ABELIAN COMPLEXITY AND SYNCHRONIZATION

Jeffrey Shallit[1]
School of Computer Science, University of Waterloo, Waterloo, Ontario, Canada
shallit@uwaterloo.ca

Received: 11/18/20, Revised: 2/19/21, Accepted: 3/15/21, Published: 3/23/21

Abstract

We present a general method for computing the abelian complexity $\rho^{\rm ab}_{\mathbf{s}}$ of an automatic sequence $\mathbf{s}$, provided that (a) the Parikh vectors of the length-n prefixes of $\mathbf{s}$ form a synchronized sequence and (b) the abelian complexity is bounded above by a constant. We illustrate the idea in detail, using the free software `Walnut` to compute the abelian complexity of the Tribonacci word $\mathbf{TR} = 0102010\cdots$, the fixed point of the morphism $0 \to 01$, $1 \to 02$, $2 \to 0$. Previously, Richomme, Saari, and Zamboni showed that the abelian complexity of this word lies in $\{3, 4, 5, 6, 7\}$, and Turek gave a Tribonacci automaton computing it. We are able to "automatically" rederive these results, and more, using the method presented here.

1. Introduction

Let $\Sigma = \{a_1, a_2, \ldots, a_k\}$ be a finite ordered alphabet, with an ordering on the letters given by $a_1 < a_2 < \cdots < a_k$. Let $w \in \Sigma^*$. We define $|w|_{a_i}$ to be the number of occurrences of a_i in w. Thus, for example, $|\texttt{banana}|_{\texttt{a}} = 3$.

The *Parikh vector* $\psi(w)$ for $w \in \Sigma^*$ is defined to be $(|w|_{a_1}, \ldots, |w|_{a_k})$, the k-tuple that counts the number of occurrences of each letter in w. Thus, for example, if $\texttt{v} < \texttt{l} < \texttt{s} < \texttt{e}$, then $\psi(\texttt{sleeveless}) = (1, 2, 3, 4)$.

Let $\mathbf{s} \in \Sigma^\omega$ be an infinite sequence over Σ. The *abelian complexity function* $\rho^{\rm ab}_{\mathbf{s}}(n)$ is defined to be the number of distinct Parikh vectors of length-n factors of $\mathbf{s}$. This concept was introduced by Richomme, Saari, and Zamboni [15] and has been studied extensively since then.

We call a numeration system for $\mathbb{N}$ *regular* if

(a) Elements of $\mathbb{N}$ have unique representations (up to leading zeros);

(b) The set of all valid representations for $\mathbb{N}$ is recognizable by a finite automaton;

(c) The exists another finite automaton recognizing the relation $\{(x, y, z) : x = y+z\}$. By this we mean that there is an automaton recognizing, in parallel, the

[1]Supported by NSERC grant 2018-04118.

representations of x, y, z satisfying $x = y + z$. Here any shorter representation is padded with leading zeros, if necessary, so that the representations of all three numbers have the same length, and can be fed into the automaton digit-by-digit, in parallel.

Examples of regular numeration systems include base k, for integers $k \geq 2$ [1]; Fibonacci numeration [9]; Tribonacci numeration [13]; and Ostrowski numeration systems [2].

In this paper we are concerned with *automatic sequences*. These are sequences $\mathbf{s}$ that can be computed by a DFAO (deterministic finite automaton with output) M in the following sense: we express n in some regular numeration system, and feed M with this representation, most significant digit first. The output associated with the last state reached is then $\mathbf{s}[n]$, the n'th term of the sequence s.

A variation on automatic sequences is the *synchronized sequence*, introduced by Carpi and Maggi [5]. A sequence $(s(n))_{n\geq 0}$ taking values in $\mathbb{N}$ (or $\mathbb{N}^k$) is synchronized if there is an automaton recognizing, in parallel, the representations of n and $s(n)$, with shorter representations padded with leading zeroes, as above.

Abelian complexity, defined above, is a variation on (ordinary) subword complexity $\rho_{\mathbf{s}}(n)$, which counts the number of length-n factors of $\mathbf{s}$. It is known that for automatic sequences, the first difference of the subword complexity function is "automatically computable", in the sense that there is an algorithm that, given the DFAO M, constructs another automaton M' that computes the sequence $(\rho_{\mathbf{s}}(n+1) - \rho_{\mathbf{s}}(n))_{n\geq 0}$ [10]. In contrast, for abelian complexity there is (in general) no such automaton, as proved in [17, Corollary 5.6].

In this note I will show that there is an algorithm that, given an automatic sequence $\mathbf{s}$ satisfying certain conditions, constructs a DFAO computing $\rho_{\mathbf{s}}^{\mathrm{ab}}$. I will illustrate the ideas in detail for the Tribonacci word, fixed point of the morphism $0 \to 01$, $1 \to 02$, $2 \to 0$.

2. The Result and Its Proof

Theorem 1. *Let* $\mathbf{s}$ *be a sequence that is automatic in some regular numeration system. Suppose that*

(a) the Parikh vectors of length-n prefixes of $\mathbf{s}$ *form a synchronized sequence; and*

(b) the abelian complexity $\rho_{\mathbf{s}}^{\mathrm{ab}}$ *is bounded above by a constant.*

Then $(\rho_{\mathbf{s}}^{\mathrm{ab}}(n))_{n\geq 0}$ *is an automatic sequence and the DFAO computing it is effectively computable.*

Furthermore, if condition (a) holds, then condition (b) can be tested algorithmically.

