RAIRO-Theor. Inf. Appl. 57 (2023) 4
https://doi.org/10.1051/ita/2022011

RAIRO - Theoretical Informatics and Applications
www.rairo-ita.org

AUTOMATIC SEQUENCES IN NEGATIVE BASES AND PROOFS OF SOME CONJECTURES OF SHEVELEV

Jeffrey Shallit[*,**], Sonja Linghui Shan and Kai Hsiang Yang

Abstract. We discuss the use of negative bases in automatic sequences. Recently the theorem-prover `Walnut` has been extended to allow the use of base $(-k)$ to express variables, thus permitting quantification over $\mathbb{Z}$ instead of $\mathbb{N}$. This enables us to prove results about two-sided (bi-infinite) automatic sequences. We first explain the theory behind negative bases in `Walnut`. Next, we use this new version of `Walnut` to give a very simple proof of a strengthened version of a theorem of Shevelev. We use our ideas to resolve two open problems of Shevelev from 2017. We also reprove a 2000 result of Shur involving bi-infinite binary words.

Mathematics Subject Classification. 68R15, 11B85, 11A63, 68Q45, 03D05.

Received September 6, 2022. Accepted December 22, 2022.

1. Introduction

`Walnut`, originally designed by Hamoon Mousavi [16], is a theorem-prover that can prove or disprove first-order logical statements about automatic sequences. Roughly speaking, automatic sequences $(a_n)_{n\geq 0}$ are those over a finite alphabet that can be computed by a DFAO (deterministic finite automaton with output) that, on input n represented in some fashion, has output a_n. The most famous example of such a sequence is the Thue-Morse sequence $\mathbf{t} = t_0 t_1 t_2 \cdots = 01101001\cdots$, defined by the relations $t_0 = 0$, $t_{2n} = t_n$, and $t_{2n+1} = 1 - t_n$ for $n \geq 0$; see [1], for example. In `Walnut` this is represented as a file `T.txt`, encoding a 2-state DFAO that computes the sequence for integer inputs represented in base 2. When referring to the Thue-Morse sequence at position n, `Walnut` uses the syntax `T[n]`.

As an example of the kind of statement that `Walnut` can prove, consider the pattern called an *overlap*: this is a word of the form $axaxa$, where a is a single letter and x is a possibly empty word, like the French word `entente`. Thue [6, 26] proved a celebrated theorem about the factors of $\mathbf{t}$ (*i.e.*, the contiguous blocks inside $\mathbf{t}$); namely, that $\mathbf{t}$ contains no factors that are overlaps. This theorem is clearly equivalent to the claim that there do not exist integers $i \geq 0$, $n \geq 1$ such that $t_{i+k} = t_{i+k+n}$ for $0 \leq k \leq n$. This condition can be restated as a first-order logical formula in the structure $\langle \mathbb{N}, +, <, n \to t_n \rangle$ as follows:

$$\neg \exists i, n \ (n \geq 1) \wedge \forall k \ (k \leq n) \implies t_{i+k} = t_{i+k+n}, \tag{1.1}$$

[*] Research funded by a grant from NSERC, 2018-04118.

Keywords and phrases: Automatic sequence, representation in negative base, Fibonacci representation, combinatorics on words, logic, decision procedure, Thue-Morse sequence.

School of Computer Science, University of Waterloo, Waterloo, ON N2L 3G1, Canada.

[**] Corresponding author: shallit@uwaterloo.ca

where, as usual, the symbol $\wedge$ denotes logical AND. When we enter this formula into `Walnut` in a suitably-translated form, *i.e.*,

```
eval has_no_overlap "~Ei,n (n>=1) & Ak (k<=n) => T[i+k]=T[i+k+n]":
```

then `Walnut` returns `TRUE`, thus rigorously proving the absence of overlaps in `t`. Here `E` is `Walnut`'s way of writing the existential quantifier, `A` is the universal quantifier, `=>` is logical implication, `&` is logical AND, and `~` is logical NOT.

The decision procedure used by `Walnut` compiles a first-order logical statement like equation (1.1) about an automatic sequence into a series of transformations on finite automata and DFAO's (deterministic finite automata with outputs on the states). Numbers are represented as words over a finite alphabet, in some numeration system such as base k. In order for the decision procedure to work, there must be finite automata checking the addition relation $x + y = z$ and the comparison relations $x < y$.

