ADDITIVE WORD COMPLEXITY AND `WALNUT`

PIERRE POPOLI, JEFFREY SHALLIIT, AND MANON STIPULANTI

Abstract. In combinatorics on words, a classical topic of study is the number of specific patterns appearing in infinite sequences. For instance, many works have been dedicated to studying the so-called factor complexity of infinite sequences, which gives the number of different factors (contiguous subblocks of their symbols), as well as abelian complexity, which counts factors up to a permutation of letters. In this paper, we consider the relatively unexplored concept of additive complexity, which counts the number of factors up to additive equivalence. We say that two words are additively equivalent if they have the same length and the total weight of their letters is equal. Our contribution is to expand the general knowledge of additive complexity from a theoretical point of view and consider various famous examples. We show a particular case of an analog of the long-standing conjecture on the regularity of the abelian complexity of an automatic sequence. In particular, we use the formalism of logic, and the software `Walnut`, to decide related properties of automatic sequences. We compare the behaviors of additive and abelian complexities, and we also consider the notion of abelian and additive powers. Along the way, we present some open questions and conjectures for future work.

2010 *Mathematics Subject Classification*: 68R15
Keywords: Combinatorics on words, Abelian complexity, Additive complexity, Automatic sequences, Walnut software

1. Introduction

Combinatorics on words is the study of finite and infinite sequences, also known as *streams* or *strings* in other theoretical contexts. Although it is rooted in the work of Axel Thue, who was the first to study regularities in infinite words in the early 1900's, words became a systematic topic of combinatorial study in the second half of the 20th century [8]. Since then, many different approaches have been developed to analyze words from various points of view. One of them is the celebrated *factor* or *subword complexity function*: given an infinite word $\mathbf{x}$ and a length $n \geq 0$, we compute the size of $\mathcal{L}_n(\mathbf{x})$, which contains all length-n contiguous subblocks of $\mathbf{x}$, also called *factors* or *subwords* in the literature. One of the most famous theorems in combinatorics on words related to the factor complexity function is due to Morse and Hedlund in 1940 [27], where they obtained a characterization of ultimately periodic words. As a consequence of this result, combinatorists defined binary aperiodic infinite words having the smallest possible factor complexity function, the so-called *Sturmian words*.

Many other complexity functions have been defined on words depending on the properties combinatorists wanted to emphasize; see, for instance, the non-exhaustive list in the introduction of [1]. As the literature on the topic is quite large, we only cite the so-called *abelian complexity function*. Instead of counting all distinct factors, we count them up to abelian equivalence: two words u and v are *abelian equivalent*, written $u \sim_{\text{ab}} v$, if they are permutations of each other. For instance, in English, `own`, `now`, and `won` are all abelian equivalent. For an infinite word $\mathbf{x}$, we let $\rho^{\text{ab}}_{\mathbf{x}}$ denote its abelian complexity function. In this paper, we study yet another equivalence relation on words; namely, *additive equivalence*. Roughly, two words are additively equivalent if the total weight of their letters is equal. In the following, for a word $w \in \Sigma^*$, we let $|w|$ denote its *length*, i.e., the number of letters it is composed of. Furthermore, for each letter $a \in \Sigma$, we let $|w|_a$ denote the number of a's in w.

Definition 1. Fix an integer $\ell \geq 1$ and the alphabet $\Sigma = \{0, 1, \ldots, \ell\}$. Two words $u, v \in \Sigma^*$ are *additively equivalent* if $|u| = |v|$ and $\sum_{i=0}^{\ell} i|u|_i = \sum_{i=0}^{\ell} i|v|_i$, which we write as $u \sim_{\text{add}} v$.

Example 2. Over the three-letter alphabet $\{0, 1, 2\}$, we have $020 \sim_{\text{add}} 101$.

Definition 3. Fix an integer $\ell \geq 1$ and the alphabet $\Sigma = \{0, 1, \ldots, \ell\}$. Let $\mathbf{x}$ be an infinite word on Σ. The *additive complexity* of $\mathbf{x}$ is the function $\rho^{\text{add}}_{\mathbf{x}} \colon \mathbb{N} \to \mathbb{N}, n \mapsto \#(\mathcal{L}_n(\mathbf{x})/{\sim_{\text{add}}})$, i.e., length-$n$ factors of $\mathbf{x}$ are counted up to additive equivalence.

Surprisingly, not so many results are known for additive equivalence and the corresponding complexity function, in contrast with the abundance of abelian results in combinatorics; see [32, 33, 39, 43, 44], for example. Notice that over a two-letter alphabet, the concepts of abelian and additive complexity coincide. Additive complexity was first introduced in [5], where the main result states that bounded additive complexity implies that the underlying infinite word contains an additive k-power for every k (an *additive k-power* is a word w that can be written as $x_1 x_2 \cdots x_k$ where the words $x_1, x_2, \ldots, x_k$ are all additively equivalent).

Later, words with bounded additive complexity were studied: first in [5], and with more attention in [6]. In particular, equivalent properties of bounded additive complexity were found. We also mention work on the particular case of constant additive and abelian complexities [6, 20, 33, 37]. As already observed, combinatorists expanded the notion of pattern avoidance to additive powers. See the most recent preprint [4] for a nice exposition of the history. For instance, Cassaigne et al. [14] proved that the fixed point of the morphism $0 \mapsto 03, 1 \mapsto 43, 3 \mapsto 1, 4 \mapsto 01$ avoids additive cubes (see Section 2 for concepts not defined in this introduction). Rao [31] proved that it is possible to avoid additive cubes over a ternary alphabet and mentioned that the question about additive squares (in one dimension and over the integers) is still open. Furthermore, we mention the following related papers. Brown and Freedman [11] also talked about the open problem about additive squares. A notion of "close" additive squares is defined in [12]. In [24], the authors showed that in every infinite word over a finite set of non-negative integers there is always a sequence of factors (not necessarily of the same length) having the same sum. In [30], the more general setting of k-power modulo a morphism was studied. Finally, in terms of computing the additive complexity of specific infinite words, to our knowledge, only that of a fixed point of a Thue–Morse-like morphism is known [17].

In this paper, we expand our general knowledge of additive complexity functions of infinite words. After giving some preliminaries in Section 2, we obtain several general results in Section 3, and in Theorem 10 we prove a particular case of the conjecture below. Note that it is itself a particular case of the long-standing similar conjecture in an abelian context: the abelian complexity of a k-automatic sequence is a k-regular sequence [29].

Conjecture 4. *The additive complexity of a k-automatic sequence is a k-regular sequence.*

In particular, the proof of our Theorem 10 relies on the logical approach to combinatorics on words: indeed, many properties of words can be phrased in first-order logic. Based on this, Mousavi [28] designed the free software `Walnut` that allows one to automatically decide the truth of assertions about many properties for a large family of words. See [40] for the formalism of the software and a survey of the combinatorial properties that can be decided. In Section 3, we show how `Walnut` may be used in an ad-hoc way to give partial answers to Conjecture 4. In Section 4, we compare the behaviors of the additive and abelian complexity functions of various words. We highlight the fact that they may behave quite differently, sometimes making use of `Walnut`. Motivated by the various behaviors we observe, we study in Section 5 some words for which the additive and abelian complexity functions are in fact equal. We end the paper by considering the related notions of abelian and additive powers.

2. Preliminaries

For a general reference on words, we guide the reader to [26]. An *alphabet* is a finite set of elements called *letters*. A *word* over an alphabet Σ is a finite or infinite sequence of letters from Σ. The *length* of a finite word w, denoted $|w|$, is the number of letters it is made of. The *empty word* is the only 0-length word, denoted by ε. For all $n \geq 0$, we let Σ^n denote the set of all length-n words over Σ. We let Σ^* denote the set of finite words over A, including the empty word, and equipped with the concatenation. In this paper, we distinguish finite and infinite words by writing the latter in bold. For each letter $a \in \Sigma$ and a word $w \in \Sigma^*$, we let $|w|_a$ denote the number of a's in w. Let us assume that the alphabet $\Sigma = \{a_1 < \cdots < a_k\}$ is ordered. For a word $w \in \Sigma^*$, we let $\Psi(w)$ denote the *abelianization* or *Parikh vector* $(|w|_{a_1}, \ldots, |w|_{a_k})$, which counts the number of different letters appearing in w. For example, over the alphabet $\{\texttt{e} < \texttt{l} < \texttt{s} < \texttt{v}\}$, we have $\Psi(\texttt{sleeveless}) = (4, 2, 3, 1)$.

A *factor* of a word is one of its (contiguous) subblocks. For a given word $\mathbf{x}$, for all $n \geq 0$, we let $\mathcal{L}_n(\mathbf{x})$ denote the set of length-n factors of $\mathbf{x}$. A *prefix* (resp., *suffix*) is a starting (resp., ending) factor. A prefix or a suffix is *proper* if it is not equal to the initial word. Infinite words are indexed starting at 0. For such a word $\mathbf{x}$, we let $\mathbf{x}(n)$ denote its nth letter with $n \geq 0$ and, for $0 \leq m \leq n$, we let $\mathbf{x}[m..n]$ denote the factor $\mathbf{x}(m) \cdots \mathbf{x}(n)$.

Let Σ and Γ be finite alphabets. A *morphism* $f \colon \Sigma^* \to \Gamma^*$ is a map satisfying $f(uv) = f(u)f(v)$ for all $u, v \in \Sigma^*$. In particular, $f(\varepsilon) = \varepsilon$, and f is entirely determined by the images of the letters in Σ. For an integer $k \geq 1$, a morphism is *k-uniform* if it maps each letter to a length-k word. A 1-uniform morphism is called a *coding*. A sequence $\mathbf{x}$ is *morphic* if there exist a morphism $f \colon \Sigma^* \to \Sigma^*$, a coding $g \colon \Sigma^* \to \Gamma^*$, and a letter $a \in \Sigma$ such that $\mathbf{x} = g(f^\omega(a))$, where $f^\omega(a) = \lim_{n\to\infty} f^n(a)$. The latter word $f^\omega(a)$ is a *fixed point* of f.