Proof. Here is a summary of how the DFAO computing $\rho_{\mathbf{s}}^{\mathrm{ab}}(n)$ can be automatically constructed. We use the fact that a first-order logical formula φ with addition, and using indexing into an automatic sequence, is itself automatic (that is, there is an automaton recognizing the values of the free variables that make φ true) [6].

1. Given a synchronized automaton computing the Parikh vectors of prefixes of $\mathbf{s}$, we can find (by subtracting) a synchronized automaton computing the Parikh vector for arbitrary factors $\mathbf{s}[i..i+n-1]$. This is first-order expressible. The resulting automaton recognizes (in parallel) triples of the form $(i, n, \psi(\mathbf{s}[i..i+n-1]))$.

2. Define
$$f(i,n) := \psi(\mathbf{s}[i..i+n-1]) - \psi(\mathbf{s}[0..n-1])$$
$$A_n := \{f(i,n) \,:\, i \geq 0\}.$$
Then $\rho_{\mathbf{s}}^{\mathrm{ab}}(n) = |A_n|$. (The $f(i,n)$ were called *relative Parikh vectors* by by Turek [20].)

3. If $\sup_{n\geq 0} \rho_{\mathbf{s}}^{\mathrm{ab}}(n) < \infty$, then the range of each coordinate of the elements of A_n is finite, and automata recognizing the representation of the ranges can be algorithmically constructed. It is now easy to check whether these automata recognize finite or infinite sets. If all of these automata recognize only finite sets, then condition (b) is satisfied.

4. If condition (b) is satisfied, then there are only finitely many possibilities for each coordinate of $f(i,n)$. We can then compute the (finite) set of all possibilities S.

5. Once we have S, we can test each of the finitely many subsets to see if whether it actually occurs for some n, and obtain an automaton recognizing those n for which it does.

6. All the different automata can then be combined into a single DFAO that, given n as input, computes A_n and $\rho_{\mathbf{s}}^{\mathrm{ab}}(n)$, using the direct product construction discussed in [18, Lemma 2.4].

□

With the exception of the last step, all these steps can be achieved using the free software `Walnut`. We implemented the last step in `Dyalog APL`, although it should be a feature of a future version of `Walnut`.

We now illustrate the ideas in detail on a particular infinite word, the Tribonacci word. In doing so, we recover almost all of the results from two papers: [14] and [20]. In particular, we avoid the long, *ad hoc*, and case-based reasoning of the first paper; it is replaced by calculations done "automatically" by `Walnut`. The second paper, by Turek, is much closer to our approach in spirit and details, but still has some *ad hoc* aspects. In an earlier paper on Tribonacci-automatic sequences [13], we stated that we could not yet obtain the abelian complexity of the Tribonacci word, a deficiency we remedy here.

3. The Tribonacci Word and Tribonacci Representations

The infinite Tribonacci word $\mathbf{TR} = \mathbf{TR}[0..\infty) = 0102010\cdots$ is the fixed point of the map $0 \to 01$, $1 \to 02$, $2 \to 0$. It was studied, for example, in [7, 3, 16].

The word $\mathbf{TR}$ is Tribonacci-automatic. This means there is a DFAO that takes the Tribonacci representation of n as input, and computes $\mathbf{TR}[n]$, the n'th term of the Tribonacci sequence. (Indexing starts with 0.) Here the Tribonacci representation of a natural number n is a binary word $w = e_1 e_2 \cdots e_r$ such that $n = \sum_{1 \le i \le r} e_i T_{r+2-i}$, where $(T_i)_{i \ge 0}$ is the Tribonacci sequence, defined by $T_0 = 0$, $T_1 = 1$, $T_2 = 1$, and $T_n = T_{n-1} + T_{n-2} + T_{n-3}$ for $n \ge 3$. The Tribonacci representation for n is unique, provided the word w begins with 1 (for $n \ge 1$) and contains no block of the form 111; see [4]. We write this unique representation as $(n)_T$. If $w = e_1 \cdots e_r$, the inverse map $[w]_T$ maps a binary word to the sum $\sum_{1 \le i \le r} e_i T_{r+2-i}$.

More specifically, we have that

$$\mathbf{TR}[n] = \begin{cases} 0, & \text{if } (n)_T \text{ ends in } 0; \\ 1, & \text{if } (n)_T \text{ ends in } 01; \\ 2, & \text{if } (n)_T \text{ ends in } 11. \end{cases}$$

To see that the Parikh vector of length-n prefixes of $\mathbf{TR}$ is synchronized, it suffices to show that each of the three functions $n \to |\mathbf{TR}[0..n-1]|_a$ is synchronized, for $a \in \{0, 1, 2\}$. This is easily seen using the following relations. Write $(n)_T = e_1 e_2 \cdots e_r$. Then

$$\begin{aligned} |\mathbf{TR}[0..n-1]|_0 &= [e_1 \cdots e_{r-1}]_T + e_r \\ |\mathbf{TR}[0..n-1]|_1 &= [e_1 \cdots e_{r-2}]_T + e_{r-1} \\ |\mathbf{TR}[0..n-1]|_2 &= [e_1 \cdots e_{r-3}]_T + e_{r-2} \end{aligned} \tag{1}$$

See, for example, [13, Theorem 20] and [8, Theorem 13].