Up to now, the domain of variables in `Walnut` (such as the variables i, k, n appearing in equation (1.1)), has been restricted to $\mathbb{N} = \{0, 1, 2, \ldots\}$, the natural numbers. In this paper we describe a recent extension to `Walnut` that allows us to extend the domain of variables, in a simple and natural way, to $\mathbb{Z}$, the set of all integers. Among other things, this enables us to mechanically prove results about certain two-sided (bi-infinite) words (or sequences; we use these terms interchangeably). These words can be viewed as maps from $\mathbb{Z}$, the integers, to a finite alphabet. For more information about `Walnut` and its capabilities, see [20].

The contributions of this paper are twofold: first, we describe the additions to `Walnut` that make this extension possible, and their theoretical justification. Second, we use our extension of `Walnut` to improve a 2017 result of Shevelev, and resolve two of his (up to now unproved) open problems. We also reprove a 2000 result of Shur.

The outline of the paper is as follows. In Section 2 we recall the basic properties of representation in base $(-k)$. In Section 3 we discuss automatic sequences in base $(-k)$. In Section 4 we give the basic theoretical constructions that allow `Walnut` to work with base $(-k)$, and implementation details are given in Section 5. The syntax and semantics of the new `Walnut` commands are discussed in Section 6; as an application we reprove a 2000 result of Shur. In Section 7 we give a new proof of a result of Shevelev and we also strengthen it. In Section 8 we solve two of Shevelev's open problems from [23] using our new techniques. In Section 9 we consider the so-called negaFibonacci representation and use it to reprove a result of Levé and Richomme on the quasiperiods of the infinite Fibonacci word [14]. Finally, in Section 10 we prove (and strengthen) one more result of Shevelev.

2. Representation in base $(-k)$

Let $k \geq 2$ be an integer. In the late 19th century, Grünwald [9] introduced the idea of representing integers in base $(-k)$. The history, use, and application of negative bases is described in detail in [11, §4.1] and [2, §3.7].

In analogy with ordinary base-k representation, representation in base $(-k)$ involves writing

$$n = \sum_{0 \leq i \leq r} a_i (-k)^i,$$

where $n \in \mathbb{Z}$ and $a_i \in \Sigma_k$, where $\Sigma_k := \{0, 1, \ldots, k-1\}$. Disregarding leading zeros, every integer has a unique such base-$(-k)$ representation, called the *canonical expansion*, as a word $a_r a_{r-1} \cdots a_0$, with $a_r \neq 0$, over the alphabet Σ_k. We denote it as $(n)_{-k}$. Similarly, if $x \in \Sigma_k^*$ is a word, we let $[x]_{-k}$ be the integer represented by x interpreted in base $(-k)$.

Table 1 gives some examples of representation in base (-2)—also called *negabinary*—where representations are given with the most significant digit first. Here ϵ denotes the empty word.

The advantage to this particular representation of $\mathbb{Z}$ is that we do not need artificial conventions such as a sign bit to represent integers. It also avoids the ambiguity associated with representations of 0, where -0 and $+0$ would have the same meaning.

TABLE 1. Representation in base (-2).

n	$(n)_{-2}$	$(-n)_{-2}$
0	ϵ	ϵ
1	1	11
2	110	10
3	111	1101
4	100	1100
5	101	1111
6	11010	1110
7	11011	1001
8	11000	1000

3. Automata in base $(-k)$

The well-studied theory of automatic sequences (see, *e.g.*, [2]) extends seamlessly to negative bases. We say a bi-infinite sequence $(a_n)_{n\in\mathbb{Z}}$ is $(-k)$-*automatic* if there exists a DFAO $(Q, \Sigma_k, \Delta, \delta, q_0, \tau)$ where

- Q is a finite nonempty set of states;
- $\Sigma_k = \{0, 1, \ldots, k-1\}$;
- Δ is a finite output alphabet;
- $\delta : Q \times \Sigma \to Q$ is the transition function;
- $q_0 \in Q$ is the start state;
- and $\tau : Q \to \Delta$ is the output mapping,

such that $a(n) = \tau(\delta(q_0, x))$ for all integers n and words $x \in \Sigma^*$ such that $[x]_{-k} = n$ (even those with leading zeros).