Introduced by Cobham [18] in the early 1970s, automatic words have several equivalent definitions depending on the point of view one wants to adopt. For the case of integer base numeration systems, a comprehensive presentation of automatic sequences is [3], while [34, 38] treat the case of more exotic numeration systems. We start with the definition of positional numeration systems. Let $U = (U(n))_{n\geq 0}$ be an increasing sequence of integers with $U(0) = 1$. A positive integer n can be decomposed, not necessarily uniquely, as $n = \sum_{i=0}^{t} c(i)\, U(i)$ with non-negative integer coefficients $c(i)$. If these coefficients are computed greedily, then for all $j < t$ we have $\sum_{i=0}^{j} c(i)\, U(i) < U(j + 1)$ and $\text{rep}_U(n) = c(t) \cdots c(0)$ is said to be the *(greedy) U-representation* of n. By convention, that of 0 is the empty word ε, and the greedy representation of $n > 0$ starts with a non-zero digit. A sequence U satisfying all the above conditions defines a *positional numeration system*. Let

$U = (U(n))_{n\geq 0}$ be such a numeration system. A sequence $\mathbf{x}$ is *U-automatic* if there exists a deterministic finite automaton with output (DFAO) $\mathcal{A}$ such that, for all $n \geq 0$, the nth term $\mathbf{x}(n)$ of $\mathbf{x}$ is given by the output $\mathcal{A}(\mathrm{rep}_U(n))$ of $\mathcal{A}$. In the particular case where U is built on powers of an integer $k \geq 2$, then $\mathbf{x}$ is said to be *k-automatic*. It is known that a sequence is k-automatic if and only if it is the image, under a coding, of a fixed point of a k-uniform morphism [3].

A generalization of automatic sequences to infinite alphabets is the notion of regular sequences [3, 34, 38]. Given a positional numeration system $U = (U(n))_{n\geq 0}$, a sequence $\mathbf{x}$ is *U-regular* if there exist a column vector λ, a row vector γ and *matrix-valued* morphism μ, i.e., the image of each letter is a matrix, such that $\mathbf{x}(n) = \lambda\mu(\mathrm{rep}_U(n))\gamma$. Such a system of matrices forms a *linear representation* of $\mathbf{x}$. In the particular case where U is built on powers of an integer $k \geq 2$, then $\mathbf{x}$ is said to be *k-regular*. Another definition of k-regular sequences is the following one [3]. Consider a sequence $\mathbf{x}$ and an integer $k \geq 2$. The *k-kernel* of $\mathbf{x}$ is the set of subsequences of the form $(\mathbf{x}(k^e n + r))_{n\geq 0}$ where $e \geq 0$ and $r \in \{0, 1, \ldots, k^e - 1\}$. Equivalently, a sequence is k-regular if the $\mathbb{Z}$-module generated by its k-kernel is finitely generated. A sequence is then k-automatic if and only if its k-kernel is finite [3].

Introduced in 2001 by Carpi and Maggi [13], synchronized sequences form a family between automatic and regular sequences. Given a positional numeration system $U = (U(n))_{n\geq 0}$, a sequence $\mathbf{x}$ is *U-synchronized* if there exists a deterministic finite automaton (DFA) that recognizes the language of U-representations of n and $\mathbf{x}(n)$ in parallel.

3. General results

In this section, we gather general results on the additive complexity of infinite words. Since abelian equivalence implies additive equivalence, we have the following lemma.

Lemma 5. *For all infinite words $\mathbf{x}$, we have $\rho_{\mathbf{x}}^{\mathrm{add}}(n) \leq \rho_{\mathbf{x}}^{\mathrm{ab}}(n)$ for all $n \geq 0$.*

As in the case of abelian complexity, we have the following lower and upper bounds for additive complexity. See [19, Rk. 4.07] and [33, Thm 2.4].

Lemma 6. *Let $k \geq 1$ be an integer and let $\mathbf{x}$ be an infinite word on $\{a_1 < \cdots < a_k\}$. We have $1 \leq \rho_{\mathbf{x}}^{\mathrm{add}}(n) \leq \binom{n+k-1}{k-1}$ for all $n \geq 0$.*

Note that the lower bound of the previous result is reached for (purely) periodic sequences. The story about the upper bound is a little more puzzling. In fact, for a window length $N \geq 1$, we can find an alphabet and a sequence over this alphabet for which its additive complexity reaches the stated upper bound on its first N values. Indeed, fix an integer $k \geq 3$ and an alphabet $\Sigma = \{a_1 < \cdots < a_k\}$ of integers. Consider the Champernowne-like sequence defined on Σ by concatenating all words of Σ^* in lexicographic order. Then, for all $N \geq 1$, we can find a valuation of Σ (i.e., a distribution of integral values for the letters of Σ) such that $\rho_{\mathbf{x}}^{\mathrm{add}}(n) = \binom{n+k-1}{k-1}$ for all $n \leq N$. However, it does not seem possible to find a sequence for which its additive complexity always reaches the upper bound. This already highlights the unusual fact that the underlying alphabet of the words plays a crucial role in additive complexity.

The classical theorem of Morse and Hedlund [27] characterizes ultimately periodic infinite words by means of their factor complexity. With the notion of additive complexity, we no longer have a characterization, only the implication below. The converse of Proposition 7 does not hold, as illustrated by several examples in Section 4.

Proposition 7. *The additive complexity of an ultimately periodic word is bounded.*

Similarly, balanced words may be characterized through their abelian complexity. A word $\mathbf{x}$ is said to be *C-balanced* if $||u|_a - |v|_a| \leq C$ for all $a \in \Sigma$ and all factors u, v of $\mathbf{x}$ of equal length. Richomme, Saari and Zamboni [33, Lemma 3] proved that an infinite word $\mathbf{x}$ is C-balanced for some $C \geq 1$ if and only if $\rho_{\mathbf{x}}^{\mathrm{ab}}$ is bounded. In our case, we only have one implication, as stated in Proposition 8, and we also provide an upper bound.

Proposition 8. *Let $\Sigma = \{a_1 < \cdots < a_k\}$ and let $\mathbf{x}$ be a C-balanced word on Σ. Then the additive complexity of $\mathbf{x}$ is bounded by a constant. More precisely, we have $\rho_{\mathbf{x}}^{\mathrm{add}}(n) \leq C \sum_{i=1}^{\lceil k/2 \rceil}(a_i - a_{k+1-i}) + 1$ for all $n \geq 0$.*

Proof. For all length-n factors y, z of $\mathbf{x}$ and $a \in \Sigma$, we have $||y|_a - |z|_a| \leq C$. So the largest possible gap between the sum of letters of y and the sum of letters of z is when, for all $i \in \{1, \ldots, \lceil k/2 \rceil\}$, $|y|_{a_i} = |z|_{a_i} + C$ and $|y|_{a_{k+1-i}} = |z|_{a_{k+1-i}} - C$, or vice versa (in short, we swap C letters from a_k to a_1, C others from a_{k-1} to a_2, and so on and so forth). □

Note that Proposition 8 is a particular case of [6, Theorem 4]. However, there are infinite words with bounded additive complexity and unbounded abelian complexity, making them not balanced. For an example, see Section 4.2.

Computing additive and abelian complexity might be "easy" in some cases. Recently, Shallit [39] provided a general method to compute the abelian complexity of an automatic sequence under some hypotheses.

Theorem 9 ([39, Thm. 1])**.** *Let $\mathbf{x}$ be a sequence that is automatic in some regular numeration system. Assume that*

1. *the abelian complexity $\rho_{\mathbf{x}}^{\mathrm{ab}}$ of $\mathbf{x}$ is bounded above by a constant, and*
2. *the Parikh vectors of length-n prefixes of $\mathbf{x}$ form a synchronized sequence.*

Then $\rho_{\mathbf{x}}^{\mathrm{ab}}$ is an automatic sequence and the DFAO computing it is effectively computable.

We obtain an adapted version in the framework of additive complexity.

Theorem 10. *Let $\mathbf{x}$ be a sequence that is automatic in some additive numeration system. Assume that*

1. *the additive complexity $\rho_{\mathbf{x}}^{\mathrm{add}}$ of $\mathbf{x}$ is bounded above by a constant, and*
2. *the Parikh vectors of length-n prefixes of $\mathbf{x}$ form a synchronized sequence.*

Then $\rho_{\mathbf{x}}^{\mathrm{add}}$ is an automatic sequence and the DFAO computing it is effectively computable.

Proof. Let $\Sigma = \{a_1 < \cdots < a_k\} \subset \mathbb{N}$ be an ordered finite alphabet. The *weighted Parikh vector* of a finite word $w \in \Sigma^*$ is $\psi^*(w) = (a_1|w|_{a_1}, \ldots, a_k|w|_{a_k})$. Then two words x, y are additively equivalent if and only if $\sum_{a\in\Sigma}[\psi^*(x)]_a = \sum_{a\in\Sigma}[\psi^*(y)]_a$, where $[\psi^*(x)]_a$ designates the ath component of the vector $\psi^*(x)$. We adapt the proof of [39, Thm. 1] in the framework of the additive complexity. The steps to find the automaton computing the additive complexity $\rho_{\mathbf{x}}^{\mathrm{add}}$ are the following:

1. Since the Parikh vectors of length-n prefixes of $\mathbf{x}$ form a synchronized sequence by assumption, so are the weighted Parikh vectors for arbitrary length-n factors $\mathbf{x}[i..i+n-1]$. This is expressible in first-order logic.
2. For $i \geq 0$ and $n \geq 1$, let us denote $\Delta_{\mathbf{x}}(i, n)$ the following integer

$$\Delta_{\mathbf{x}}(i,n) = \sum_{a\in\Sigma}[\psi^*(\mathbf{x}[i..i+n-1])]_a - \sum_{a\in\Sigma}[\psi^*(\mathbf{x}[0..n-1])]_a.$$

The additive complexity $\rho_{\mathbf{x}}^{\mathrm{add}}$ is bounded if and only if there is a constant C such that the cardinality of the set $A_n^* := \{\Delta_{\mathbf{x}}(i,n) : i \geq 0\}$, is bounded above by C for all $n \geq 1$.

(3) In this case, the range of possible values of A_n^* is finite (it may take at most $2C+1$ values) and can be computed algorithmically.
(4) Once this range is known, there are finitely many possibilities for $\Delta_{\mathbf{x}}(i,n)$ for all $i \geq 0$. Then, we compute the set S of all of these possibilities.
(5) Once we have S, we can test each of the finitely many values to see if it occurs for some n, and we obtain an automaton recognizing those n for which it does.
(6) All the different automata can then be combined into a single DFAO computing $\rho_{\mathbf{x}}^{\mathrm{add}}(n)$, using the direct product construction.

This finishes the proof. □

Remark 11. The advantage of the proof above is that it is constructive. However, in practice, it will be more convenient to use the so-called *semigroup trick* algorithm, as discussed in [40, § 4.11]. This algorithm should be used when a regular sequence is believed to be automatic, i.e., when it takes only finitely many values. The semigroup trick algorithm halts if and only if the sequence is automatic and produces a DFAO if this is the case. Therefore, Theorem 10 ensures that, under some mild hypotheses, the algorithm halts.

Theorem 10 may be applied to a particular family of infinite words: those that are generated by so-called Parikh-collinear morphisms. In recent years, combinatorists have been studying them; see, e.g., [16, 2, 35, 36].