In the next section we prove, using a computation by `Walnut`, the following theorem about $\mathbf{TR}$:

Theorem 2. *The following hold:*

(a) The only possible relative Parikh vectors for $\mathbf{TR}$ *are* $\{(0,0,0), (1,0,-1), (1,-1,0), (0,1,-1), (-1,2,-1), (-1,1,0), (0,-1,1), (-1,0,1), (-1,-1,2)\}$.

(b) *There are only 26 distinct* A_n*, and they are given by* A_i *for* i *in the set* $\mathcal{T} = \{0, 1, 2, 3, 4, 5, 6, 9, 11, 17, 30, 31, 55, 57, 101, 105, 185, 340, 341, 342, 355, 629, 653, 1157, 1201, 3914\}$.

(c) *For all* $i \in \mathcal{T} - \{0\}$*, the set* $\{j \geq 0 : A_j = A_i\}$ *is infinite.*

(d) *The set-valued function* $n \to A_n$ *is Tribonacci-automatic.*

(e) *The function* $n \to \rho^{\rm ab}_{\mathbf{TR}}(n)$ *is Tribonacci-automatic.*

(f) *The abelian complexity function* $\rho^{\rm ab}_{\mathbf{TR}}(n)$ *takes on only the values* $\{3, 4, 5, 6, 7\}$ *for* $n \geq 1$.

The first four results are new. Result (e) was proved previously by Turek [20, Theorem 10]. Result (f) was proved previously by Richomme, Saari, and Zamboni [14, Theorem 1.4].

4. `Walnut` Implementation of the Algorithm for TR

In this section we show how to implement our ideas using the word **TR** as an example, thus providing a proof of Theorem 2. The code is given in `Walnut`, a theorem-prover for first-order logical formulas on automatic sequences [12].

Let us start by implementing the formulas (1) given above.

In order to do this, we need to be able to tell whether a number is a right-shift of another (in Tribonacci representation). For example, $6 = [110]_T$ is the right shift of $12 = [1101]_T$. It is easy to create a 2-D Tribonacci automaton for this, and it is illustrated in Figure 1. It takes the Tribonacci representations of m and n in parallel, and accepts if $(n)_T$ is the right shift of $(m)_T$. We call the result `$rst` and we store it in the `Result` directory of `Walnut`. We also need to be able to compute the last bit of n in Tribonacci representation. Again, this is very easy, and an automaton computing it is illustrated in Figure 2. We call the result `TRL`, and the appropriate file is stored under the name `TRL.txt` in the `Word Automata Library` of `Walnut`.

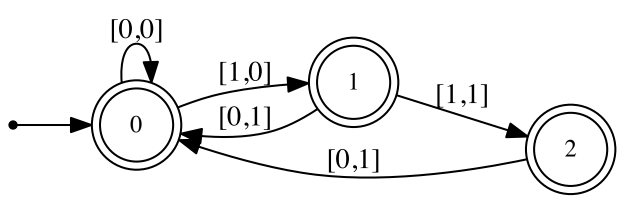

Figure 1: Tribonacci shift is synchronized.

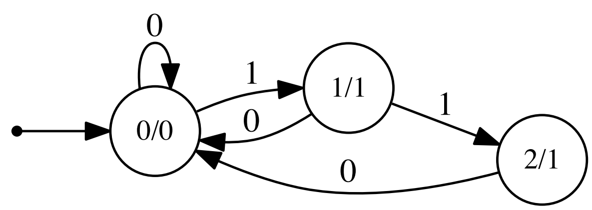

Figure 2: DFAO for the last bit.

Now, using this, we can find DFAO's computing the maps $n \to |\mathbf{TR}[0..n-1]|_a$ for $a \in \{0, 1, 2\}$. Here is the `Walnut` code:

```
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)":
```

This gives three synchronized automata computing $|\mathbf{TR}[0..n-1]|_a$ for $a \in \{0, 1, 2\}$, in Figures 3–5. We could, if we wished, combine the three automata in those figures into one synchronized automaton computing all three elements of the Parikh vector of $\mathbf{TR}[0..n-1]$, but the resulting automaton has 31 states and is a little awkward to display.

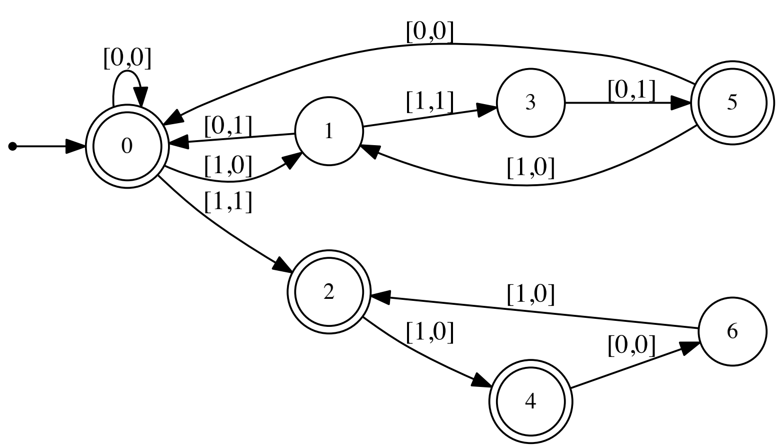

Figure 3: Synchronized automaton for $|\mathbf{TR}[0..n-1]|_0$.