Let us look at an example. The one-sided Thue-Morse sequence

$$\mathbf{t} = 0110100110010110\cdots$$

can be extended in two separate ways to a two-sided generalization: either

$$\mathbf{t}' = \mathbf{t}^R.\mathbf{t} = \cdots 100110010110.011010011001\cdots, \tag{3.1}$$

or

$$\mathbf{t}'' = \overline{\mathbf{t}}^R.\mathbf{t} = \cdots 011001101001.011010011001\cdots, \tag{3.2}$$

where an R as an exponent changes a one-sided right-infinite word into a one-sided left-infinite word, the overline denotes binary complement, and the period indicates that the 0 index begins immediately to the right. In Theorem 6.1, we show that these are natural generalizations.

Base-(-2) automata for these two two-sided sequences are given in Figure 1. Note that these automata are topologically identical; only the output functions associated with the states differ. We explain how to derive them in Section 6.5.

4. Components for working with base $(-k)$

In this section we describe the automaton components needed for working with base $(-k)$.

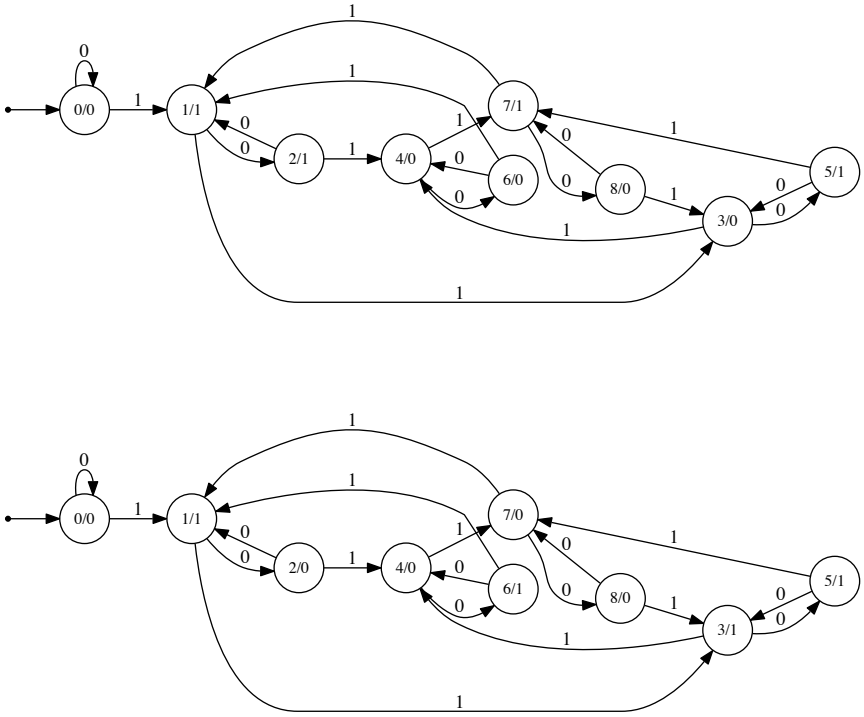

FIGURE 1. Base-(-2) automata generating the bi-infinite words $\mathbf{t}'$ (*top*) and $\mathbf{t}''$ (*bottom*).

4.1. Adder for base $(-k)$

Let $x, y, z \in \Sigma_k^*$. We first describe the construction of automata checking the relation $[x]_{-k} + [y]_{-k} = [z]_{-k}$ in base $(-k)$. Here the automaton reads the base-$(-k)$ representations in parallel, starting with the most significant digit. The representations are padded with leading zeros, if necessary, to make them all the same length.

Define a DFA $M_k = (Q, \Sigma_k \times \Sigma_k \times \Sigma_k, \delta, q_0, F)$ where

1. $Q = \{q_0, q_1, q_2, q_3\}$;
2. $F = \{q_0\}$; and
3. for all $[a, b, c] \in \Sigma_k \times \Sigma_k \times \Sigma_k$, we have

$$\delta(q_0, [a,b,c]) = \begin{cases} q_0, & \text{if } a+b-c = 0; \\ q_1, & \text{if } a+b-c = -1; \\ q_2, & \text{if } a+b-c = 1; \end{cases}$$
$$\delta(q_1, [0,0,k-1]) = q_2;$$
$$\delta(q_2, [a,b,c]) = \begin{cases} q_0, & \text{if } a+b-c-k = 0; \\ q_1, & \text{if } a+b-c-k = -1; \\ q_2, & \text{if } a+b-c-k = 1, \end{cases}$$

with undefined transitions all leading to q_3.

