SOME TRIBONACCI CONJECTURES

JEFFREY SHALLIT

ABSTRACT. In a recent talk of Robbert Fokkink, some conjectures related to the infinite Tribonacci word were stated by the speaker and the audience. In this note, we show how to prove (or disprove) the claims in a "purely mechanical" fashion, using the `Walnut` theorem-prover.

1. INTRODUCTION

The *Tribonacci sequence* $(T_n)_{n\geq 0}$ satisfies the three-term recurrence relation

$$T_n = T_{n-1} + T_{n-2} + T_{n-3}$$

for $n \geq 3$, with initial values $T_0 = 0$, $T_1 = 1$, $T_2 = 1$; see [6, 15]. It is sequence A000073 in the *On-Line Encyclopedia of Integer Sequences* (OEIS); see [13]. The first few values are listed in Table 1.

n	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
T_n	1	1	2	4	7	13	24	44	81	149	274	504	927	1705	3136

TABLE 1. First few values of the Tribonacci sequence.

As is well-known, every nonnegative integer N has a unique representation in the form

$$N = \sum_{2 \leq i \leq t} e_i T_i,$$

where $e_t = 1$, $e_i \in \{0, 1\}$, and $e_i e_{i+1} e_{i+2} \neq 1$ for $i \geq 2$ [3]. This representation is conventionally written as a binary word (or string) $(N)_T$ with the most significant digit first: $e_t e_{t-1} \cdots e_2$. For example, $(43)_T = 110110$. There is a related function $[x]_T = N$, if $x = x_1 \cdots x_t \in \{0, 1\}^*$ and $N = \sum_{1 \leq i \leq t} x_i T_{t+3-i}$.

Intimately connected with the Tribonacci sequence is the *infinite Tribonacci word*

$$\mathbf{TR} = t_0 t_1 t_2 \cdots = 0102010 \cdots ,$$

where t_N is defined to be the number of trailing 1s in $(N)_T$. This is a famous infinite word that has been extensively studied (e.g., [2, 18, 14, 9, 10]). It is sequence A080843 in the OEIS.

The Tribonacci sequence and its relatives are intimately connected with a number of combinatorial games [5]. In this note, I will explain how one can rigorously prove properties of these sequences "purely mechanically", using various computational techniques involving formal languages and automata. As an example of the method, we show how to prove (or disprove!) some recent conjectures made in a talk by Robbert Fokkink. One of these conjectures also appears just before Section 2 in [8].

We use the `Walnut` theorem-prover originally designed by Hamoon Mousavi [11]. This free software is able to rigorously prove or disprove first-order statements involving automatic and synchronized sequences. For previous work with `Walnut` and the infinite Tribonacci word, see [12].

2. Automata

Our methods involve two classes of infinite words, called Tribonacci-automatic and Tribonacci-synchronized. For more about them, see [12, 17].

An infinite word $\mathbf{x}$ is said to be *Tribonacci-automatic* if there is a deterministic finite automaton with output (DFAO) that, on input $[N]_T$, the Tribonacci representation of N, reaches a state with output $\mathbf{x}[N]$. For more about automatic sequences, see [1].

An infinite sequence of integers $(y_n)_{n \geq 0}$ is said to be *Tribonacci-synchronized* if there is a deterministic finite automaton (DFA) that, on input the Tribonacci representations of N and x in parallel (the shorter one, if necessary, padded with leading zeros), accepts if and only if $x = y_N$. For more about synchronized sequences, see [16].

The Tribonacci word $\mathbf{TR}$ is Tribonacci-automatic, and is generated by the automaton in Figure 1, where the output of the state numbered i is i itself.
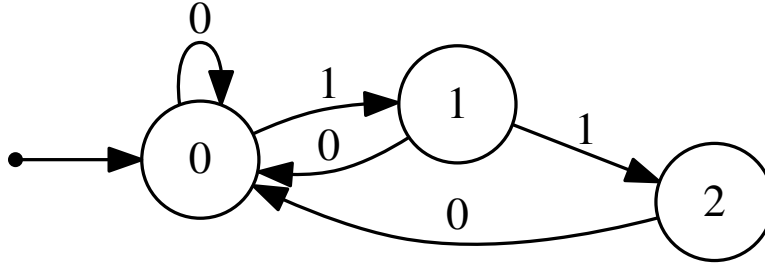

FIGURE 1. Tribonacci automaton computing $\mathbf{TR}$.