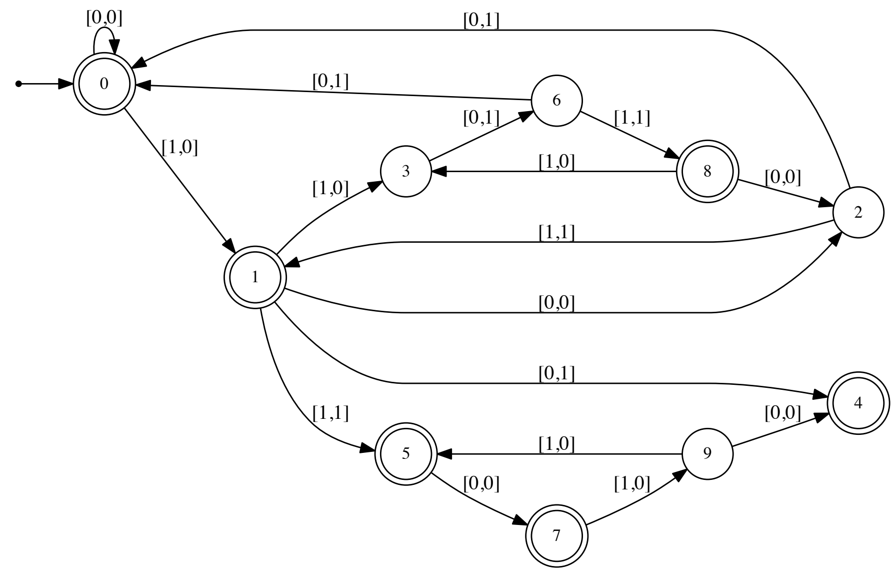

Figure 4: Synchronized automaton for $|\mathbf{TR}[0..n-1]|_1$.

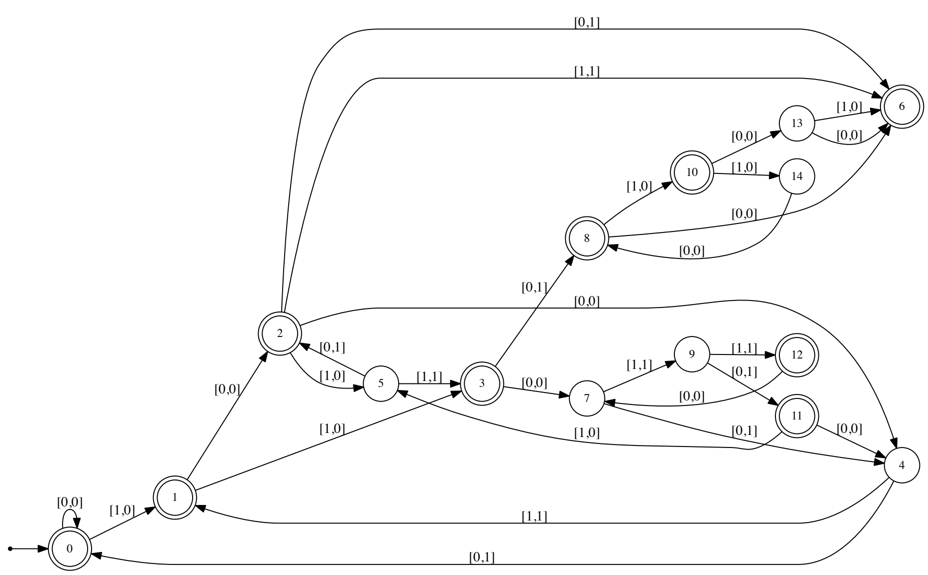

Figure 5: Synchronized automaton for $|\mathbf{TR}[0..n-1]|_2$.

Next we compute synchronized Tribonacci automata computing $|\mathbf{TR}[i..i+n-1]|_a$ for $a \in \{0,1,2\}$. We can do this with the following first-order formulas computing

$$|\mathbf{TR}[0..i+n-1]|_a - |\mathbf{TR}[0..i-1]|_a,$$

namely

```
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)":
```

The resulting automata have 239, 283, and 406 states, respectively — much too large to display here.

We now move on to computing the abelian complexity of **TR**. We want to compute the number of distinct triples $\psi(\mathbf{TR}[i..i+n-1])$ for $i \geq 0$. This is the same as the number of distinct triples of the form

$$f(i,n) := \psi(\mathbf{TR}[i..i+n-1]) - \psi(\mathbf{TR}[0..n-1]).$$

In the notation used in Section 2, we then have

$$A_n := \{f(i,n) \,:\, i \geq 0\}.$$

In order to determine $f(i,n)$, we first need to know the range of each coordinate. Since this range could include negative integers, and `Walnut` is currently restricted to $\mathbb{N}$, this is slightly awkward, but can be done as follows:

```
def posrange0 "?msd_trib E i,n,s,t $tribfac0(i,n,s) & $tribfac0(0,n,t) & u+t=s":
def negrange0 "?msd_trib E i,n,s,t $tribfac0(i,n,s) & $tribfac0(0,n,t) & u+s=t":
def posrange1 "?msd_trib E i,n,s,t $tribfac1(i,n,s) & $tribfac1(0,n,t) & u+t=s":
def negrange1 "?msd_trib E i,n,s,t $tribfac1(i,n,s) & $tribfac1(0,n,t) & u+s=t":
def posrange2 "?msd_trib E i,n,s,t $tribfac2(i,n,s) & $tribfac2(0,n,t) & u+t=s":
def negrange2 "?msd_trib E i,n,s,t $tribfac2(i,n,s) & $tribfac2(0,n,t) & u+s=t":
```

By inspecting the resulting six automata, we see that

$$f(i,n) \in \{-1,0,1\} \times \{-1,0,1,2\} \times \{-1,0,1,2\} \tag{2}$$

for all n, which incidentally proves that **TR** has bounded abelian complexity and also proves that **TR** is 2-balanced [14, Theorem 1.3].