The DFA M_2 is given in Figure 2 with transitions to q_3 hidden. For $x, y, z \in \Sigma_k^*$ of length n, we interpret $[x, y, z]$

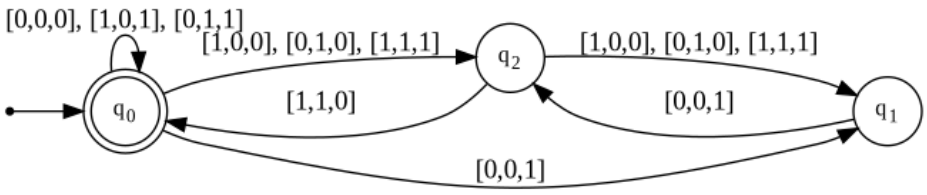

FIGURE 2. M_2.

as $[x_1, y_1, z_1][x_2, y_2, z_2] \cdots [x_n, y_n, z_n]$. To see that M_k recognizes the language $\{[x, y, z] : [x]_{-k} + [y]_{-k} = [z]_{-k}\}$, we have the following result.

Theorem 4.1. *Let $x, y, z \in \Sigma_k^*$ be words of length n. Let $q = \delta(q_0, [x, y, z])$.*

- *If $q = q_0$ then $[x]_{-k} + [y]_{-k} - [z]_{-k} = 0$.*
- *If $q = q_1$ then $[x]_{-k} + [y]_{-k} - [z]_{-k} = -1$.*
- *If $q = q_2$ then $[x]_{-k} + [y]_{-k} - [z]_{-k} = 1$.*
- *If $q = q_3$ then $\left|[x]_{-k} + [y]_{-k} - [z]_{-k}\right| > 1$.*

Proof. We proceed by induction on n.

- Suppose $n = 0$. Then $q = \delta(q_0, \varepsilon) = q_0$. Trivially, $[x]_{-k} + [y]_{-k} - [z]_{-k} = 0$.
- Suppose $n > 0$. Let $x = x'a, y = y'b, z = z'c$ for $x', y', z' \in \Sigma_k^*$ and $a, b, c \in \Sigma_k$. Suppose the claim holds for $[x', y', z']$. Let $q' = \delta(q_0, [x', y', z'])$. Then $q = \delta(q', [a, b, c])$.
  - Case $q' = q_0$. By the inductive hypothesis, $[x']_{-k} + [y']_{-k} - [z']_{-k} = 0$. Then

$$[x]_{-k} + [y]_{-k} - [z]_{-k} = [x'0]_{-k} + [y'0]_{-k} - [z'0]_{-k} + a + b - c = a + b - c.$$

    If $q = q_0, q_1, q_2$ or q_3, we have $a + b - c = 0, -1, 1$ or $|a + b - c| > 1$ respectively.
  - Case $q' = q_1$. By the inductive hypothesis, $[x']_{-k} + [y']_{-k} - [z']_{-k} = -1$. Then

$$[x]_{-k} + [y]_{-k} - [z]_{-k} = [x'0]_{-k} + [y'0]_{-k} - [z'0]_{-k} + a + b - c = k + a + b - c.$$

    If $q = q_2$, $a = b = 0$ and $c = k - 1$. So $k + a + b - c = 1$. If $q = q_3$, $a \neq 0$, $b \neq 0$ or $c \neq k - 1$. Suppose $|k + a + b - c| \leq 1$. Then $a + b - c \leq -k + 1$. It follows that $a = b = 0$ and $c = k - 1$, a contradiction. Hence, $|k + a + b - c| > 1$.
  - Case $q' = q_2$. By the inductive hypothesis, $[x']_{-k} + [y']_{-k} - [z']_{-k} = 1$. Then

$$[x]_{-k} + [y]_{-k} - [z]_{-k} = [x'0]_{-k} + [y'0]_{-k} - [z'0]_{-k} + a + b - c = a + b - c - k.$$

    If $q = q_0, q_1, q_2$ or q_3, we have $a + b - c - k = 0, -1, 1$ or $|a + b - c - k| > 1$ respectively.
  - Case $q' = q_3$. Since $q' = q_3$, $q = q_3$. By the inductive hypothesis, $\left|[x']_{-k} + [y']_{-k} - [z']_{-k}\right| > 1$. It follows that $\left|[x']_{-k} + [y']_{-k} - [z']_{-k}\right| \geq 2$ and then $\left|[x'0]_{-k} + [y'0]_{-k} - [z'0]_{-k}\right| \geq 2k$. Since $|a + b - c| \leq 2k - 2$,

$$\left|[x]_{-k} + [y]_{-k} - [z]_{-k}\right| = \left|[x'0]_{-k} + [y'0]_{-k} - [z'0]_{-k} + a + b - c\right| > 1$$

    In all cases, the claim holds for $[x, y, z]$.