Three examples of Tribonacci-synchronized sequences are the sequences $(D_n)_{n \geq 0}$, $(E_n)_{n \geq 0}$, $(F_n)_{n \geq 0}$ defined as follows:

$$D_n = |\mathbf{TR}[0..n-1]|_0$$
$$E_n = |\mathbf{TR}[0..n-1]|_1$$
$$F_n = |\mathbf{TR}[0..n-1]|_2.$$

Tribonacci automata for these sequences are given in [17, Section 10.12]. These are sequences A276796, A276797, and A276798−1 in the OEIS, respectively.

3. Two Sequences Related to Combinatorial Games

Our starting point is two sequences related to combinatorial games. These are sequences A140100 and A140101 in the OEIS, from which we quote the definition more-or-less verbatim:

Definition 3.1. *Start with $X(0) = 0$, $Y(0) = 0$, $X(1) = 1$, $Y(1) = 2$. For $n > 1$, choose the least positive integers $Y(n) > X(n)$ such that neither $Y(n)$ nor $X(n)$ appear in $\{Y(k) : 1 \leq k < n\}$ or $\{X(k) : 1 \leq k < n\}$ and such that $Y(n) - X(n)$ does not appear in $\{Y(k) - X(k) : 1 \leq k < n\}$ or $\{Y(k) + X(k) : 1 \leq k < n\}$.*

Table 2 gives the first few terms of these sequences.

n	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$X(n)$	0	1	3	4	6	7	9	10	12	14	15	17	18	20	21	23
$Y(n)$	0	2	5	8	11	13	16	19	22	25	28	31	33	36	39	42

TABLE 2. First few values of the sequences $(X(k))_{k \geq 0}$ and $(Y(k))_{k \geq 0}$.

These two sequences were apparently first studied by Duchêne and Rigo [5, Corollary 3.6] in connection with a combinatorial game, and they established the close connection between them and the infinite Tribonacci word **TR**. Further properties were given in [4].

Because of the already-established connection with **TR**, one might conjecture that the sequences $\mathbf{X} = (X(k))_{k \geq 0}$ and $\mathbf{Y} = (Y(k))_{k \geq 0}$ are both Tribonacci-synchronized. But how can this be verified? There is no known method to take an arbitrary recursive description of sequences, like the one given in Definition 3.1, and turn it into automata computing the sequences.

We do so in two steps. The first is to use some heuristics to "guess" the automata for **X** and **Y**. We do that using (essentially) the classical Myhill-Nerode theorem, as described below for an arbitrary sequence of natural numbers.

3.1. **Guessing an Automaton.** Let $(x_n)_{n \geq 0}$ be a sequence of natural numbers that we suspect is Tribonacci-synchronized. The following heuristic procedure guesses a synchronized automaton for it. Let $A = (\{0,1\} \times \{0,1\})^*$. For two words $w, z \in \{0,1\}^*$ of equal length, we define $w \times z$ to be the word in A whose projection onto the first coordinate is w and second is z.

Define the language $L \subseteq A$ as follows:

$$L = \{w \times z \ : \ |w| = |z|, \ w, z \in \{0,1\}^*, \ \exists n \ [w]_T = n, \ [z]_T = x_n\}.$$

Furthermore, define an equivalence relation on A as follows: $r \equiv_i s$ if and only if $rt \in L$ if and only if $st \in L$ for all $t \in A$ with $|t| \leq i$. Clearly this equivalence relation partitions A into finitely many equivalence classes.

Now, do a breadth-first search on the elements of A in increasing order of length, where a word r is marked as "do not expand further" if it is equivalent (under $\equiv_i$) to a word that was previously examined. In practice, this is done by maintaining a queue of vertices to consider. When the queue becomes empty, an automaton can be formed by letting the states be the set of vertices, and a transition from r to s marked a if $ra \equiv_i s$. A state corresponding to r is accepting if $r \in L$.

One can then look at the resulting automata that are created for increasing i. If they stabilize (that is, the automata are the same for two consecutive values of i), we can surmise that we have found a synchronizing automaton. The correctness of the guessed automaton then needs to be verified rigorously by some different approach.