Definition 12. A morphism $\varphi\colon \Sigma^* \to \Delta^*$ is *Parikh-collinear* if the Parikh vectors $\Psi(\varphi(a))$, $a \in \Sigma$, are collinear (or pairwise $\mathbb{Z}$-linearly dependent). In other words, the associated *adjacency matrix* of φ, i.e., the matrix whose columns are the vectors $\Psi(\varphi(a))$, for all $a \in \Sigma$, has rank 1.

Theorem 13 ([35, 36])**.** *Let $\varphi\colon \Sigma^* \to \Sigma^*$ be a Parikh-collinear morphism prolongable on the letter a, and write $\mathbf{x} := \varphi^\omega(a)$. Then the abelian complexity function $\rho_{\mathbf{x}}^{\mathrm{ab}}$ of $\mathbf{x}$ is k-automatic for $k = \sum_{b\in\Sigma} |\varphi(b)|_b$. Moreover, the automaton generating $\rho_{\mathbf{x}}^{\mathrm{ab}}$ can be effectively computed given φ and a.*

Putting together Lemma 5 and Theorem 13, we obtain the following.

Corollary 14. *Let $\mathbf{x}$ be a fixed point of a Parikh-collinear morphism. Then the abelian and additive complexity functions of $\mathbf{x}$ are bounded.*

Theorem 15. *Let $\varphi\colon \Sigma^* \to \Sigma^*$ be a Parikh-collinear morphism prolongable on the letter a, and write $\mathbf{x} := \varphi^\omega(a)$. The additive complexity function $\rho_{\mathbf{x}}^{\mathrm{add}}$ of $\mathbf{x}$ is k-automatic for $k = \sum_{b\in\Sigma} |\varphi(b)|_b$. Moreover, the automaton generating $\rho_{\mathbf{x}}^{\mathrm{add}}$ can be effectively computed given φ and a.*

Proof. By Corollary 14, $\rho_{\mathbf{x}}^{\mathrm{add}}$ is bounded by a constant, so Item 1 of Theorem 10 is satisfied. Then Item 2 of of Theorem 10 holds by [36, Lemma 26]. Hence, Theorem 10 allows to finish the proof. □

We now give a detailed example of Theorem 15. Let $f\colon \{0,1,2\}^* \to \{0,1,2\}$ be defined by $0 \mapsto 012, 1 \mapsto 112002, 2 \mapsto \varepsilon$. Since the three vectors $\Psi(f(0)) = (1,1,1)$, $\Psi(f(1)) = (2,2,2)$ and $\Psi(f(2)) = (0,0,0)$ are collinear, it follows that f is Parikh-collinear. Consider $\mathbf{x} = 012112002112002\cdots$, the fixed point of f starting with 0. In [36], the authors proved that the abelian complexity of $\mathbf{x}$ is equal to the eventually periodic word $135(377)^\omega$. We have a similar result for additive complexity.

Proposition 16. *Let $f\colon \{0,1,2\}^* \to \{0,1,2\}, 0 \mapsto 012, 1 \mapsto 112002, 2 \mapsto \varepsilon$. The additive complexity of the fixed point* $\mathbf{x} = 0121120022112002\cdots$ *of f is equal to* $134(355)^\omega$.

Proof. Computing $\sum_{a=0}^{2}|f(a)|_a = 3$, we know from classical results that $\mathbf{x}$ is 3-automatic. We thus know that $\mathbf{x}$ is generated by a 3-uniform morphism. Following the procedure of [35], we have $\mathbf{x} = \tau(h^\omega(0))$ with $h : 0,6 \mapsto 012,\ 1,4 \mapsto 134,\ 2,3,5 \mapsto 506$, and the coding $\tau : 0,5 \mapsto 0,\ 1,3 \mapsto 1$, and $2,4,6 \mapsto 2$.

In Walnut, we can compute the synchronized functions fac0, fac1 and fac2 that computes the number of letter 0, 1 and 2 in every factor of $\mathbf{x}$, see [36] for more details. First, we introduce the 3-automatic word $\mathbf{x}$ to Walnut as follows:

```
morphism h "0->012 1->134 2->506 3->506 4->134 5->506 6->012";
morphism tau "0->0 1->1 2->2 3->1 4->2 5->0 6->2";
promote H h;
image X tau H;
```

From [35], we know that the sequence mapping $n \geq 0$ to the number of each letter in the prefix of length $n+1$ of $\mathbf{x}$ is synchronized. In Walnut, we require the following commands:

```
def cut "?msd_3 n=0 | (n>=3 & X[n-1]=@2 & ~(X[n-3]=@1))";
def prev "?msd_3 x<=n & $cut(x) & (Ay (y>x & y<=n)=>~$cut(y))";
def prefn0 "?msd_3 (n<=2 & y=1) | (3<=n & Em,z ($prev(n,m)
    & 3*y=m+3*z & ((X[m]=@0 & z=1) | (X[m]=@1 & ((n<m+3 & z=0)
    | (n=m+3 & z=1) | (n>=m+4 & z=2))))))";
def prefn1 "?msd_3 Em,z $prev(n,m) & 3*y=m+3*z
    & ((X[m]=@0 & ((m=n & z=0) | (n>=m+1 & z=1)))
    | (X[m]=@1 & ((m=n & z=1) | (n>=m+1 & z=2))))";
def prefn2 "?msd_3 Em,z $prev(n,m) & 3*y=m+3*z
    & ((X[m]=@0 & ((n<m+2 & z=0) | (m+2=n & z=1)))
    | (X[m]=@1 & ((n<m+2 & z=0) | (n>=m+2 & n<m+5 & z=1)
    | (n=m+5 & z=2))))";
```

The next step is to determine the number of each letter in every factor of $\mathbf{x}$. We compute the corresponding synchronized functions $n \mapsto |\mathbf{x}[i..i+n-1]|_a$ for $a \in \{0,1,2\}$ in the following way:

```
def fac0 "?msd_3 Aq,r ($prefn0(i+n,q) & $prefn0(i,r)) => (q=r+s)":
def fac1 "?msd_3 Aq,r ($prefn1(i+n,q) & $prefn1(i,r)) => (q=r+s)":
def fac2 "?msd_3 Aq,r ($prefn2(i+n,q) & $prefn2(i,r)) => (q=r+s)":
```

Next, wecompute additive complexity of the fixed point $\mathbf{x}$ of f. So we test whether the factors $u = \mathbf{x}[i..i+n-1]$ and $v = \mathbf{x}[j..j+n-1]$ of $\mathbf{x}$ are additively equivalent. For that, it is enough to check the equality between the quantities $|u|_1 + 2|u|_2$ and $|v|_1 + 2|v|_2$.

```
def addFacEq "?msd_3 Ep,q,r,s $fac1(i,n,p) & $fac2(i,n,q)
    & $fac1(j,n,r) & $fac2(j,n,s) & p+2*q=r+2*s":
```

Finally, we write that $\mathbf{x}[i..i+n-1]$ is a novel occurrence of a length-n factor of $\mathbf{x}$ representing its additive equivalence class and obtain a linear representation for the number of such positions i as follows:

```
eval addCompRepLin n "?msd_3 Aj j<i => ~$addFacEq(i,j,n)":
```

`Walnut` then returns a linear representation of size 55.

The first step is to take the linear representation computed by `Walnut`, and minimize it. The result is a linear representation of rank 7, using the algorithm in [9, § 2.3]. Once we have this linear representation, we can carry out the so-called semigroup trick algorithm, as discussed in [40, § 4.11]. As it terminates, we prove that the additive complexity of the word $\mathbf{x}$ is bounded, and takes on only the values $\{1, 3, 4, 5\}$ for $n \geq 0$. Furthermore, it produces a 4-state DFAO computing the additive complexity, called `addCompExample`, that we display in Figure 1.

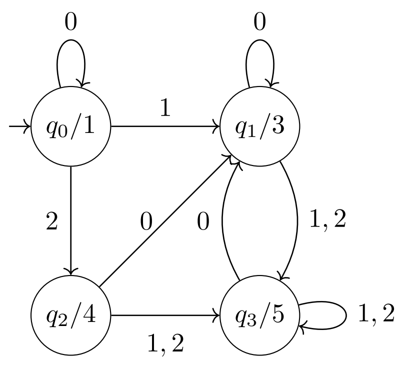

FIGURE 1. A four-state DFAO computing the additive complexity of the fixed point of $f\colon \{0,1,2\}^* \to \{0,1,2\}, 0 \mapsto 012, 1 \mapsto 112002, 2 \mapsto \varepsilon$.

By inspecting this DFAO, we easily prove that the additive complexity of $\mathbf{x}$ is $134(355)^\omega$. This could also be checked easily with `Walnut` with the following commands:

```
reg form3 msd_3 "0*(0|1|2)*0":
eval check3 "?msd_3 An ($form3(n) & n>=3) => addCompExample[n]=@3":
reg form5 msd_3 "0*(0|1|2)*(1|2)":
eval check5 "?msd_3 An ($form5(n) & n>=3) => addCompExample[n]=@5":
```

and both return `True`. Notice that these two forms cover all integers $n \geq 3$ and the first few values can be checked by hand. □

4. Different behaviors and curiosities

In this section, we exhibit different behaviors between the additive and abelian complexity functions by making use of the software `Walnut`. By Lemma 5, the behavior of additive complexity of a sequence is constrained by its abelian complexity. Here we show that the functions may behave differently; in particular, see Section 4.2.

4.1. **Bounded additive and abelian complexities.**

4.1.1. *The Tribonacci word.* The *Tribonacci word* $\mathbf{tr}$ is the fixed point of the morphism $0 \mapsto 01$, $1 \mapsto 02$, $2 \mapsto 0$. This well-known word belongs to the family of episturmian words, a generalization of the famous Sturmian words. This word is *Tribonacci*-automatic, where the underlying numeration system is built on the sequence of *Tribonacci numbers* defined by $T(0) = 1$, $T(1) = 2$, $T(2) = 4$, and $T(n) = T(n-1) + T(n-2) + T(n-3)$ for all $n \geq 3$. Notice that this word is not the fixed point of a Parikh-collinear morphism; otherwise it would be k-automatic for some integer $k \geq 2$. A generalization of Cobham's theorem for substitutions [21] would then imply that $\mathbf{tr}$ is ultimately periodic. The possible values of the abelian complexity of the word $\mathbf{tr}$ were studied in [32, Thm. 1.4]. Also see Fig. 2.