This completes our proof by induction. □

By Theorem 4.1, we have M_k accepts $[x, y, z]$ if and only if $[x]_{-k} + [y]_{-k} = [z]_{-k}$.

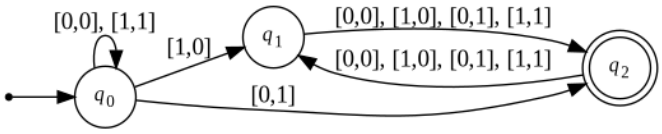

FIGURE 3. N_2.

4.2. Comparison automaton for base $(-k)$

In this section we describe the construction of an automaton for checking if $x < y$.
Define the DFA $N_k = (Q, \Sigma_k \times \Sigma_k, \delta, q_0, F)$ where

1. $Q = \{q_0, q_1, q_2\}$,
2. $F = \{q_2\}$, and
3. for all $[a, b] \in \Sigma_k \times \Sigma_k$, we have

$$\delta(q_0, [a,b]) = \begin{cases} q_0, & \text{if } a - b = 0; \\ q_1, & \text{if } a - b > 0; \\ q_2, & \text{if } a - b < 0; \end{cases}$$
$$\delta(q_1, [a,b]) = q_2$$
$$\delta(q_2, [a,b]) = q_1.$$

The DFA N_2 is given in Figure 3. For $x, y \in \Sigma_k^*$ of length n, we similarly interpret $[x, y]$ as $[x_1, y_1][x_2, y_2] \cdots [x_n, y_n]$. To see that N_k recognizes the language $\{[x, y] : [x]_{-k} < [y]_{-k}\}$, we have the following result.

Theorem 4.2. *Let $x, y \in \Sigma_k^*$ be words of length n. Let $q = \delta(q_0, [x, y])$.*

- *If $q = q_0$ then $[x]_{-k} - [y]_{-k} = 0$.*
- *If $q = q_1$ then $[x]_{-k} - [y]_{-k} > 0$.*
- *If $q = q_2$ then $[x]_{-k} - [y]_{-k} < 0$.*

Proof. We proceed by induction on n.

- Suppose $n = 0$. Then $q = \delta(q_0, \varepsilon) = q_0$. Trivially, $[x]_{-k} - [y]_{-k} = 0$.
- Suppose $n > 0$. Let $x = x'a, y = y'b$ for $x', y' \in \Sigma_k^*$ and $a, b \in \Sigma_k$. Suppose the result holds for $[x', y']$. Let $q' = \delta(q_0, [x', y'])$. Then $q = \delta(q', [a, b])$.
  - Case $q' = q_0$. By the inductive hypothesis, $[x']_{-k} - [y']_{-k} = 0$. Then

$$[x]_{-k} - [y]_{-k} = [x'0]_{-k} - [y'0]_{-k} + a - b = a - b.$$

    If $q = q_0, q_1$ or q_2, we have $a - b = 0$, $a - b > 0$ or $a - b < 0$ respectively.
  - Case $q' = q_1$. By the inductive hypothesis, $[x']_{-k} - [y']_{-k} > 0$. It follows that $[x']_{-k} - [y']_{-k} \geq 1$ and then $[x'0]_{-k} - [y'0]_{-k} \leq -k$. Then

$$[x]_{-k} - [y]_{-k} = [x'0]_{-k} - [y'0]_{-k} + a - b \leq a - b - k < 0,$$

    where in the last inequality we used the fact that $a - b < k$.