Remark 3.2. In place of the language L described above, one could also use an alternative language L', where one also insists that neither the projections of r nor s contain three consecutive 1s. It is not always clear which approach will result in smaller automata, and either one can be used with `Walnut`. In fact, this is the one we actually used.

When we carry out this procedure for the sequences **X** and **Y** mentioned above, we find Tribonacci-synchronized automata of 27 and 30 states, respectively. In `Walnut`, we call them

xaut and yaut, and store them in the `Automaton Library` directory of `Walnut`. If the reader wishes to verify the computations in this paper, the two automaton files can be downloaded from the author's web page:

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

All the other `Walnut` commands can be cut-and-pasted into `Walnut` to verify our claims.

3.2. **Verifying the Automata.** Now that we have candidate automata, it remains to verify their correctness. We can do this using mathematical induction. We say that a triple (n, x, y) is *good* if all of the following conditions hold:

(1) $y > x$;
(2) $x \notin \{X(k) : 1 \leq k < n\}$
(3) $y \notin \{X(k) : 1 \leq k < n\}$
(4) $x \notin \{Y(k) : 1 \leq k < n\}$
(5) $y \notin \{Y(k) : 1 \leq k < n\}$
(6) $y - x \notin \{Y(k) - X(k) : 1 \leq k < n\}$
(7) $y - x \notin \{Y(k) + X(k) : 1 \leq k < n\}$

To carry out the induction proof, we must show three things:

(1) The triple $(n, X(n), Y(n))$ is good for all $n \geq 1$;
(2) If (n, x, y) is good, then $x \geq X(n)$;
(3) If $(n, X(n), y)$ is good, then $y \geq Y(n)$.

The latter two conditions ensure that each value of $X(n)$ and $Y(n)$ chosen iteratively in Definition 3.1 is indeed the minimal possible value among good candidates.

This verification can be carried out by the following `Walnut` code. To interpret it, you need to know the following basics of `Walnut` syntax:

- `E` is the existential quantifier, and `A` is the universal quantifier.
- `?msd_trib` tells `Walnut` to represent integers in the Tribonacci representation system.
- `~` is logical NOT, `&` is logical AND, `|` is logical OR, `=>` is logical implication.
- `def` defines an automaton accepting the values of the free variables making the logical statement true.
- `eval` evaluates a logical statement with no free variables as `TRUE` or `FALSE`.

Now, we can create `Walnut` predicates to verify the claims. The code for `good` asserts that its arguments (n, x, y) represent a good triple. The three commands that follow it verify the three conditions.

```
def good "?msd_trib y>x &
   (~Ek k<n & $xaut(k,x)) &
   (~Ek k<n & $xaut(k,y)) &
   (~Ek k<n & $yaut(k,x)) &
   (~Ek k<n & $yaut(k,y)) &
   (~Ek,a,b k<n & $xaut(k,a) & $yaut(k,b) & y-x=b-a) &
   (~Ek,a,b k<n & $xaut(k,a) & $yaut(k,b) & y-x=b+a)":
eval check1 "?msd_trib An,x,y (n>=1 & $xaut(n,x) & $yaut(n,y)) => $good(n,x,y)":
eval check2 "?msd_trib An,x,y (n>=1 & $good(n,x,y)) => (Ez $xaut(n,z) & x>=z)":
eval check3 "?msd_trib An,x,y (n>=1 & $xaut(n,x) & $good(n,x,y)) =>
   (Ez $yaut(n,z) & y>=z)":
```

and the last three commands all return `TRUE`.

4. Verifying a Conjecture

Now that we have rigorously-proven automata for the sequences **X** and **Y**, we can use them to verify a conjecture of Robbert Fokkink, made in a talk he gave on October 3, 2022 [7]. It also appears just before Section 2 in the paper of Fokkink and Rust [8]. (Notice that they wrote a_n for $X(n)$ and b_n for $Y(n)$.)

Conjecture 4.1. For all $n \geq 1$, $X(Y(n)) = X(n) + Y(n)$ or $X(Y(n)) = X(n) + Y(n) - 1$.