Theorem 17 ([32, Thm. 1.4])**.** *Let* $\mathbf{tr}$ *be the Tribonacci word, i.e., the fixed point of the morphism* $0 \mapsto 01$*,* $1 \mapsto 02$*,* $2 \mapsto 0$*. The abelian complexity function* $\rho^{\mathrm{ab}}_{\mathbf{tr}}$ *takes on only the values in the set* $\{3,4,5,6,7\}$ *for* $n \geq 1$*.*

This result was reproved by Shallit [39] using Walnut by providing an automaton computing $\rho^{\mathrm{ab}}_{\mathbf{tr}}$. Furthermore, this automaton allows us to prove that each value is taken infinitely often. We prove the following result concerning the additive complexity of the Tribonacci word. See again Fig. 2.

Theorem 18. *Let* $\mathbf{tr}$ *be the Tribonacci word, i.e., the fixed point of the morphism* $0 \mapsto 01$*,* $1 \mapsto 02$*,* $2 \mapsto 0$*. The additive complexity function* $\rho^{\mathrm{add}}_{\mathbf{tr}}$ *takes on only the values in the set* $\{3,4,5\}$ *for* $n \geq 1$*. Furthermore, each of the three values is taken infinitely often and it is computed by a 76-state Tribonacci DFAO.*

Proof. We reuse some ideas (especially, Walnut code) from [39, 41]. The Tribonacci word is stored as TRL in Walnut. The synchronized function rst takes the Tribonacci representations of m and n in parallel and accepts if $(n)_T$ is the right shift of $(m)_T$. In Walnut, the following three predicates allow us to obtain DFAO's that compute the maps $n \mapsto |\mathbf{tr}[0..n-1]|_a$ for $a \in \{0,1,2\}$, i.e., the number of letters $0,1,2$ in the length-n prefix of the Tribonacci word $\mathbf{tr}$. Note that the predicates are obtained using a special property of $\mathbf{tr}$; for a full explanation, see [39, Sec. 3].

```
def tribsync0 "?msd_trib Ea Eb (s=a+b) & ((TRL[n]=@0)=>b=0)
    & ((TRL[n]=@1)=>b=1) & $rst(n,a)":
def tribsync1 "?msd_trib Ea Eb Ec (s=b+c) & ((TRL[a]=@0)=>c=0)
    & ((TRL[a]=@1)=>c=1) & $rst(n,a) & $rst(a,b)":
def tribsync2 "?msd_trib Ea Eb Ec Ed (s=c+d) & ((TRL[b]=@0)=>d=0)
    & ((TRL[b]=@1)=>d=1) & $rst(n,a) & $rst(a,b) & $rst(b,c)":
```

From now on, we follow the same steps as Proposition 16. First, we compute the Tribonacci synchronized functions $n \mapsto |\mathbf{tr}[i..i+n-1]|_a$ for $a \in \{0,1,2\}$, that are

```
def tribFac0 "?msd_trib Aq Ar ($tribsync0(i+n,q)
    & $tribsync0(i,r)) => (q=r+s)":
def tribFac1 "?msd_trib Aq Ar ($tribsync1(i+n,q)
    & $tribsync1(i,r)) => (q=r+s)":
def tribFac2 "?msd_trib Aq Ar ($tribsync2(i+n,q)
    & $tribsync2(i,r)) => (q=r+s)":
```

Next, we compute the additive equivalence between two factors, that is the following Tribonacci synchronized function

```
def tribAddFacEq "?msd_trib Ep,q,r,s $tribFac1(i,n,p) & $tribFac2(i,n,q)
    & $tribFac1(j,n,r) & $tribFac2(j,n,s) & p+2*q=r+2*s":
```

Finally, we obtain a linear representation, as defined at the end of Section 2, of the additive complexity as follows

```
eval tribAddCompRepLin n "?msd_trib Aj j<i => ~$tribAddFacEq(i,j,n)":
```

And Walnut then returns a linear representation of size 184. Then we apply the same procedure than in Proposition 16.

After minimization, the result is a linear representation of rank 62 and we carry out the semigroup trick. This algorithm terminates, which proves that the additive complexity of the Tribonacci word is bounded, and takes on only the values

$\{1, 3, 4, 5\}$ for $n \geq 0$. Furthermore, it produces a 76-state DFAO computing the additive complexity. In `Walnut`, let us import this DFAO under the name `TAC`. To show that each value appears infinitely often, we test the following three predicates

```
eval tribAddComp_3 "?msd_trib An Em (m>n) & TAC[m]=@3":
eval tribAddComp_4 "?msd_trib An Em (m>n) & TAC[m]=@4":
eval tribAddComp_5 "?msd_trib An Em (m>n) & TAC[m]=@5":
```

and `Walnut` then returns `TRUE` each time. □

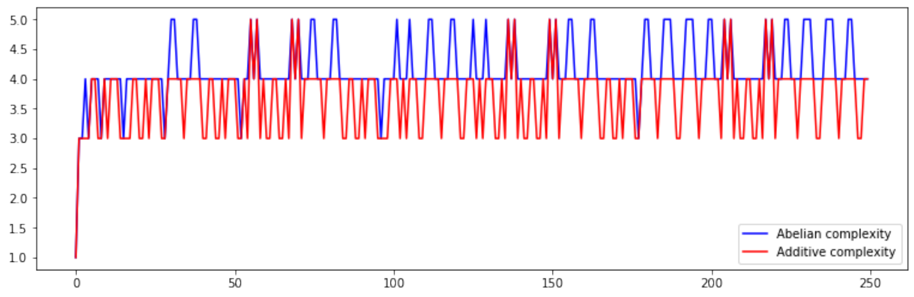

FIGURE 2. The first few values of the abelian and additive complexities for the Tribonacci word.

Remark 19. From the automaton, which is too large to display here, it is easy to find infinite families for each value of the additive complexity function. Indeed, it suffices to detect a loop in the automaton leading to a final state for each value. For instance, we have the following infinite families:

(a) If $(n)_T = 100(100)^k$, for $k \geq 0$, then $\rho^{\text{add}}_{\mathbf{tr}}(n) = 3$.
(b) If $(n)_T = 1101(01)^k$, for $k \geq 0$, then $\rho^{\text{add}}_{\mathbf{tr}}(n) = 4$.
(c) If $(n)_T = 1101001100(1100)^k$, for $k \geq 0$, then $\rho^{\text{add}}_{\mathbf{tr}}(n) = 5$.

One can check with `Walnut` that these infinite families are convenient with the following commands

```
reg form3 msd_trib "0*100(100)*":
reg form4 msd_trib "0*1101(01)*":
reg form5 msd_trib "0*1101001100(1100)*":
eval check3 "?msd_trib An ($form3(n) & n>=1) => TAC[n]=@3":
eval check4 "?msd_trib An ($form4(n) & n>=1) => TAC[n]=@4":
eval check5 "?msd_trib An ($form5(n) & n>=1) => TAC[n]=@5":
```

which returns `TRUE` for each command. One can also notice that from the automaton, we can build infinitely many infinite families of solutions of each of those values. However, the question about the respective proportion of solutions remains open.

Remark 20. With `Walnut`, we can also build a DFAO computing the minimum (resp., maximum) possible sum of a length-n block occurring in $\mathbf{tr}$. Furthermore, for each n, every possible sum between these two extremes actually occurs for some length-n factor in $\mathbf{tr}$.

4.1.2. *The generalized Thue–Morse word on three letters.* We introduce a family of words over three letters that are closed to a generalization of the Thue–Morse word.

Definition 21. Let ℓ, m be integers such that $1 \leq \ell < m$. The (ℓ, m)-*Thue–Morse word* $\mathbf{t}_{\ell,m}$ is the fixed point of the morphism $0 \mapsto 0\ell m$, $\ell \mapsto \ell m 0$, $m \mapsto m 0 \ell$.

In the case where $\ell = 1$ and $m = 2$, we find the so-called *ternary Thue–Morse word* $\mathbf{t}_3$, which is the fixed point of the morphism $0 \mapsto 012$, $1 \mapsto 120$, $2 \mapsto 201$. This word is a natural generalization of the ubiquitous Thue–Morse sequence, since it corresponds to the sum-of-digit function in base 3, taken mod 3.

Theorem 22 ([25, Thm. 4.1])**.** *Consider the ternary Thue–Morse word* $\mathbf{t}_3$*, i.e., the fixed point of the morphism* $0 \mapsto 012$*,* $1 \mapsto 120$*,* $2 \mapsto 201$*. The abelian complexity function* $\rho^{\mathrm{ab}}_{\mathbf{t}_3}$ *is the periodic infinite word* $13(676)^\omega$*.*

Theorem 23. *Consider the ternary Thue–Morse word* $\mathbf{t}_3$*, i.e., the fixed point of the morphism* $0 \mapsto 012$*,* $1 \mapsto 120$*,* $2 \mapsto 201$*. The additive complexity function* $\rho^{\mathrm{add}}_{\mathbf{t}_3}$ *is the periodic infinite word* 135^ω*.*

Proof. The following Walnut provides a linear representation of size 138 for the additive complexity of $\mathbf{t}_3$:

```
morphism h "0->012 1->120 2->201":
promote TMG h:

def tmgPref0 "?msd_3 Er,t n=3*t+r & r<3 & (r=0 => s=t)
    & ((r=1 & TMG[n-1]=@0) => s=t+1)
    & ((r=1 & (TMG[n-1]=@1 | TMG[n-1]=@2)) => s=t)
    & ((r=2 & (TMG[n-1]=@0 | TMG[n-1]=@1)) => s=t+1)
    & ((r=2 & TMG[n-1]=@2) => s=t)":
def tmgPref1 "?msd_3 Er,t n=3*t+r & r<3 & (r=0 => s=t)
    & ((r=1 & TMG[n-1]=@1) => s=t+1)
    & ((r=1 & (TMG[n-1]=@0 | TMG[n-1]=@2)) => s=t)
    & ((r=2 & (TMG[n-1]=@1 | TMG[n-1]=@2)) => s=t+1)
    & ((r=2 & TMG[n-1]=@0) => s=t)":
def tmgPref2 "?msd_3 Eq,r $tmgPref0(n,q) & $tmgPref1(n,r) & q+r+s=n":

def tmgFac0 "?msd_3 Et,u $tmgPref0(i+n,t) & $tmgPref0(i,u) & s+u=t":
def tmgFac1 "?msd_3 Et,u $tmgPref1(i+n,t) & $tmgPref1(i,u) & s+u=t":
def tmgFac2 "?msd_3 Et,u $tmgPref2(i+n,t) & $tmgPref2(i,u) & s+u=t":

def tmgAddFacEq "?msd_3 Ep,q,r,s $tmgFac1(i,n,p) & $tmgFac2(i,n,q)
    & $tmgFac1(j,n,r) & $tmgFac2(j,n,s) & p+2*q=r+2*s":

eval tmgAddCompRepLin n "?msd_3 Aj j<i => ~$tmgAddFacEq(i,j,n)":
```

The end of the proof is the same as for Theorem 18. The size of the minimal linear representation is 13 and the semigroup trick algorithm terminates and produces the 3-state DFAO of Figure 3. The result follows immediately. ☐

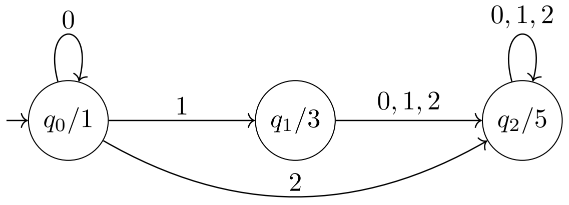

FIGURE 3. A DFAO computing the additive complexity of the (1, 2)-Thue–Morse word.