We now proceed to determine which specific triples can appear as elements of of A_n. Since we know the range from (2), we can do this as follows:

```
def validtriples "?msd_trib Ei,n,a,b,c,d,e,f $tribfac0(i,n,a) &
   $tribfac0(0,n,b) & s+b=a+1 & $tribfac1(i,n,c) & $tribfac1(0,n,d)
   & t+d=c+1 & $tribfac2(i,n,e) & $tribfac2(0,n,f) & u+f=e+1":
```

Here the free variables (s,t,u) encode the possible elements $f(i,n)+(1,1,1)$, which from (2) is guaranteed to be in $\mathbb{N}^3$. The resulting automaton is as follows:

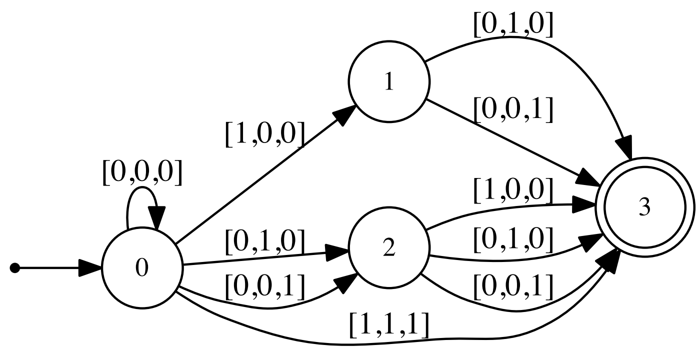

Figure 6: Range of $f(i,n)+(1,1,1)$.

As we can see by inspecting the automaton in Figure 6, the automaton accepts the Tribonacci representation of exactly nine triples, namely

$$\{(1,1,1),(2,1,0),(2,0,1),(1,2,0),(0,3,0),(0,2,1),(1,0,2),(0,1,2),(0,0,3)\}.$$

Thus we have proven that the range of $f(i,n)$ is

$$\{(0,0,0),(1,0,-1),(1,-1,0),(0,1,-1),(-1,2,-1),(-1,1,0),(0,-1,1),(-1,0,1),(-1,-1,2)\}.$$

This proves part (a) of Theorem 2.

It now remains to see which of the 2^9 possible subsets of A can be an A_n for some n. (Actually, since $(0,0,0)$ occurs for each n, a priori there are only 2^8 possible subsets.) We could do this by blindly following the recipe in Section 2, but since most subsets are not reachable, we can use a faster way. First let us make `Walnut` formulas for the assertion that $f(i,n)$ equals each of the nine tuples listed

```
def t000 "?msd_trib Ea,b,c,d,e,f $tribfac0(i,n,a) &
   $tribfac0(0,n,b) & a=b & $tribfac1(i,n,c) & $tribfac1(0,n,d)
   & c=d & $tribfac2(i,n,e) & $tribfac2(0,n,f) & e=f":
def t10m1 "?msd_trib Ea,b,c,d,e,f $tribfac0(i,n,a) &
   $tribfac0(0,n,b) & a=b+1 & $tribfac1(i,n,c) & $tribfac1(0,n,d)
   & c=d & $tribfac2(i,n,e) & $tribfac2(0,n,f) & e+1=f":
def t1m10 "?msd_trib Ea,b,c,d,e,f $tribfac0(i,n,a) &
   $tribfac0(0,n,b) & a=b+1 & $tribfac1(i,n,c) & $tribfac1(0,n,d)
   & c+1=d & $tribfac2(i,n,e) & $tribfac2(0,n,f) & e=f":
def t01m1 "?msd_trib Ea,b,c,d,e,f $tribfac0(i,n,a) &
   $tribfac0(0,n,b) & a=b & $tribfac1(i,n,c) & $tribfac1(0,n,d)
   & c=d+1 & $tribfac2(i,n,e) & $tribfac2(0,n,f) & e+1=f":
def tm12m1 "?msd_trib Ea,b,c,d,e,f $tribfac0(i,n,a) &
   $tribfac0(0,n,b) & a+1=b & $tribfac1(i,n,c) & $tribfac1(0,n,d)
```

```
   & c=d+2 & $tribfac2(i,n,e) & $tribfac2(0,n,f) & e+1=f":
def tm110 "?msd_trib Ea,b,c,d,e,f $tribfac0(i,n,a) &
   $tribfac0(0,n,b) & a+1=b & $tribfac1(i,n,c) & $tribfac1(0,n,d)
   & c=d+1 & $tribfac2(i,n,e) & $tribfac2(0,n,f) & e=f":
def t0m11 "?msd_trib Ea,b,c,d,e,f $tribfac0(i,n,a) &
   $tribfac0(0,n,b) & a=b & $tribfac1(i,n,c) & $tribfac1(0,n,d)
   & c+1=d & $tribfac2(i,n,e) & $tribfac2(0,n,f) & e=f+1":
def tm101 "?msd_trib Ea,b,c,d,e,f $tribfac0(i,n,a) &
   $tribfac0(0,n,b) & a+1=b & $tribfac1(i,n,c) & $tribfac1(0,n,d)
   & c=d & $tribfac2(i,n,e) & $tribfac2(0,n,f) & e=f+1":
def tm1m12 "?msd_trib Ea,b,c,d,e,f $tribfac0(i,n,a) &
   $tribfac0(0,n,b) & a+1=b & $tribfac1(i,n,c) & $tribfac1(0,n,d)
   & c+1=d & $tribfac2(i,n,e) & $tribfac2(0,n,f) & e=f+2":
```