Proof. We use the following `Walnut` code:

```
eval fokkink "?msd_trib An,x,y,xy (n>=1 & $xaut(n,x) & $yaut(n,y)
   & $xaut(y,xy)) => (xy=x+y|xy=x+y-1)":
```

and `Walnut` returns `TRUE`. □

We can also prove two conjectures of Julien Cassaigne, made during the same talk:

Conjecture 4.2.
   (1) For all $n \geq 1$, $X(Y(n)) = X(n) + Y(n)$ or $Y(X(n)) = X(n) + Y(n)$.
   (2) For all $n \geq 1$ we have $X(n) + Y(n) - Y(X(n)) \in \{0, 1, 2\}$.

Proof. We use `Walnut` as follows:

```
eval julien1 "?msd_trib An,x,y,xy,yx ($xaut(n,x) & $yaut(n,y) &
   $xaut(y,xy) & $yaut(x,yx)) => (xy=x+y|yx=x+y)":
eval julien2 "?msd_trib An,x,y,yx ($xaut(n,x) & $yaut(n,y) & $yaut(x,yx)) =>
   (x+y-yx=0|x+y-yx=1|x+y-yx=2)":
```

and `Walnut` returns `TRUE` for both. □

Similarly, we can refute a conjecture of Dan Rust, made in the same talk:

Conjecture 4.3. For all $n \geq 1$ we have $X(Y(n)) + Y(X(n)) = 2X(n) + 2Y(n) - 1$.

Disproof. We can *disprove* the conjecture as follows:

```
def rust "?msd_trib Ex,y,xy,yx $xaut(n,x) & $yaut(n,y) & $xaut(y,xy)
   & $yaut(x,yx) & xy+yx=2*x+2*y-1":
eval testrust1 "?msd_trib An (n>=1) => $rust(n)":
eval testrust2 "?msd_trib $rust(20)":
eval testrust3 "?msd_trib Am En (n>m & ~$rust(n))":
```

Here, `Walnut` returns, respectively, `FALSE`, `FALSE`, and `TRUE`. This shows that there are infinitely many counterexamples to Rust's conjecture, and one of them is $n = 20$. □

5. Other Assorted Properties

Let's prove that **X** and **Y** are complementary sequences.

Theorem 5.1. *The sets* $\{X(n) : n \geq 1\}$ *and* $\{Y(n) : n \geq 1\}$ *form a disjoint partition of* $\mathbb{N} = \{1, 2, 3, \ldots\}$.

Proof. We use the following `Walnut` code:

```
eval thm6 "?msd_trib (An (n>=1) => ((Ey $xaut(y,n))|(Ey $yaut(y,n))))
   & (An (n>=1) => (~Ey,z $xaut(y,n) & $yaut(z,n)))":
```

and `Walnut` returns `TRUE`. □

Similarly, let us prove that $(Y(n) - X(n))_{n \geq 1}$ and $(Y(n) + X(n))_{n \geq 1}$ are complementary sequences.

Theorem 5.2. *The sets $\{Y(n) - X(n) : n \geq 1\}$ and $\{Y(n) + X(n) : n \geq 1\}$ form a disjoint partition of $\mathbb{N} = \{1, 2, 3, \ldots\}$.*

Proof. We use the following `Walnut` code:

```
def diff "?msd_trib Ex,y $xaut(n,x) & $yaut(n,y) & t=y-x":
def sum "?msd_trib Ex,y $xaut(n,x) & $yaut(n,y) & t=x+y":
eval thm7 "?msd_trib (An (n>=1) => ((Ey $diff(y,n))|(Ey $sum(y,n))))
  & (An (n>=1) => (~Ey,z $diff(y,n) & $sum(z,n)))":
```

and `Walnut` returns TRUE. □

Define the sequence $a(n)$ (resp., $b(n)$, $c(n)$) to be one more than the position of the nth occurrence of the symbol 0 (resp., 1, 2) in **TR**. This is sequence [A003144](#) (resp., [A003145](#), [A003146](#)) in the OEIS. Here, the "first occurrence" means $n = 1$.