Changing the letters ℓ and m does not modify the abelian complexity, so for all $1 \le \ell < m$, we have $\rho^{\mathrm{ab}}_{\mathbf{t}_{\ell,m}} = \rho^{\mathrm{ab}}_{\mathbf{t}_3}$. However, additive complexity might change over a different alphabet. In the particular case where $\ell = 1$ and $m = 2$, the following gives an alternative proof of Theorem 23 with only combinatorial tools. Note that the statement on $\rho^{\mathrm{ab}}_{\mathbf{t}_{\ell,m}}$ was also proven in [25], but we provide here a simpler and more concise proof.

Theorem 24. *Let ℓ, m be integers such that $1 \le \ell < m$. Consider the (ℓ, m)-Thue–Morse word, i.e., the fixed point of the morphism $0 \mapsto 0\ell m$, $\ell \mapsto \ell m 0$, $m \mapsto m0\ell$. Then its abelian complexity satisfies $\rho^{\mathrm{ab}}_{\mathbf{t}_{\ell,m}} = 136(766)^\omega$ and its additive complexity satisfies $\rho^{\mathrm{add}}_{\mathbf{t}_{\ell,m}} = \rho^{\mathrm{ab}}_{\mathbf{t}_{\ell,m}}$ if $m \ne 2\ell$, and $\rho^{\mathrm{add}}_{\mathbf{t}_{\ell,m}} = 135^\omega$ if $m = 2\ell$.*

Proof. We clearly have $\rho^{\mathrm{add}}_{\mathbf{t}_{\ell,m}}(0) = 1$, $\rho^{\mathrm{add}}_{\mathbf{t}_{\ell,m}}(1) = 3$, and $\rho^{\mathrm{add}}_{\mathbf{t}_{\ell,m}}(2)$ is equal to 5 or 6 depending on whether $m = 2\ell$ or not. We examine length-n factors of $\mathbf{t}_{\ell,m}$ for $n \ge 2$. Each such factor can be written as $y = pf(x)s$ where f is the morphism $0 \mapsto 0\ell m$, $\ell \mapsto \ell m0$, $m \mapsto m0\ell$ of Definition 21 and p (resp., s) is a proper suffix (resp., prefix) of an image $f(a)$ for $a \in \{0, \ell, m\}$. In particular, note that $p, s \in \{\varepsilon, 0, \ell, m, 0\ell, \ell m, m0\}$. In the following, we examine the *weight* of y, which is the quantity $0 \cdot |y|_0 + \ell \cdot |y|_\ell + m \cdot |y|_m$. More precisely, we count how many different weights y can have, which in turn gives the number of different additive equivalence classes.

First assume that $|y| = 3n$ for some $n \ge 1$. Then we have two cases depending on whether p, s are empty or not. If $p = s = \varepsilon$, then $|x| = n$ and this case corresponds to the first line of Table 1. Otherwise, $|x| = n - 1$ and $|ps| = 3$. In that case, since the roles of p and s are symmetric when computing the weight of the factor, all the possible cases are depicted in Table 1. From the third, fourth and fifth

p	s	$\lvert y\rvert_0$	$\lvert y\rvert_\ell$	$\lvert y\rvert_m$	$0 \cdot \lvert y\rvert_0 + \ell \cdot \lvert y\rvert_\ell + m \cdot \lvert y\rvert_m$
ε	ε	n	n	n	$\ell n + mn$
0	0ℓ	$n+1$	n	$n-1$	$\ell n + m(n-1)$
0	ℓm	n	n	n	$\ell n + mn$
0	$m0$	$n+1$	$n-1$	n	$\ell(n+1) + m(n-1)$
ℓ	0ℓ	n	$n+1$	$n-1$	$\ell n + m(n+1)$
ℓ	ℓm	$n-1$	$n+1$	n	$\ell(n-1) + mn$
ℓ	$m0$	n	n	n	$\ell n + mn$
m	0ℓ	n	n	n	$\ell n + mn$
m	ℓm	$n-1$	n	$n+1$	$\ell n + m(n+1)$
m	$m0$	n	$n-1$	$n+1$	$\ell(n-1) + m(n+1)$

Table 1. The possible weights of factors of the (ℓ, m)-Thue–Morse word $\mathbf{t}_{\ell,m}$ of the form $y = pf(x)s$ where $|y| = 3n$ for some $n \ge 1$.

columns of the table, we observe that there are seven different abelian classes (only the class where $|y|_0 = |y|_\ell = |y|_m = n$ appears more than once) and this proves that $\rho^{\mathrm{ab}}_{\mathbf{t}_{\ell,m}}(3n) = 7$. The corresponding weights of these seven abelian classes can be written as $(n-1) \cdot (\ell + m) + \delta$ with $\delta \in \{\ell, m, 2\ell, \ell + m, 2m, 2\ell + m, \ell + 2m\}$. Since

$$\begin{aligned} \ell < \min\{m, 2\ell\} \le{}& \max\{m, 2\ell\} < \ell + m \\ &< \min\{2m, 2\ell + m\} \le \max\{2m, 2\ell + m\} < \ell + 2m, \end{aligned}$$

this now proves that $\rho^{\mathrm{add}}_{\mathbf{t}_{\ell,m}}(3n)$ is equal to 7 if $m \ne 2\ell$, and to 5 otherwise.

Assume that $|y| = 3n + 1$ for some $n \ge 1$. With similar reasoning and up to the symmetry between p and s, we list all cases in Table 2. Therefore, we see that

p	s	$\lvert y\rvert_0$	$\lvert y\rvert_\ell$	$\lvert y\rvert_m$	$0\cdot\lvert y\rvert_0+\ell\cdot\lvert y\rvert_\ell+m\cdot\lvert y\rvert_m$
0	ε	$n+1$	n	n	$\ell n+mn$
ℓ	ε	n	$n+1$	n	$\ell(n+1)+mn$
m	ε	n	n	$n+1$	$\ell n+m(n+1)$
0ℓ	0ℓ	$n+1$	$n+1$	$n-1$	$\ell(n+1)+m(n-1)$
0ℓ	ℓm	n	$n+1$	n	$\ell(n+1)+mn$
0ℓ	$m0$	$n+1$	n	n	$\ell n+mn$
ℓm	ℓm	$n-1$	$n+1$	$n+1$	$\ell(n-1)+m(n+1)$
ℓm	$m0$	n	n	$n+1$	$\ell n+m(n+1)$
$m0$	$m0$	$n+1$	$n-1$	$n+1$	$\ell(n-1)+m(n+1)$

TABLE 2. The possible weights of factors of the (ℓ,m)-Thue–Morse word $\mathbf{t}_{\ell,m}$ of the form $y=pf(x)s$ where $|y|=3n+1$ for some $n\geq 1$.

there are six different abelian classes, which proves that $\rho^{\text{ab}}_{\mathbf{t}_{\ell,m}}(3n+1)=6$. The corresponding weights of these six abelian classes can be written as $(n-1)\cdot(\ell+m)+\delta$ with $\delta\in\{2\ell,\ell+m,2m,2\ell+m,\ell+2m,2\ell+2m\}$. Since

$$2\ell<\ell+m<\min\{2\ell+m,2m\}\leq\max\{2\ell+m,2m\}<\ell+2m<2\ell+2m,$$

we have that $\rho^{\text{add}}_{\mathbf{t}_{\ell,m}}(3n+1)$ is equal to 6 if $m\neq 2\ell$, and to 5 otherwise.

Finally, assume that $|y|=3n+2$ for some $n\geq 1$. Up to the symmetry between p and s, we list all cases in Table 3. Once again, we see that there are six different

p	s	$\lvert y\rvert_0$	$\lvert y\rvert_\ell$	$\lvert y\rvert_m$	$0\cdot\lvert y\rvert_0+\ell\cdot\lvert y\rvert_\ell+m\cdot\lvert y\rvert_m$
0	0	$n+2$	n	n	$\ell n+mn$
0	ℓ	$n+1$	$n+1$	n	$\ell(n+1)+mn$
0	m	$n+1$	n	$n+1$	$\ell n+m(n+1)$
ℓ	ℓ	n	$n+2$	n	$\ell(n+2)+mn$
ℓ	m	n	$n+1$	$n+1$	$\ell(n+1)+m(n+1)$
m	m	n	n	$n+2$	$\ell n+m(n+2)$
0ℓ	ε	$n+1$	$n+1$	n	$\ell(n+1)+mn$
ℓm	ε	n	$n+1$	$n+1$	$\ell(n+1)+m(n+1)$
$m0$	ε	$n+1$	$n+1$	n	$\ell(n+1)+mn$

TABLE 3. The possible weights of factors of the (ℓ,m)-Thue–Morse word $\mathbf{t}_{\ell,m}$ of the form $y=pf(x)s$ where $|y|=3n+2$ for some $n\geq 0$.

abelian classes, so $\rho^{\text{ab}}_{\mathbf{t}_{\ell,m}}(3n+2)=6$. The corresponding weights of these six abelian classes can be written as $n\cdot(\ell+m)+\delta$ with $\delta\in\{0,\ell,m,2\ell,\ell+m,2m\}$. Since

$$0<\ell<\min\{2\ell,m\}\leq\max\{2\ell,m\}<\ell+m<2m,$$

we have that $\rho^{\text{add}}_{\mathbf{t}_{\ell,m}}(3n+2)$ is equal to 6 if $m\neq 2\ell$, and to 5 otherwise. □

4.2. Bounded additive and unbounded abelian complexities: a variant of the Thue–Morse word.

Thue introduced a variation of his sequence that is sometimes called the *ternary squarefree Thue–Morse word*, and abbreviated as **vtm** (the letter "v" stands for "variant"). It is the sequence [42, A036577] in the OEIS; for more on the word **vtm**, see [7].

