (13 \cdot 2^{p-2} - 1, 7 \cdot 2^{p-2}), & \text{if } m \equiv 0 \bmod 3; \\ (5 \cdot 2^p - 1, 2^p), & \text{if } m \equiv 1 \bmod 3; \\ (3 \cdot 2^{p+1} - 1, 2^{p-1}), & \text{if } m \equiv 2 \bmod 3. \end{cases}$$

Proof. In Code 2, replace `X` by `PF`, `k` by 2, and `LX` by

```
[0,0]*[0,1] | [0,0]*[1,1] | [0,0]*[1,0][0,1] | [0,0]*[1,1][1,1]
| [0,0]*[1,0][1,1][0,1] | [0,0]*[1,0][0,1][0,0][1,0]
| [0,0]*[1,0][1,1][0,1][1,0] | [0,0]*[1,0][0,0][0,1][1,0][1,0]
| [0,0]*[1,0][0,0][1,1][1,1][1,0] | [0,0]*[1,0][1,1][1,0][0,1][1,0]
| [0,0]*[1,0][0,0][0,1][1,0][1,0][1,0][1,0]*
| [0,0]*[1,0][0,0][1,0][1,1][1,0][1,0][1,0]*
| [0,0]*[1,0][1,1][0,1][0,1][1,0][1,0][1,0]*
```

Then running Code 2 in `Walnut` returns `TRUE`. □

4.5 The Mephisto-Waltz sequence

The *Mephisto-Waltz* sequence $\mathbf{mw} = aabaabbba\cdots$ is defined as the fixed point of the morphism $a \mapsto aab$, $b \mapsto bba$ starting with a. It is thus 3-automatic and is generated by the automaton in Figure 1d. Another definition of this sequence is, for all $n \geq 0$, $\mathbf{mw}[n] = a$ if $|\operatorname{rep}_3(n)|_2$ is even, $\mathbf{mw}[n] = b$ otherwise, i.e., we store the parity of the number of 2's in the base-3 representation of n.

Theorem 9. *Let $z(\mathbf{mw}) = (z_0, z_1, \ldots)$ be the z-factorization of the Mephisto-Waltz sequence $\mathbf{mw}$. Then, for all $m \in \{0, \ldots, 3\}$, z_m is given in Table 1 and, for all $m \geq 4$, $z_m = \mathbf{mw}[i..i+n-1]$ where, for $p = \lfloor \frac{m}{3} \rfloor$,*

$$(i,n) = \begin{cases} (8 \cdot 3^{p-1} + 1, 2 \cdot 3^{p-1}), & \text{if } m \equiv 0 \bmod 3; \\ (10 \cdot 3^{p-1} + 1, 8 \cdot 3^{p-1}), & \text{if } m \equiv 1 \bmod 3; \\ (2 \cdot 3^{p+1} + 1, 2 \cdot 3^p), & \text{if } m \equiv 2 \bmod 3. \end{cases}$$

Proof. First, in `Walnut`, code the Mephisto-Waltz sequence with the commands `morphism h "0->001 1->110":` and `promote MW h`. Then, in Code 1, replace `X` by `MW`, `k` by 3, and `LX` by

```
[0,0]*[0,1] | [0,0]*[1,2] | [0,0]*[1,1][0,1] | [0,0]*[2,1][1,1]
| [0,0]*[1,0][0,2][2,2] | [0,0]*[2,0][2,2][0,0]*[1,0]
| [0,0]*[1,0][0,2][1,2][0,0]*[1,0] | [0,0]*[2,0][0,2][0,0]*[1,0]":
```

Then running Code 1 in Walnut returns TRUE. □

Theorem 10. *Let $c(\mathbf{mw}) = (c_0, c_1, \ldots)$ be the c-factorization of the Mephisto-Waltz sequence* **mw**. *Then, for all $m \in \{0, \ldots, 3\}$, c_m is given in Table 1 and, for all $m \geq 4$, $c_m = \mathbf{mw}[i..i+n-1]$ where, for $p = \lfloor \frac{m}{3} \rfloor$,*

$$(i,n) = \begin{cases} (10 \cdot 3^{p-2}, 8 \cdot 3^{p-2}), & \text{if } m \equiv 0 \bmod 3; \\ (2 \cdot 3^{p}, 2 \cdot 3^{p-1}), & \text{if } m \equiv 1 \bmod 3; \\ (8 \cdot 3^{p-1}, 2 \cdot 3^{p-1}), & \text{if } m \equiv 2 \bmod 3. \end{cases}$$

Proof. In Code 2, replace X by MW coded in Walnut as in the proof of Theorem 9, k by 3, and LX by

```
[0,0]*[0,1] | [0,0]*[1,1] | [0,0]*[2,1] | [0,0]*[1,0][1,0]
| [0,0]*[1,0][0,2][1,2][0,0]* | [0,0]*[2,0][0,2][0,0]*
| [0,0]*[2,0][2,2][0,0]*":
```