Interestingly enough, each of these automata has 101 states.

We now create a `Walnut` formula of two arguments, m and n, that is true if $A_m = A_n$.

```
def subset "?msd_trib ((Ei $t000(i,m)) <=> (Ej $t000(j,n)))
& ((Ei $t10m1(i,m)) <=> (Ej $t10m1(j,n))) & ((Ei $t1m10(i,m)) <=> (Ej $t1m10(j,n)))
& ((Ei $t01m1(i,m)) <=> (Ej $t01m1(j,n))) & ((Ei $tm12m1(i,m)) <=> (Ej $tm12m1(j,n)))
& ((Ei $tm110(i,m)) <=> (Ej $tm110(j,n))) & ((Ei $t0m11(i,m)) <=> (Ej $t0m11(j,n)))
& ((Ei $tm101(i,m)) <=> (Ej $tm101(j,n))) & ((Ei $tm1m12(i,m)) <=> (Ej $tm1m12(j,n)))":
```

The resulting Tribonacci automaton has 5251 states.

Now we iteratively determine the possible subsets of A that occur as some A_n. We will do this iteratively as follows: Starting with the subset $A_0 = \{(0,0,0)\}$ we find the least n for which a different subset occurs than the ones we found previously. Then we find the subset corresponding to this particular n. This gives us all possible subsets occurring as an A_n and the smallest n for which this subset occurs:

$$
\begin{aligned}
A_0 &= \{(0,0,0)\}\\
A_1 &= \{(-1,0,1),(-1,1,0),(0,0,0)\}\\
A_2 &= \{(0,-1,1),(0,0,0),(1,-1,0)\}\\
A_3 &= \{(-1,0,1),(-1,1,0),(0,-1,1),(0,0,0)\}\\
A_4 &= \{(0,0,0),(0,1,-1),(1,0,-1)\}\\
A_5 &= \{(-1,0,1),(-1,1,0),(0,0,0),(0,1,-1)\}\\
A_6 &= \{(0,-1,1),(0,0,0),(1,-1,0),(1,0,-1)\}\\
A_9 &= \{(-1,0,1),(0,-1,1),(0,0,0),(1,-1,0)\}\\
A_{11} &= \{(-1,1,0),(0,0,0),(0,1,-1),(1,0,-1)\}\\
A_{17} &= \{(0,0,0),(0,1,-1),(1,-1,0),(1,0,-1)\}\\
A_{30} &= \{(0,-1,1),(0,0,0),(0,1,-1),(1,-1,0),(1,0,-1)\}\\
A_{31} &= \{(-1,0,1),(-1,1,0),(0,-1,1),(0,0,0),(0,1,-1)\}\\
A_{55} &= \{(-1,0,1),(-1,1,0),(0,0,0),(0,1,-1),(1,0,-1)\}\\
A_{57} &= \{(-1,0,1),(0,-1,1),(0,0,0),(1,-1,0),(1,0,-1)\}\\
A_{101} &= \{(-1,0,1),(-1,1,0),(0,-1,1),(0,0,0),(1,-1,0)\} \qquad (3)\\
A_{105} &= \{(-1,1,0),(0,0,0),(0,1,-1),(1,-1,0),(1,0,-1)\}\\
A_{185} &= \{(-1,0,1),(-1,1,0),(-1,2,-1),(0,0,0),(0,1,-1)\}\\
A_{340} &= \{(-1,-1,2),(-1,0,1),(0,-1,1),(0,0,0),(1,-1,0)\}\\
A_{341} &= \{(-1,-1,2),(-1,0,1),(-1,1,0),(0,-1,1),(0,0,0)\}\\
A_{342} &= \{(-1,0,1),(-1,1,0),(0,-1,1),(0,0,0),(0,1,-1),(1,0,-1)\}\\
A_{355} &= \{(-1,0,1),(0,-1,1),(0,0,0),(0,1,-1),(1,-1,0),(1,0,-1)\}\\
A_{629} &= \{(-1,0,1),(-1,1,0),(0,-1,1),(0,0,0),(1,-1,0),(1,0,-1)\}\\
A_{653} &= \{(-1,0,1),(-1,1,0),(0,0,0),(0,1,-1),(1,-1,0),(1,0,-1)\}\\
A_{1157} &= \{(-1,1,0),(0,-1,1),(0,0,0),(0,1,-1),(1,-1,0),(1,0,-1)\}\\
A_{1201} &= \{(-1,0,1),(-1,1,0),(0,-1,1),(0,0,0),(0,1,-1),(1,-1,0)\}\\
A_{3914} &= \{(-1,0,1),(-1,1,0),(0,-1,1),(0,0,0),(0,1,-1),(1,-1,0),(1,0,-1)\}
\end{aligned}
$$