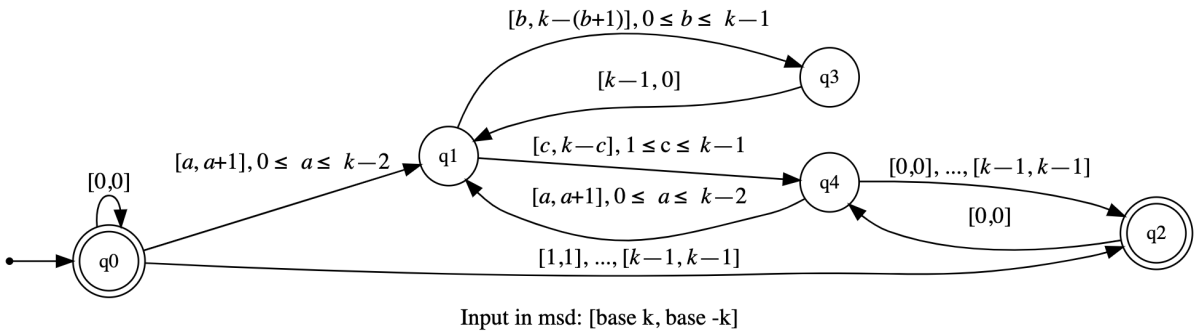

FIGURE 4. P_k.

- Case $q' = q_2$. By the inductive hypothesis, $[x']_{-k} - [y']_{-k} < 0$. It follows that $[x']_{-k} - [y']_{-k} \leq -1$ and thus $[x'0]_{-k} - [y'0]_{-k} \geq k$. Then

$$[x]_{-k} - [y]_{-k} = [x'0]_{-k} - [y'0]_{-k} + a - b \geq k + a - b > 0,$$

  where in the last inequality, we used the fact that $k + a - b > 0$.

In all cases, the result holds for $[x, y, z]$.

This completes our induction proof. □

By Theorem 4.2, we have N_k accepts $[x, y]$ if and only if $[x]_{-k} < [y]_{-k}$.

4.3. Conversion from base $(-k)$ to base k

In this section we describe an automaton that can be used to convert from base $(-k)$ to base k and vice versa. The theory behind this can be found in [2, §5.3].

We define a DFA $P_k = (Q, \Sigma_k \times \Sigma_k, \delta, q_0, F)$ as shown in Figure 4 where

- $Q = \{q_0, q_1, q_2, q_3, q_4\}$,
- $F = \{q_0, q_2\}$, and
- We define the following transitions

$$\begin{aligned}
&\delta(q_0, [0, 0]) = q_0,\\
&\delta(q_0, [a, a+1]) = q_1, 0 \leq a \leq k-2,\\
&\delta(q_0, [m, m]) = q_2, 1 \leq m \leq k-1,\\
&\delta(q_1, [b, k-(b+1)]) = q_3, 0 \leq b \leq k-1,\\
&\delta(q_1, [c, k-c]) = q_4, 1 \leq c \leq k-1,\\
&\delta(q_2, [0, 0]) = q_4,\\
&\delta(q_3, [k-1, 0]) = q_1,\\
&\delta(q_4, [a, a+1]) = q_1, 0 \leq a \leq k-2,\\
&\delta(q_4, [n, n]) = q_2, 0 \leq n \leq k-1.
\end{aligned}$$

Theorem 4.3. *Let $x, y \in \Sigma_k^*$ be words of length n. Let $q = \delta(q_0, [x, y])$.*

- *If $q = q_0$ then $[x]_k = [y]_{-k} = 0$.*
- *If $q = q_1$ then $x, y > 0$ and $[x]_k - [y]_{-k} = -1$.*

- *If* $q = q_2$ *then* $x, y > 0$ *and* $[x]_k - [y]_{-k} = 0$.
- *If* $q = q_3$ *then* $x > 0$, $y < 0$, *and* $[x]_k + [y]_{-k} = -1$.
- *If* $q = q_4$ *then* $x > 0$, $y < 0$, *and* $[x]_k + [y]_{-k} = 0$.

Proof. We proceed by induction on n.

Let $n = 0$, then $\delta(q_0, \epsilon) = q_0$ and we have $[x]_k = [y]_{-k} = 0$.

Let $n > 0$. Let $x = x'a$ and $y = y'b$, where $x', y' \in \Sigma_k^*$ and $a, b \in \Sigma_k$. Let $q' = \delta(q_0, [x', y'])$ and $q = \delta(q_0, [x, y]) = \delta(q', [a, b])$. Consider the following cases:

- If $q = q_1$, then by the construction of P_k, n is odd and $x, y > 0$. We consider the following cases for q':
  - If $q' = q_0$ or q_4 then $[x']_k + [y']_{-k} = 0$ by the induction hypothesis and $[a]_k - [b]_{-k} = -1$ by the construction of P_k. Then
$$[x]_k - [y]_{-k} = k \cdot ([x']_k + [y']_{-k}) + [a]_k - [b]_{-k} = [a]_k - [b]_{-k} = -1.$$
  - If $q' = q_3$ then $[x']_k + [y']_{-k} = -1$ by the induction hypothesis and $[a]_k - [b]_{-k} = k - 1$ by the construction of P_k. Then
$$[x]_k - [y]_{-k} = k \cdot ([x']_k + [y']_{-k}) + [a]_k - [b]_{-k} = -k + k - 1 = -1.$$
- If $q = q_2$, then by the construction of P_k, n is odd and $x, y > 0$. For both $q' = q_0$ and $q' = q_4$, $[x']_k + [y']_{-k} = 0$ by the induction hypothesis and $[a]_k - [b]_{-k} = 0$ by the construction of P_k. Then
$$[x]_k - [y]_{-k} = k \cdot ([x']_k + [y']_{-k}) + [a]_k - [b]_{-k} = 0.$$
- If $q = q_3$, then by the construction of P_k, n is even and $x > 0$ whereas $y < 0$. Since $q' = q_1$, $[x']_k - [y']_{-k} = -1$ by the induction hypothesis and $[a]_k + [b]_{-k} = k - 1$ by the construction of P_k. Then
$$[x]_k + [y]_{-k} = k \cdot ([x']_k - [y']_{-k}) + [a]_k + [b]_{-k} = -k + k - 1 = -1.$$
- If $q = q_4$, then by the construction of P_k, n is even and $x > 0$ whereas $y < 0$. We consider the following cases for q':
  - If $q' = q_1$ then $[x']_k - [y']_{-k} = -1$ by the induction hypothesis and $[a]_k + [b]_{-k} = k$ by the construction of P_k. Then
$$[x]_k + [y]_{-k} = k \cdot ([x']_k - [y']_{-k}) + [a]_k + [b]_{-k} = -k + k = 0.$$
  - If $q' = q_2$ then $[x']_k - [y']_{-k} = 0$ by the induction hypothesis and $[a]_k + [b]_{-k} = 0$ by the construction of P_k. Then
$$[x]_k + [y]_{-k} = k \cdot ([x']_k - [y']_{-k}) + [a]_k + [b]_{-k} = 0.$$

□

5. Extending `Walnut` to base $(-k)$

In this section, we describe how to extend `Walnut` to base $(-k)$. Suppose $x, y, z \in \mathbb{Z}$ are written in base $(-k)$. Recall from Section 4.1 that we can check the relations $x + y = z$ and $x < y$ using the base-$(-k)$ automata discussed there. These observations allow extending `Walnut` to base $(-k)$. Specifically, the comparison automaton N_k allows checking relations involving the relational operators, $=, \neq, <, >, \leq$, and $\geq$, available in `Walnut`. Moreover, the handling of logical operators, quantification, and indexing into automata does not need to be

changed to work specifically with base $(-k)$. It remains to discuss how `Walnut` can be extended to handle arithmetic operators in base $(-k)$.

`Walnut` for base $(-k)$ is available at

https://github.com/jono1202/Walnut.

5.1. Constants and negation in base $(-k)$

The relation $x = 0$ may be checked by the automaton corresponding to $\mathtt{0}^*$. Similarly, every representation matching $\mathtt{0}^*\mathtt{1}$ represents the integer 1. So the relation $x = 1$ may be checked by the automaton corresponding to $\mathtt{0}^*\mathtt{1}$. We now describe how to check $x = m$ for any $m \in \mathbb{Z}$. First, when $m = 2^n$ for $n \geq 0$, the relation $x = m$ can be rewritten using n additions. For example, $x = 8$ can be rewritten as

$$\exists y, z, w \in \mathbb{Z}\ (y + y = x) \wedge (z + z = y) \wedge (w + w = z) \wedge (w = 1).$$

Then for any $m > 0$, the relation $x = m$ can be rewritten using at most $\lfloor \log_2 m \rfloor - 1$ additions by considering the binary representation of m. For example, $x = 11 = 2^3 + 2^1 + 2^0$ can be rewritten as

$$\exists y, z, w, u \in \mathbb{Z}\ (x = y + z) \wedge (z = w + u) \wedge (y = 2^3) \wedge (w = 2^1) \wedge (u = 2^0).$$

Alternatively, when $m < 0$, the relation $x = m$ can be rewritten as

$$\exists y, z \in \mathbb{Z}\ (x + y = z) \wedge (y = -m) \wedge (z = 0).$$

This allows checking the relation $x = m$ for all $m \in \mathbb{Z}$. This last technique can be applied more generally; if $p(x_1, \ldots, x_m)$ is an arithmetic expression involving variables $x_1, \ldots, x_m$, the relation $x = -p(x_1, \ldots, x_m)$ can be rewritten as

$$\exists y, z \in \mathbb{Z}\ (x + y = z) \wedge (y = p(x_1, \ldots, x_m)) \wedge (z = 0).$$

In this way, we also have the ability to negate any arithmetic expression.