Then running Code 2 in Walnut returns TRUE. □

5 Conclusion

In this paper, we investigated the following problem: given an abstract numeration system S and an S-automatic sequence $\mathbf{x}$, is it possible to use Walnut to obtain a description of both the Crochemore and Ziv-Lempel factorizations of $\mathbf{x}$ that only depend on the numeration system S? We produced a detailed code for several classical automatic sequences in the Zeckendorff system as well as in bases 2 and 3. According to us, a general answer to the previous question is far from being obvious to obtain. Indeed, first, the software Walnut only works in the case of so-called *addable* abstract numeration systems, i.e., when addition can be performed by an automaton. Then, as said previously, a candidate for the factorizations has to be known in advance in the hope of using Walnut. We believe that finding such candidates might be tricky for a general automatic sequence, when not much information is known about the inner structure of the sequence. Observe also that, already among the 2-automatic sequences we considered, the pairs of positions and lengths of the factors of the factorizations strongly depend on the sequence itself and not only on the underlying numeration system. Finally, we wish to point that we examined non purely morphic sequences for which Berstel and Savelli write in [2, Sec. 6] that "it is not yet clear whether a satisfactory description [of the c-factorization] can be obtained".

Acknowledgments

We thank Narad Rampersad for useful discussions.

Manon Stipulanti is an FNRS Research Associate supported by the Research grant 1.C.104.24F.

References

1. Allouche, J.P., Shallit, J.: Automatic sequences: theory, applications, generalizations. Cambridge University Press, Cambridge (2003). https://doi.org/10.1017/CBO9780511546563
2. Berstel, J., Savelli, A.: Crochemore factorization of Sturmian and other infinite words. In: Mathematical Foundations of Computer Science (MFCS) 2006, Lecture Notes in Comput. Sci., vol. 4162, pp. 157–166. Springer, Berlin (2006). https://doi.org/10.1007/11821069_14
3. Berthé, V., Rigo, M. (eds.): Combinatorics, Automata, and Number Theory, Encyclopedia of Mathematics and its Applications, vol. 135. Cambridge University Press (2010). https://doi.org/10.1017/CBO9780511777653
4. Brillhart, J., Morton, P.: A case study in mathematical research: the Golay-Rudin-Shapiro sequence. Mathematical Intelligencer **13**(1), 36–48 (1991)
5. Constantinescu, S., Ilie, L.: The Lempel–Ziv complexity of fixed points of morphisms. SIAM Journal on Discrete Mathematics **21**(2), 466–481 (2007). https://doi.org/10.1137/050646846
6. Crochemore, M.: Recherche linéaire d'un carré dans un mot. C. R. Acad. Sci. Paris Sér. I Math. **296**(18), 781–784 (1983)
7. Crochemore, M., Rytter, W.: Text algorithms. The Clarendon Press, Oxford University Press, New York (1994)
8. Devaney, R.L.: An introduction to chaotic dynamical systems. CRC Press, Boca Raton, FL, third edn. (2022)
9. Fici, G.: Factorizations of the Fibonacci infinite word. J. Integer Seq. **18**(9), Article 15.9.3, 14 (2015)
10. Ghareghani, N., Mohammad-Noori, M., Sharifani, P.: On z-factorization and c-factorization of standard episturmian words. Theoret. Comput. Sci. **412**(39), 5232–5238 (2011). https://doi.org/10.1016/j.tcs.2011.05.035
11. Jahannia, M., Mohammad-Noori, M., Rampersad, N., Stipulanti, M.: Palindromic Ziv-Lempel and Crochemore factorizations of m-bonacci infinite words. Theoret. Comput. Sci. **790**, 16–40 (2019). https://doi.org/10.1016/j.tcs.2019.05.010
12. Jahannia, M., Mohammad-Noori, M., Rampersad, N., Stipulanti, M.: Closed Ziv-Lempel factorization of the m-bonacci words. Theoret. Comput. Sci. **918**, 32–47 (2022). https://doi.org/10.1016/j.tcs.2022.03.019
13. Lecomte, P.B.A., Rigo, M.: Numeration systems on a regular language. Theory Comput. Syst. **34**(1), 27–44 (2001). https://doi.org/10.1007/s002240010014
14. Lempel, A., Ziv, J.: On the complexity of finite sequences. IEEE transactions on information theory **22**(1), 75–81 (1976)
15. Mousavi, H.: Automatic theorem proving in `Walnut` (2021), preprint available at https://arxiv.org/abs/1603.06017
16. Rigo, M., Maes, A.: More on generalized automatic sequences. Journal of Automata, Languages, and Combinatorics **7**(3), 351–376 (2002). https://doi.org/10.25596/jalc-2002-351
17. Shallit, J.: The logical approach to automatic sequences: exploring combinatorics on words with `Walnut`. London Mathematical Society Lecture Note Series, Cambridge University Press (2022). https://doi.org/10.1017/9781108775267
18. Shallit, J.: A generalization of automatic sequences. Theoret. Comput. Sci. **61**(1), 1–16 (1988). https://doi.org/10.1016/0304-3975(88)90103-X
19. Wen, Z.X., Wen, Z.Y.: Some properties of the singular words of the Fibonacci word. European J. Combin. **15**(6), 587–598 (1994). https://doi.org/10.1006/eujc.1994.1060

20. Zeckendorf, E.: Représentation des nombres naturels par une somme de nombres de Fibonacci ou de nombres de Lucas. Bull. Soc. Roy. Sci. Liège **41**, 179–182 (1972)
21. Ziv, J., Lempel, A.: A universal algorithm for sequential data compression. IEEE Trans. Inform. Theory **IT-23**(3), 337–343 (1977). https://doi.org/10.1109/tit.1977.1055714