For example, having computed $A_0, \ldots, A_{1201}$, to find the next and final term we can use the following `Walnut` code:

```
def last "?msd_trib ~($subset(n,0) | $subset(n,1) | $subset(n,2)
| $subset(n,3) | $subset(n,4) | $subset(n,5) | $subset(n,6)
| $subset(n,9) | $subset(n,11) | $subset(n,17) | $subset(n,30)
| $subset(n,31) | $subset(n,55) | $subset(n,57) | $subset(n,101)
| $subset(n,105) | $subset(n,185) | $subset(n,340) | $subset(n,341)
```

```
| $subset(n,342) | $subset(n,355) | $subset(n,629) | $subset(n,653)
| $subset(n,1157) | $subset(n,1201))":
def missing "?msd_trib $last(n) & Am (m<n) => ~$last(m)":
```

Then the Tribonacci automaton computed by `Walnut` and stored in `missing.txt`, depicted in Figure 7, accepts exactly one word, which is 10011000000000, representing 3914 in Tribonacci representation. So we know the next term is A_{3914}.

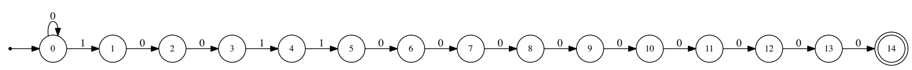

Figure 7: The next term is 3914.

This proves part (b) of Theorem 2.

Now we can use the direct product construction mentioned above in Section 2 to obtain two DFAO's, which differ only in output function. The transition function δ and the two output functions τ_1 and τ_2 are given in Table 1. The DFAO with output function τ_1 computes, for each n, the minimal n' such that $A_n = A_{n'}$; it is called `TRAS.txt`. The DFAO with output function τ_2 computes, for each n, the value of $\rho^{\text{ab}}_{\mathbf{TR}}(n)$; it is called `TRAC.txt`. This proves parts (d), (e), and (f) of Theorem 2.

The reason why the second automaton given in Table 1 differs from that of Turek [20] is that our automaton detects illegal Tribonacci representations (in state 7) and produces an output -1 for those representations, while Turek's automaton does not. We have checked the output of Turek's automaton against ours for the first 1,000,000 terms and they agree in all cases. This serves as a double-check on our results.

Once we have the automaton, additional basic results such as Turek's Corollary 7.2 of [19] (asserting that $\rho^{\text{ab}}_{\mathbf{TR}}(n) = 4$ for infinitely many n) follow immediately. It can also be easily obtained with `Walnut` as follows:

```
eval test4 "?msd_trib An Em (m>n) & TRAC[m]=@4":
```

which evaluates to `true`. In fact, the analogous result holds for each of the abelian complexities $3, 4, 5, 6, 7$.

Finally, part (c) of Theorem 2 can be verified with a similar `Walnut` computation. For example, for $i = 3914$ we can execute the command

```
eval testa3914 "?msd_trib An Em (m>n) & TRAS[m]=@3914":
```

which evaluates to `true`.

All the `Walnut` code referred to in this paper is available from

https://cs.uwaterloo.ca/~shallit/papers.html

q	$\delta(q,0)$	$\delta(q,1)$	$\tau_1(q)$	$\tau_2(q)$	q	$\delta(q,0)$	$\delta(q,1)$	$\tau_1(q)$	$\tau_2(q)$	q	$\delta(q,0)$	$\delta(q,1)$	$\tau_1(q)$	$\tau_2(q)$
0	0	1	0	1	26	31	19	55	5	52	57	24	629	6
1	2	3	1	3	27	29	32	5	4	53	58	7	3	4
2	4	5	2	3	28	33	5	57	5	54	59	39	653	6
3	6	7	3	4	29	16	20	9	4	55	60	61	5	4
4	8	9	4	3	30	34	35	6	4	56	41	62	185	5
5	10	11	5	4	31	36	24	101	5	57	63	46	1157	6
6	12	5	6	4	32	37	7	3	4	58	64	65	30	5
7	7	7	-1	-1	33	38	39	105	5	59	66	46	1201	6
8	6	13	3	4	34	18	40	11	4	60	49	67	9	4
9	14	15	1	3	35	41	42	185	5	61	68	7	341	5
10	16	5	9	4	36	43	27	30	5	62	69	7	341	5
11	17	7	3	4	37	26	44	30	5	63	70	46	342	6
12	18	19	11	4	38	45	46	31	5	64	31	40	55	5
13	6	3	3	4	39	47	15	31	5	65	71	72	185	5
14	4	20	2	3	40	30	48	3	4	66	73	27	355	6
15	21	7	3	4	41	49	50	340	5	67	31	25	55	5
16	22	23	17	4	42	51	7	341	5	68	26	74	30	5
17	12	20	6	4	43	52	19	342	6	69	34	65	6	4
18	21	24	3	4	44	29	53	5	4	70	70	46	3914	7
19	17	15	3	4	45	54	27	355	6	71	49	67	340	5
20	10	25	5	4	46	47	32	31	5	72	58	7	341	5
21	26	27	30	5	47	33	20	57	5	73	70	39	3914	7
22	28	19	31	5	48	36	7	101	5	74	60	75	5	4
23	29	15	5	4	49	22	55	17	4	75	76	7	341	5
24	28	11	31	5	50	31	11	55	5	76	26	77	30	5
25	30	7	3	4	51	34	56	6	4	77	60	72	5	4

Table 1: The two DFAO's computing A_n and $\rho^{\mathrm{ab}}_{\mathbf{TR}}(n)$.