Definition 25 (Variant of Thue–Morse). We let **vtm** be the fixed point of $f:0\mapsto 012, 1\mapsto 02, 2\mapsto 1$, starting with 0.

The abelian complexity of the variant of the Thue–Morse word is unbounded.

Theorem 26 ([10, Cor. 1])**.** *Let* **vtm** *be the fixed point of* $f : 0 \mapsto 012, 1 \mapsto 02, 2 \mapsto 1$*, starting with* 0*. Its abelian complexity is* $O(\log n)$ *with constant approaching* $3/4$ *(assuming base-*2 *logarithm), and it is* $\Omega(1)$ *with constant* 3*.*

However, we prove that the additive complexity of the word **vtm** is bounded.

Theorem 27. *Let* **vtm** *be the fixed point of* $f : 0 \mapsto 012, 1 \mapsto 02, 2 \mapsto 1$*, starting with* 0*. Its additive complexity is the periodic infinite word* 13^ω*.*

Proof. Let $n \geq 1$ and $x \in \mathcal{L}_n(\mathbf{vtm})$. Let us prove that $\sum_{a=0}^{2} a \cdot |x|_a \in \{n-1, n, n+1\}$. Write $x = pf(y)s$ where p (resp., s) is a proper suffix (resp., prefix) of an image $f(a)$, $a \in \{0,1,2\}$. Then we have $p \in \{\varepsilon, 12, 2\}$ and $s \in \{\varepsilon, 0, 01\}$. By definition of the morphism f, observe that $|f(y)|_2 = |f(y)|_0$. Therefore, depending on the words p and s, $|x|_2 = |x|_0 + c$ with $c \in \{-1, 0, 1\}$, which suffices since $|x|_0 + |x|_1 + |x|_2 = n$. □

Therefore, the word **vtm** has unbounded abelian complexity and bounded additive complexity; also see Fig. 4. In particular, [33, Lemma 3] implies that **vtm** cannot be balanced, so there exist non-balanced infinite words with bounded additive complexity. Another example exhibiting the same behavior for its abelian and additive complexity is given in [5].

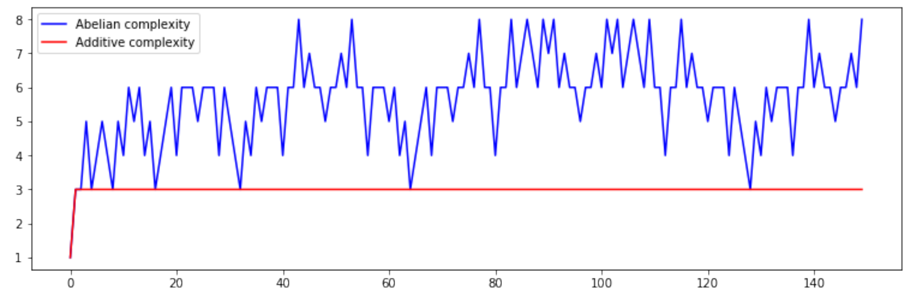

FIGURE 4. The first few values of the abelian and additive complexities for the variant of the Thue–Morse word.

4.3. **Unbounded additive and abelian complexities.**

In this short section, we exhibit a word such that both its additive and abelian complexities are both unbounded.

Theorem 28 ([17, Thm. 1 and Cor. 1])**.** *Let* $\mathbf{x}$ *be the fixed point of the Thue–Morse-like morphism* $0 \mapsto 01$*,* $1 \mapsto 12$*,* $2 \mapsto 20$*. Then* $\rho_{\mathbf{x}}^{\text{add}}(n) = 2\lfloor \log_2 n \rfloor + 3$ *for all* $n \geq 1$*. In particular, the sequence* $(\rho_{\mathbf{x}}^{\text{add}}(n))_{n \geq 0}$ *is* 2*-regular.*

Recall that, for an integer $k \geq 1$, a word w is an *abelian* k*-power* if we can write $w = x_1 x_2 \cdots x_k$ where each x_i, $i \in \{1, \ldots, k\}$, is a permutation of x_1. For instance, `reap`·`pear` and `de`·`ed`·`ed` are respectively an abelian square and cube in English. Similarly, w is an *additive* k*-power* if we can write $w = x_1 x_2 \cdots x_k$ with $|x_i| = |x_1|$ for all $i \in \{1, \ldots, k\}$ and $x_1 \sim_{\text{add}} x_2 \sim_{\text{add}} \cdots \sim_{\text{add}} x_k$. The length of each x_i, $i \in \{1, \ldots, k\}$, is called the *order* of w. As mentioned in the introduction, the following result is one of the main results known on additive complexity.

Theorem 29 ([5, Thm. 2.2])**.** *Let* $\mathbf{x}$ *be an infinite word over a finite subset of* $\mathbb{Z}$*. If* $\rho_{\mathbf{x}}^{\text{add}}$ *is bounded, then* $\mathbf{x}$ *contains an additive* k*-power for every positive integer* k*.*

Proposition 30. *Let* $\mathbf{w}$ *be the fixed point of the morphism* $0 \mapsto 03$, $1 \mapsto 43$, $3 \mapsto 1$, $4 \mapsto 01$. *Then* $\rho^{\mathrm{add}}_{\mathbf{w}}$ *is unbounded.*

Proof. In [14] it is shown that $\mathbf{w}$ is additive-cube-free. The result then follows from Theorem 29. □

However, for the latter word $\mathbf{w}$, it seems interesting to study $\rho^{\mathrm{ab}}_{\mathbf{w}} - \rho^{\mathrm{add}}_{\mathbf{w}}$, since these two complexities are very close. Indeed, surprisingly the first time these two complexities are different appears at $n = 23$, as the two factors 11011031430110343430314 and 30310110110314303434303 are additively but not abelian equivalent. Also, notice that every additive square of the word $\mathbf{w}$ is an abelian square [14, Theorem 5.1]. Together with the fact that this word is additive-cube-free, it shows that abelian and additive properties of this word are relatively close. Indeed, Fig. 5 illustrates that the values of the difference between the additive and abelian complexity functions is close to 0. This motivates the study of the next section.

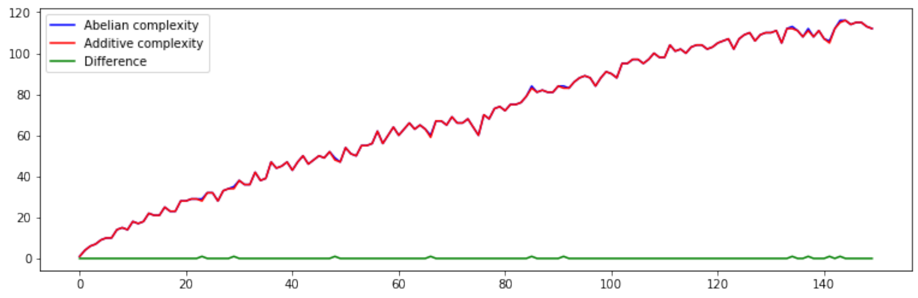

FIGURE 5. The first few values of the abelian and additive complexities as well as their difference for the fixed point of the morphism $0 \mapsto 03$, $1 \mapsto 43$, $3 \mapsto 1$, $4 \mapsto 01$.

5. Equality between abelian and additive complexities

It is clear that abelian complexity does not depend on the values of the alphabet, in contrast with additive complexity. A map $v : A \to \mathbb{N}$ is called a *valuation* over an alphabet A. One might consider the following question.

Question 31. Given an alphabet A, is there a valuation such that the additive complexity of a given sequence is equal to its abelian complexity?

For instance for the word **vtm**, defined originally over the alphabet $\{0, 1, 2\}$, we have already proved in Theorem 27 that $\rho^{\mathrm{add}}_{\mathbf{vtm}}(n) = 3$ for all $n \geq 1$. However, over the alphabet $\{0, 1, 3\}$, i.e., changing 2 into 3, (resp., $\{0, 1, 4\}$), one can easily check that the first time that the additive and abelian complexities are not equal is for $n = 11$ (resp., $n = 43$). But, over the alphabet $\{0, 1, 5\}$, we have observed that both complexities are equal up to $n = 50000$. The main idea is that if the value of a letter is sufficiently large compared to the other values, then two additively equivalent factors are also abelian equivalent. Using this idea, we prove the following theorem.

Theorem 32. *Consider the fixed point* $\mathbf{vtm}_\lambda$ *of the morphism* $f_\lambda : 0 \mapsto 01\lambda, 1 \mapsto 0\lambda, \lambda \mapsto 1$, *where* λ *is a non-negative integer and* $\lambda \geq 2$. *For all* $\lambda \geq 5$, *we have* $\rho^{\mathrm{add}}_{\mathbf{vtm}_\lambda}(n) = \rho^{\mathrm{ab}}_{\mathbf{vtm}_\lambda}(n)$.

Proof. By Lemma 5, it is sufficient to prove that two factors are additively equivalent if and only if they are abelian equivalent.

Let $x, y \in \mathcal{L}_n(\mathbf{vtm}_\lambda)$ such that $x \sim_{\text{add}} y$. Write $x = pf_\lambda(x')s$ where p (resp., s) is a proper suffix (resp., prefix) of an image $f_\lambda(a)$ with $a \in \{0, 1, \lambda\}$. Observe that $p \in \{\varepsilon, \lambda, 1\lambda\}$ and $s \in \{\varepsilon, 0, 01\}$. Also, by definition of the morphism f_λ, we have $|f_\lambda(x')|_\lambda = |f_\lambda(x')|_0$. Therefore, depending on the words p and s, we have $|x|_\lambda = |x|_0 + c_x$ for some $c_x \in \{-1, 0, 1\}$. In a similar way, we have $|y|_\lambda = |y|_0 + c_y$ for some $c_y \in \{-1, 0, 1\}$. By the assumption that $x \sim_{\text{add}} y$, we have $0|x|_0 + 1|x|_1 + \lambda|x|_\lambda = 0|y|_0 + 1|y|_1 + \lambda|y|_\lambda$. From the previous observations, we may write this equality as

$$|x|_0 + |x|_1 + |x|_\lambda + (\lambda - 2)|x|_\lambda + c_x = |y|_0 + |y|_1 + |y|_\lambda + (\lambda - 2)|y|_\lambda + c_y.$$

Since $|x|_0 + |x|_1 + |x|_\lambda = n = |y|_0 + |y|_1 + |y|_\lambda$, we have $(\lambda-2)(|x|_\lambda - |y|_\lambda) = c_y - c_x$. However, $c_y - c_x \in \{-2, \dots, 2\}$ implies that $|x|_\lambda = |y|_\lambda$ and $c_y = c_x$, since $\lambda - 2 \geq 3$. Thus, $|x|_0 = |y|_0$. Since $|x| = n = |y|$, we also have $|x|_1 = |y|_1$, and then that x and y are abelian equivalent. This ends the proof. □

For C-balanced words over an alphabet of fixed size k, we prove that it is always possible to find a valuation for the alphabet such that the additive complexity is the same as the abelian complexity.