Theorem 5.3. *For $n \geq 1$ we have $Y(n) = a(n) + n$.*

Proof. We use the following `Walnut` code, where `triba` is a Tribonacci-synchronized automaton for $a(n)$.

```
reg shift {0,1} {0,1} "([0,0]|[0,1][1,1]*[1,0])*":
def triba "?msd_trib (s=0 & n=0) | Ex $shift(n-1,x) & s=x+1":
eval thm8 "?msd_trib An,x,y (n>=1 & $triba(n,x) & $yaut(n,y)) => y=x+n":
```

And `Walnut` returns TRUE. Here, the definitions of `shift` and `triba` come from [17]. □

Theorem 5.4. *For all $n \geq 0$, we have $X(n) = D_{n-1} + n$, where D_n is the sequence defined in Section [2](#).*

Proof. We use the following `Walnut` code, where `tribd` is a Tribonacci-synchronized automaton for D_n.

```
def tribd "?msd_trib Et,u $triba(s,t) & $triba(s+1,u) & t<=n & n<u":
eval thm9 "?msd_trib An,x,y (n>=1 & $tribd(n-1,x) & $xaut(n,y)) =>
  x+n=y":
```

And `Walnut` returns TRUE. Here, the definition of `tribd` comes from [17]. □

We can also find some relations between **X** and **Y** and the sequences a, b, c defined above.

Theorem 5.5. *We have*

    (a) $X(n) = b(n) - a(n)$ *for $n \geq 0$;*
    (b) $Y(n) = c(n) - b(n)$ *for $n \geq 0$.*

Proof. We use the `Walnut` commands:

```
def tribb "?msd_trib (s=0&n=0) | Ex,y $shift(n-1,x) &
  $shift(x,y) & s=y+2":
def tribc "?msd_trib (s=0&n=0) | Ex,y,z $shift(n-1,x) &
  $shift(x,y) & $shift(y,z) & s=z+4":
eval parta "?msd_trib An,a,b,x ($triba(n,a) & $tribb(n,b) & $xaut(n,x))
  => x=b-a":
eval partb "?msd_trib An,b,c,y ($tribb(n,b) & $tribc(n,c) & $yaut(n,y))
  => y=c-b":
```

and `Walnut` returns `TRUE` for both. □

Let $(\beta(n))_{n\geq 0}$ be the characteristic sequence of **X**, that is $\gamma(n) = 1$ if there exists i such that $n = X(i)$, and similarly let $(\gamma(n))_{n\geq 0}$ be the characteristic sequence of **Y**. Here, β is sequence [A305385](#) and γ is sequence [A305386](#) in the OEIS.

Theorem 5.6. *The sequences β and γ are Tribonacci-automatic.*

Proof. We can easily construct Tribonacci DFAOs generating these sequences, as follows:

```
def beta "?msd_trib Ei $xaut(i,n)":
combine BETA beta:
def gamma "?msd_trib Ei $yaut(i,n)":
combine GAMMA gamma:
```

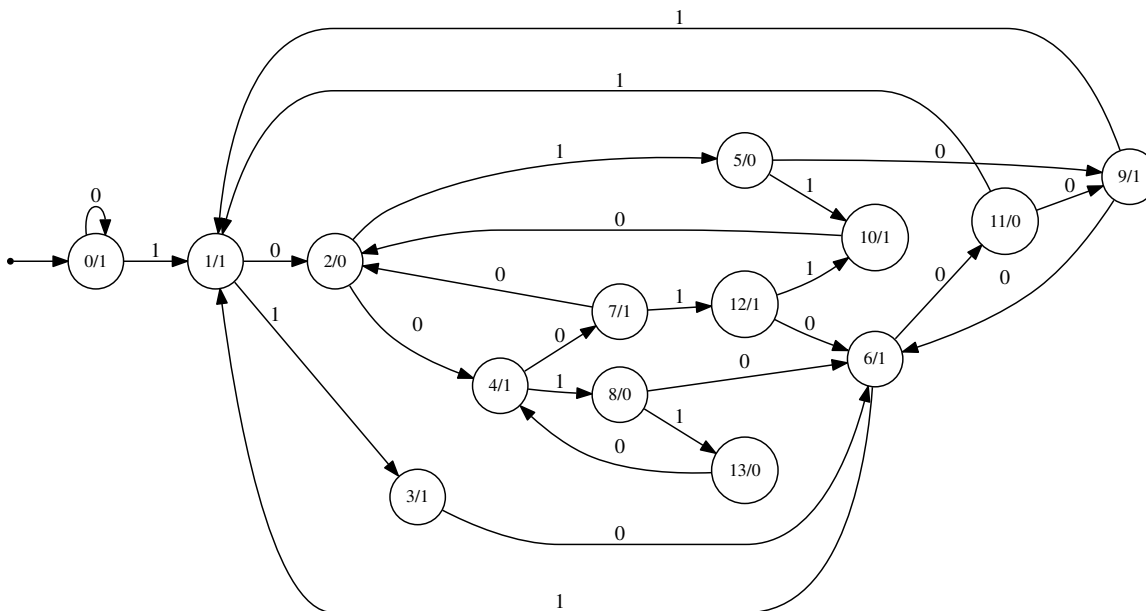The automaton for β is depicted in Figure 2.