5. Conclusions

We have shown that in some cases the abelian complexity function of an automatic sequence can be "automatically" computed. This continues in the spirit of other papers (e.g., [6, 13]) that try to automate results in combinatorics on words that formerly needed extensive case analysis to prove. For example, exactly the same method can be used to compute the abelian complexity of (i) the Thue-Morse word $\mathbf{t} = 01101001\cdots$ [15] and (ii) the so-called ternary Thue-Morse word [11].

Acknowledgment. I am pleased to thank Ondřej Turek, Gwenaël Richomme, and the referee for their helpful comments.

References

[1] J.-P. Allouche and J. O. Shallit, *Automatic Sequences*, Cambridge University Press, 2003.

[2] A. R. Baranwal, Decision algorithms for Ostrowski-automatic sequences, M. Math. thesis, University of Waterloo, School of Computer Science, 2020. Available at https://uwspace.uwaterloo.ca/handle/10012/15845.

[3] E. Barcucci, L. Bélanger, and S. Brlek, On Tribonacci sequences, *Fibonacci Quart.* **42** (2004), 314–319.

[4] P. S. Bruckman, The generalized Zeckendorf theorems, *Fibonacci Quart.* **27** (1989), 338–347.

[5] A. Carpi and C. Maggi, On synchronized sequences and their separators, *RAIRO Inform. Théor. App.* **35** (2001), 513–524.

[6] E. Charlier, N. Rampersad, and J. Shallit, Enumeration and decidable properties of automatic sequences, in G. Mauri and A. Leporati, eds., *Developments in Language Theory, 15th International Conference, DLT 2011*, Vol. 6795 of *Lect. Notes in Comp. Sci.*, pp. 165–179, Springer-Verlag, 2011.

[7] N. Chekhova, P. Hubert, and A. Messaoudi, Propriétés combinatoires, ergodiques et arithmétiques de la substitution de Tribonacci, *J. Théorie Nombres Bordeaux* **13** (2001), 371–394.

[8] F. M. Dekking, J. Shallit, and N. J. A. Sloane, Queens in exile: non-attacking queens on infinite chess boards, *Elect. J. Combinatorics* **27** (1) (2020), #P1.52.

[9] C. Frougny, Fibonacci numeration systems and rational functions, in J. Gruska, B. Rovan, and J. Wiedermann, eds., *MFCS 86*, *Lect. Notes in Comp. Sci.*, Vol. 233, pp. 350–359. Springer-Verlag, 1986.

[10] D. Goč, L. Schaeffer, and J. Shallit, The subword complexity of k-automatic sequences is k-synchronized, in M.-P. Béal and O. Carton, eds., *DLT 2013*, Vol. 7907 of *Lecture Notes in Computer Science*, pp. 252–263. Springer-Verlag, 2013.

[11] I. Kaboré and B. Kientéga, Abelian complexity of Thue-Morse word over a ternary alphabet, in S. Brlek, F. Dolce, C. Reutenauer, and É. Vandomme, eds., *WORDS 2017*, Lecture Notes in Computer Science, pp. 132–143, Springer-Verlag, 2017.

[12] H. Mousavi, Automatic theorem proving in Walnut, preprint available at `https://arxiv.org/abs/1603.06017`, 2016. Software available at `https://github.com/hamousavi/Walnut` .

[13] H. Mousavi and J. Shallit, Mechanical proofs of properties of the Tribonacci word, in F. Manea and D. Nowotka, eds., *WORDS 2015*, Lect. Notes in Comp. Sci., Vol. 9304, pp. 170–190, Springer-Verlag, 2015.

[14] G. Richomme, K. Saari, and L. Q. Zamboni, Balance and Abelian complexity of the Tribonacci word, *Adv. Appl. Math.* **45** (2010), 212–231.

[15] G. Richomme, K. Saari, and L. Q. Zamboni, Abelian complexity of minimal subshifts, *J. London Math. Soc.* (2) **83** (2011), 79–95.

[16] S. W. Rosema and R. Tijdeman, The Tribonacci substitution, *INTEGERS* **5** (3) (2005), Paper #A13.

[17] L. Schaeffer, Deciding properties of automatic sequences, Master's thesis, University of Waterloo, School of Computer Science, 2013. Available at `https://cs.uwaterloo.ca/~shallit/thesisLukeSept4.pdf`.

[18] J. Shallit and R. Zarifi, Circular critical exponents for Thue-Morse factors, *RAIRO Theor. Inf. Appl.* **53** (2019), 37–49.

[19] O. Turek, Abelian complexity and abelian co-decomposition, *Theor. Comput. Sci.* **469** (2013), 77–91.

[20] O. Turek, Abelian complexity function of the Tribonacci word, *J. Integer Sequences* **18** (2015), Article 15.3.4.