Theorem 33. *Let $k, C \geq 1$ be two integers. There exists an alphabet $\Sigma \subset \mathbb{N}$ of size k such that, for each C-balanced word $\mathbf{w}$ over Σ, we have $\rho^{\text{add}}_{\mathbf{w}} = \rho^{\text{ab}}_{\mathbf{w}}$.*

Proof. Such as in the proof of Theorem 32, we prove that over the alphabet Σ, two additively equivalent same-length factors of $\mathbf{w}$ are also abelian equivalent. Let $\Sigma = \{a_1, \dots, a_k\}$ be a subset of $\mathbb{N}$ such that

$$a_1 = 0, a_2 = 1 \quad \text{and} \quad (a_1 + \cdots + a_{j-1})C < a_j \quad \text{for all } 2 \leq j \leq k. \tag{5.1}$$

Now take $x, y \in \mathcal{L}_n(\mathbf{w})$ with $x \sim_{\text{add}} y$. This condition can be rewritten as

$$a_1(|x|_{a_1} - |y|_{a_1}) + \cdots + a_{k-1}(|x|_{a_{k-1}} - |y|_{a_{k-1}}) = a_k(|y|_{a_k} - |x|_{a_k}). \tag{5.2}$$

Observe that the balancedness of $\mathbf{w}$ together with Inequalities (5.1) imply that the left-hand side of Equality (5.2) belongs to the set $\{-a_k + 1, \dots, a_k - 1\}$. Since the right-hand side of Equality (5.2) is a multiple of a_k, we must have

$$\begin{cases} |x|_{a_k} = |y|_{a_k}, \\ a_1(|x|_{a_1} - |y|_{a_1}) + \cdots + a_{k-1}(|x|_{a_{k-1}} - |y|_{a_{k-1}}) = 0. \end{cases} \tag{5.3}$$

Using similar reasoning, replacing Equality (5.2) with Equalities (5.3), we deduce that $|x|_{a_{k-1}} = |y|_{a_{k-1}}$. Continuing in this fashion, we prove that $|x|_a = |y|_a$ for every $a \in \Sigma$, which is enough. □

Remark 34. Since the Tribonacci word $\mathbf{tr}$ is 2-balanced, Theorem 33 implies that over the alphabet $\{0, 1, 3\}$, its additive complexity is equal to its abelian complexity.

6. Abelian and additive powers

In [22, 23], abelian powers of Sturmian words were examined. In particular, the following result was obtained for the Fibonacci word $\mathbf{f} = 0100101001001010010100\cdots$, which is the fixed point of the morphism $0 \mapsto 01, 1 \mapsto 0$. Also, see the sequence [42, A336487] in the OEIS.

Proposition 35 ([22, 23])**.** *Let $k \geq 1$ be an integer and consider the Fibonacci word $\mathbf{f}$, i.e., the fixed point of the morphism $0 \mapsto 01, 1 \mapsto 0$. Then $\mathbf{f}$ has an abelian k-power of order n if and only if $\lfloor k\varphi n \rfloor \equiv 0, -1 \pmod{k}$, where $\varphi = \frac{1+\sqrt{5}}{2}$ is the golden ratio.*

For instance, when $k = 3$, we can compute an 11-state DFA accepting, in Fibonacci representations, exactly those n for which there is an abelian cube of order n in $\mathbf{f}$.

As Arnoux-Rauzy and episturmian sequences generalize Sturmian sequences, it is quite natural to try to understand the orders of abelian and additive powers in these sequences. An archetypical example is the Tribonacci word $\mathbf{tr}$ (recall Section 4.1.1). We obtain the following results on squares and cubes using `Walnut` and the fact that the frequency of each letter $0, 1, 2$ in $\mathbf{tr}$ is Tribonacci-synchronized (see [40, § 10.12] and/or Section 4.1.1).

Theorem 36 ([40, Thm. 10.13.5])**.** *Let* $\mathbf{tr}$ *be the Tribonacci word, i.e., the fixed point of the morphism* $0 \mapsto 01$, $1 \mapsto 02$. *There are abelian squares of all orders in* $\mathbf{tr}$. *Furthermore, if we consider two abelian squares* xx' *and* yy' *to be equivalent if* $x \sim_{\text{ab}} y$, *then every order has either one or two abelian squares. Both possibilities occur infinitely often.*

Theorem 37. *Let* $\mathbf{tr}$ *be the Tribonacci word, i.e., the fixed point of the morphism* $0 \mapsto 01$, $1 \mapsto 02$. *There is a (minimal) Tribonacci automaton of* 1169 *(resp.,* 4927*) states recognizing the Tribonacci representation of those* n *for which there is an abelian (resp., additive) cube of order* n *in* $\mathbf{tr}$.

Proof. For the part about abelian cubes, see [40, p. 295]. See also the respective sequences [42, A345717,A347752] in the OEIS. For the additive cubes, we can determine the orders of additive cubes in $\mathbf{tr}$ with the following function:

```
def tribAddCube "?msd_trib Ei $tribAddFacEq(i,i+n,n)
   & $tribAddFacEq(i,i+2*n,n)":
```

where `tribAddFacEq` is the function defined in the proof of Theorem 18. This leads to a Tribonacci automaton of 4927 states. □

We also note that Theorems 18 and 29 imply the existence of additive k-powers in $\mathbf{tr}$ for all $k \geq 1$. When $k = 2$, additive squares exist for all orders by Theorem 36. For $k = 3$, orders of additive cubes are given in Theorem 37 by a large automaton, and no simple description seems to be possible. When $k = 4$, the same procedure on `Walnut` requires a much larger memory, and it appears that a simple desk computer cannot achieve it. We naturally wonder about larger powers and leave the following as a relatively difficult open question.

Problem 38. Characterize the orders of additive k-powers in the Tribonacci word $\mathbf{tr}$.

It is shown in [15] that the behavior of the abelian complexity of Arnoux-Rauzy words might be erratic. In particular, there exist such words with unbounded abelian complexity. We leave open the research direction of studying the additive complexity of such words and episturmian sequences. For instance, is there a result similar to Proposition 35 in the framework of additive powers?

Appendix A. Semigroup trick

In this section, we give more details about the semigroup trick algorithm discussed in Remark 11, as well as a simple example. For more details, we refer to [40, § 4.11].

Suppose we are given a linear representation of a regular sequence $\mathbf{x}$ in a positional numeration U, i.e., there exist a column vector λ, a row vector γ and a matrix-valued morphism μ such that $\mathbf{x}(n) = \lambda\mu(\text{rep}_U(n))\gamma$. If we suspect that $\mathbf{x}$ takes on only finitely many values, i.e., $\mathbf{x}$ is automatic, one can apply the `Semigroup`

`Trick` algorithm presented in [40, § 4.11]. This algorithm explores the tree of possibilities for the vectors $\lambda\mu(x)$ for $x \in \Sigma^*$ using breadth-first search until no new vector is generated. If the search halts, then the semigroup $\{\lambda\mu(x) : x \in \Sigma^*\}$ is finite. Furthermore, the algorithm constructs a DFAO computing $\mathbf{x}$ by letting the states be the set of distinct vectors that are reachable, the initial state be λ, and the output function associated with each vector w be $w\gamma$.

Example 39. Consider the following linear representation

$$\lambda = \begin{pmatrix} 1 & 0 & 0 & 0 \end{pmatrix},\ \mu(0) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix},\ \mu(1) = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -1 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix},\ \gamma = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}.$$

The first values of the sequence are $0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 1, \ldots$. The steps of the semigroup trick algorithm are the following:

(1) We start with the vector λ and we compute $\lambda\mu(0)$ and $\lambda\mu(1)$. We have $\lambda\mu(0) = \lambda$ and $\lambda\mu(1) = \begin{pmatrix} 0\ 0\ 1\ 0 \end{pmatrix} = w_1$, which is a new state. Therefore we add the transitions $\lambda \xrightarrow{0} \lambda$ and $\lambda \xrightarrow{1} w_1$ in the automaton. We add w_1 to the queue.
(2) We compute $w_1\mu(0)$ and $w_1\mu(1)$. We have $w_1\mu(0) = \lambda$ and $w_1\mu(1) = \begin{pmatrix} 0\ 0\ -1\ 0 \end{pmatrix} = w_2$. Therefore we add the transitions $w_1 \xrightarrow{0} \lambda$ and $w_1 \xrightarrow{1} w_2$ in the automaton. We add w_2 to the queue.
(3) We compute $w_2\mu(0)$ and $w_2\mu(1)$. We have $w_2\mu(0) = \begin{pmatrix} -1\ 0\ 0\ 0 \end{pmatrix} = w_3$ and $w_2\mu(1) = w_1$. Therefore we add the transitions $w_2 \xrightarrow{0} w_3$ and $w_2 \xrightarrow{1} w_1$ in the automaton. We add w_3 to the queue.
(4) We compute $w_3\mu(0)$ and $w_3\mu(1)$. We have $w_3\mu(0) = w_3$ and $w_3\mu(1) = w_2$. Therefore we add the transitions $w_3 \xrightarrow{0} w_3$ and $w_3 \xrightarrow{1} w_2$ in the automaton. Since there is no new state, the algorithm halts.
(5) For each state w, we compute the value $w\gamma$. We have $\lambda\gamma = 0$, $w_1\gamma = 0$, $w_2\gamma = 1$ and $w_3\gamma = 1$, which are the outputs in the DFAO.

Finally, the obtained DFAO for the sequence $\mathbf{x}$ is given in Fig. 6.

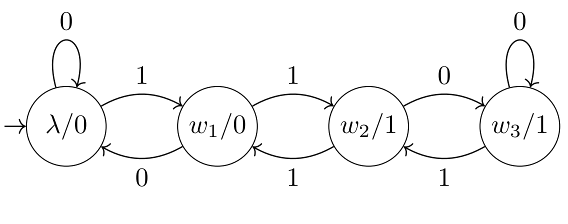

FIGURE 6. An example of DFAO obtained with the semigroup trick algorithm.

Now we recognize the automaton of the famous Rudin–Shapiro sequence. Notice that from a minimal linear representation, the semigroup trick algorithm halts if and only if the sequence is bounded. Since `Walnut` does not provide a minimal linear representation, it may be not sufficient to use the semigroup trick algorithm without minimization. However, for the examples given in this paper, the semigroup trick algorithm halts even before minimization but the obtained automata are not minimal.