FIGURE 2. Tribonacci DFAO computing β.

Here, the notation q/i in a state indicates that the state is numbered q and has output i. To get the automaton for γ, change the output of every state (except state 0) from 0 to 1 and vice versa. □

6. FINAL REMARKS

Nothing we have said in this note crucially depends on Tribonacci representation. Exactly the same approach often works for sequences defined in terms of Fibonacci representation, base-k representation, and some other kinds of representations such as Ostrowski representation. See [17] for a discussion of what kinds of representations work.

`Walnut` is available for free download at

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

`Walnut` has a number of limitations, and certainly cannot be used to automatically prove all properties of all sequences, but in its small domain it can be highly effective, as we hope to have shown here.

References

[1] J.-P. Allouche and J. Shallit, *Automatic Sequences: Theory, Applications, Generalizations*, Cambridge University Press, 2003.

[2] E. Barcucci, L. Bélanger, and S. Brlek, *On Tribonacci sequences*, The Fibonacci Quarterly, **42.4** (2004), 314–319.

[3] L. Carlitz, R. Scoville, and V. E. Hoggatt, Jr., *Fibonacci representations of higher order*, The Fibonacci Quarterly, **10.1** (1972), 43–69, 94.

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

[5] E. Duchêne and M. Rigo, *A morphic approach to combinatorial games: The Tribonacci case*, RAIRO Inform. Théor. App., **42** (2008), 375–393.

[6] M. Feinberg, *Fibonacci–Tribonacci*, The Fibonacci Quarterly, **1.3** (1963), 71–74.

[7] R. Fokkink, *A few words on games*, Talk for the One World Seminar on Combinatorics on Words, October 3 2022, available at https://tinyurl.com/mts7j2wh.

[8] R. Fokkink and D. Rust, *Queen reflections—a modification of Wythoff Nim*, Int. J. Game Theory, 2022, https://doi.org/10.1007/s00182-022-00824-1.

[9] Y. Huang and Z. Wen, *The numbers of repeated palindromes in the Fibonacci and Tribonacci words*, Disc. Appl. Math., **230** (2017), 78–90.

[10] M. Lejeune, M. Rigo, and M. Rosenfeld, *Templates for the k-binomial complexity of the Tribonacci word*, Adv. in Appl. Math., **112** (2020), 101947.

[11] H. Mousavi, *Automatic theorem proving in* `Walnut`, Arxiv preprint arXiv:1603.06017 [cs.FL], available at http://arxiv.org/abs/1603.06017, 2016.

[12] H. Mousavi and J. Shallit, *Mechanical proofs of properties of the Tribonacci word*, In: F. Manea and D. Nowotka, eds., *Proc. WORDS 2015*, Vol. 9304 of Lecture Notes in Computer Science, pp. 1–21, Springer-Verlag, 2015.

[13] OEIS Foundation Inc. (2022), The On-Line Encyclopedia of Integer Sequences, https://oeis.org.

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

[15] A. Scott, T. Delaney, and V. E. Hoggatt, Jr., *The Tribonacci sequence*, The Fibonacci Quarterly, **15.3** (1977), 193–200.

[16] J. Shallit, *Synchronized sequences*, In: T. Lecroq and S. Puzynina, eds., *WORDS 2021*, Vol. 12847 of Lecture Notes in Computer Science, pp. 1–19, Springer-Verlag, 2021.

[17] J. Shallit, *The Logical Approach to Automatic Sequences: Exploring Combinatorics on Words with* `Walnut`, Cambridge University Press, 2022.

[18] B. Tan and Z.-Y. Wen, *Some properties of the Tribonacci sequence*, European J. Combinatorics, **28** (2007), 1703–1719.

School of Computer Science, University of Waterloo, Waterloo, Ontario, N2L 3G1, Canada
Email address: shallit@uwaterloo.ca