Acknolwedgments

Pierre Popoli is supported by ULiège's Special Funds for Research, IPD-STEMA Program. Jeffrey Shallit is supported by NSERC grants 2018-04118 and 2024-03725. Manon Stipulanti is an FNRS Research Associate supported by the Research grant 1.C.104.24F

We thank Matthieu Rosenfeld and Markus Whiteland for helpful discussions, and Eric Rowland for implementing useful Mathematica code.

References

[1] Jean-Paul Allouche, John Campbell, Jeffrey Shallit, and Manon Stipulanti. The reflection complexity of sequences over finite alphabets. Preprint available at `https://arxiv.org/abs/2406.09302`.

[2] Jean-Paul Allouche, Michel Dekking, and Martine Queffélec. Hidden automatic sequences. *Comb. Theory*, 1:15, 2021. Id/No 20. `doi:10.5070/C61055386`.

[3] Jean-Paul Allouche and Jeffrey Shallit. *Automatic sequences: Theory, applications, generalizations*. Cambridge University Press, Cambridge, 2003. `doi:10.1017/CBO9780511546563`.

[4] Jonathan Andrade and Lucas Mol. Avoiding abelian and additive powers in rich words, 2024. Preprint available at `https://www.arxiv.org/pdf/2408.15390`.

[5] Hayri Ardal, Tom Brown, Veselin Jungić, and Julian Sahasrabudhe. On abelian and additive complexity in infinite words. *Integers*, 12(5):795–804, 2012. `doi:10.1515/integers-2012-0005`.

[6] Graham Banero. On additive complexity of infinite words. *J. Integer Seq.*, 16(1):Article 13.1.5, 20, 2013.

[7] Jean Berstel. Sur les mots sans carré définis par un morphisme. In *Automata, languages and programming (Sixth Colloq., Graz, 1979)*, volume 71 of *Lecture Notes in Comput. Sci.*, pages 16–25. Springer, Berlin-New York, 1979.

[8] Jean Berstel and Dominique Perrin. The origins of combinatorics on words. *Eur. J. Comb.*, 28(3):996–1022, 2007. `doi:10.1016/j.ejc.2005.07.019`.

[9] Jean Berstel and Christophe Reutenauer. *Noncommutative Rational Series With Applications*, volume 137 of *Encyclopedia of Mathematics and Its Applications*. Cambridge Univ. Press, 2011.

[10] Francine Blanchet-Sadri, James D. Currie, Narad Rampersad, and Nathan Fox. Abelian complexity of fixed point of morphism $0 \mapsto 012$, $1 \mapsto 02$, $2 \mapsto 1$. *Integers*, 14:A11, 2014. URL: `http://math.colgate.edu/%7Eintegers/o11/o11.Abstract.html`.

[11] Thomas C. Brown and Allen R. Freedman. Arithmetic progressions in lacunary sets. *Rocky Mountain J. Math.*, 17:587–596, 1987.

[12] Tom Brown. Approximations of additive squares in infinite words. *Integers*, 12(5):805–809, a22, 2012. `doi:10.1515/integers-2012-0006`.

[13] Arturo Carpi and Cristiano Maggi. On synchronized sequences and their separators. *Theor. Inform. Appl.*, 35(6):513–524, 2001. `doi:10.1051/ita:2001129`.

[14] Julien Cassaigne, James D. Currie, Luke Schaeffer, and Jeffrey Shallit. Avoiding three consecutive blocks of the same size and same sum. *J. ACM*, 61(2):Art. 10, 17, 2014. `doi:10.1145/2590775`.

[15] Julien Cassaigne, Sébastien Ferenczi, and Luca Q. Zamboni. Imbalances in Arnoux-Rauzy sequences. *Ann. Inst. Fourier*, 50(4):1265–1276, 2000. `doi:10.5802/aif.1792`.

[16] Julien Cassaigne, Gwénaël Richomme, Kalle Saari, and Luca Q. Zamboni. Avoiding abelian powers in binary words with bounded abelian complexity. *Int. J. Found. Comput. Sci.*, 22(4):905–920, 2011. `doi:10.1142/S0129054111008489`.

[17] Jin Chen, Zhixiong Wen, and Wen Wu. On the additive complexity of a Thue-Morse-like sequence. *Discrete Appl. Math.*, 260:98–108, 2019. `doi:10.1016/j.dam.2019.01.008`.

[18] Alan Cobham. Uniform tag sequences. *Math. Systems Theory*, 6:164–192, 1972. `doi:10.1007/BF01706087`.

[19] Ethan M. Coven and G. A. Hedlund. Sequences with minimal block growth. *Math. Systems Theory*, 7:138–153, 1973. `doi:10.1007/BF01762232`.

[20] James Currie and Narad Rampersad. Recurrent words with constant abelian complexity. *Adv. Appl. Math.*, 47(1):116–124, 2011. `doi:10.1016/j.aam.2010.05.001`.

[21] Fabien Durand. Cobham's theorem for substitutions. *Journal of the European Mathematical Society*, 13(6):1799–1814, September 2011. URL: `https://ems.press/doi/10.4171/jems/294`, `doi:10.4171/jems/294`.

[22] Gabriele Fici, Alessio Langiu, Thierry Lecroq, Arnaud Lefebvre, Filippo Mignosi, Jarkko Peltomäki, and Élise Prieur-Gaston. Abelian powers and repetitions in Sturmian words. *Theoret. Comput. Sci.*, 635:16–34, 2016. `doi:10.1016/j.tcs.2016.04.039`.

[23] Gabriele Fici, Alessio Langiu, Thierry Lecroq, Arnaud Lefebvre, Filippo Mignosi, and Élise Prieur-Gaston. Abelian repetitions in Sturmian words. In *Developments in Language Theory*, volume 7907 of *Lecture Notes in Comput. Sci.*, pages 227–238. Springer, Heidelberg, 2013. `doi:10.1007/978-3-642-38771-5\_21`.

[24] Lorenz Halbeisen and Norbert Hungerbühler. An application of Van der Waerden's theorem in additive number theory. *INTEGERS*, 0:#A7, 2000. Available online at `https://math.colgate.edu/~integers/a7/a7.pdf`.

[25] Idrissa Kaboré and Boucaré Kientéga. Abelian complexity of Thue-Morse word over a ternary alphabet. In *Combinatorics on words*, volume 10432 of *Lecture Notes in Comput. Sci.*, pages 132–143. Springer, Cham, 2017. URL: `https://doi.org/10.1007/978-3-319-66396-8_13`, `doi:10.1007/978-3-319-66396-8\_13`.

[26] M. Lothaire. *Combinatorics on words*. Cambridge Mathematical Library. Cambridge University Press, Cambridge, 1997. `doi:10.1017/CBO9780511566097`.

[27] Marston Morse and Gustav A. Hedlund. Symbolic dynamics. II: Sturmian trajectories. *Am. J. Math.*, 62:1–42, 1940. `doi:10.2307/2371431`.

[28] Hamoon Mousavi. Automatic theorem proving in Walnut, 2016. Preprint available at `https://arxiv.org/abs/1603.06017`.

[29] Aline Parreau, Michel Rigo, Eric Rowland, and Élise Vandomme. A new approach to the 2-regularity of the ℓ-abelian complexity of 2-automatic sequences. *Electron. J. Comb.*, 22(1):research paper p1.27, 44, 2015. URL: `www.combinatorics.org/ojs/index.php/eljc/article/view/v22i1p27`.

[30] Giuseppe Pirillo and Stefano Varricchio. On uniformly repetitive semigroups. *Semigroup Forum*, 49:125–129, 1994.

[31] Michaël Rao. On some generalizations of abelian power avoidability. *Theoret. Comput. Sci.*, 601:39–46, 2015. `doi:10.1016/j.tcs.2015.07.026`.

[32] Gwénaël Richomme, Kalle Saari, and Luca Q. Zamboni. Balance and abelian complexity of the Tribonacci word. *Adv. in Appl. Math.*, 45(2):212–231, 2010. `doi:10.1016/j.aam.2010.01.006`.

[33] Gwénaël Richomme, Kalle Saari, and Luca Q. Zamboni. Abelian complexity of minimal subshifts. *J. Lond. Math. Soc. (2)*, 83(1):79–95, 2011. `doi:10.1112/jlms/jdq063`.

[34] Michel Rigo and Arnaud Maes. More on generalized automatic sequences. *Journal of Automata, Languages, and Combinatorics*, 7(3):351–376, 2002. `doi:10.25596/jalc-2002-351`.

[35] Michel Rigo, Manon Stipulanti, and Markus A. Whiteland. Automaticity and Parikh-collinear morphisms. In *Combinatorics on words*, volume 13899 of *Lecture Notes in Comput. Sci.*, pages 247–260. Springer, Cham, 2023. `doi:10.1007/978-3-031-33180-0\_19`.

[36] Michel Rigo, Manon Stipulanti, and Markus A. Whiteland. Automatic abelian complexities of Parikh-collinear fixed points, 2024. To be published in Theory Comput. Syst. Preprint available at `https://arxiv.org/abs/2405.18032`.

[37] Julian Sahasrabudhe. Sturmian words and constant additive complexity. *Integers*, 15:Paper No. A30, 8, 2015.

[38] Jeffrey Shallit. A generalization of automatic sequences. *Theoret. Comput. Sci.*, 61(1):1–16, 1988. `doi:10.1016/0304-3975(88)90103-X`.

[39] Jeffrey Shallit. Abelian complexity and synchronization. *Integers*, 21:Paper No. A36, 14, 2021.

[40] Jeffrey Shallit. *The logical approach to automatic sequences—exploring combinatorics on words with* `Walnut`, volume 482 of *London Mathematical Society Lecture Note Series*. Cambridge University Press, Cambridge, 2023.

[41] Jeffrey Shallit. Note on a Fibonacci parity sequence. *Cryptogr. Commun.*, 15(2):309–315, 2023. `doi:10.1007/s12095-022-00592-5`.

[42] Neil J. A. Sloane and et al. The On-Line Encyclopedia of Integer Sequences. URL: `https://oeis.org`.

[43] Ondřej Turek. Abelian complexity and abelian co-decomposition. *Theoret. Comput. Sci.*, 469:77–91, 2013. `doi:10.1016/j.tcs.2012.10.034`.

[44] Ondřej Turek. Abelian complexity function of the Tribonacci word. *J. Integer Seq.*, 18(3):Article 15.3.4, 29, 2015.

Department of Mathematics, University of Liège, Belgium
Email address: `pierre.popoli@uliege.be`

School of Computer Science, University of Waterloo, Canada
Email address: `shallit@waterloo.ca`

Department of Mathematics, University of Liège, Belgium
Email address: `m.stipulanti@uliege.be`