5.2. Arithmetic expressions in base $(-k)$

In this section, we show how addition, subtraction, multiplication by a constant, and division by a nonzero constant can be done in base $(-k)$. The ability to check relations $x = m$ for all $m \in \mathbb{Z}$, allows checking addition or subtraction, possibly involving integer constants. For example, $m - x = y$ can be rewritten as

$$\exists z\ (x + y = z) \wedge (z = m).$$

When $m \geq 0$, the relation $x = my$ can be handled in the same way as in non-negative bases. When $m < 0$, my can be rewritten as the negation of the arithmetic expression $(-m)y$. Specifically, we rewrite $x = my$ as $x = -((-m)y)$. This allows checking the relation $x = my$ for all $m \in \mathbb{Z}$. Finally, we need to be able to check the relation $x = \lfloor y/m \rfloor$ for nonzero integers m. When $m < 0$, we rewrite $x = \lfloor y/m \rfloor$ as

$$\exists r \in \mathbb{Z}\ (y = xm + r) \wedge (m < r) \wedge (r \leq 0).$$

Similarly, when $m > 0$, we rewrite $x = \lfloor y/m \rfloor$ as

$$\exists r \in \mathbb{Z}\ (y = xm + r) \wedge (0 \leq r) \wedge (r < m).$$

This allows checking the relation $x = \lfloor y/m \rfloor$ for nonzero integer constants m.

6. New `Walnut` commands

In this section, we describe the new split, reverse-split, and join commands in `Walnut` which will be useful for working in base $(-k)$.

`Walnut` with the split, reverse-split, and join commands is available at https://github.com/jono1202/Walnut.

6.1. Syntax of base $(-k)$ comands

Analogously to base k, the phrase `?msd_neg_k` preceding a formula specifies that integers in the formula are represented in base $(-k)$, with most significant digit first. Alternatively, the phrase `?lsd_neg_k` preceding a formula specifies that integers are represented in base $(-k)$, with least significant digit first. The unary negation operation is written using `_` (an underscore), and can be written preceding any arithmetic expression, variable, or constant.

One thing to keep in mind when using base $(-k)$ is that the universal and existential quantifiers now apply to all of $\mathbb{Z}$, not just $\mathbb{N}$.

6.2. The `split` command

The observation from Section 4.3 implies that if $(b_n)_{n\in\mathbb{Z}}$ is a $(-k)$-automatic sequence, then the sequence $(b_n)_{n\in\mathbb{N}}$ is k-automatic. Moreover, the base-k DFAO for $(b_n)_{n\in\mathbb{N}}$ can be computed using existing `Walnut` functionality.

As an example, suppose $(b_n)_{n\in\mathbb{Z}}$ is over the alphabet `{0,1}`, `B` is the base-$(-k)$ DFAO for $(b_n)_{n\in\mathbb{Z}}$, and `compare` is the conversion automaton from base k to base $(-k)$. Using existing `Walnut` commands, we may produce the base-k DFAO for $(b_n)_{n\in\mathbb{N}}$ as follows.

```
eval b "En B[n] = @1 & $compare(m,n)";
combine BS b;
```

The new `split` command is used to easily compute the same automaton, as follows.

```
split BS B[+];
```

The `split` command can also produce the automaton for $(b_{-n})_{n\in\mathbb{N}}$ and can be used on DFAO with any number of inputs. For example, given a base-$(-k)$ DFAO, `B2`, with two inputs, we may use `split` to transform `B2` as follows.

```
split B2S B2[+] [-];
```

`B2S` outputs on (x, y) in base k the same output as `B2` does on $(x, -y)$ in base $(-k)$.

6.3. The reverse-split command

The observation from Section 4.3 also implies that, given a k-automatic sequence $(c_n)_{n\in\mathbb{N}}$, the corresponding transformed sequence $(c_n)_{n\in\mathbb{Z}}$, where $c_n := 0$ if $n < 0$, is $(-k)$-automatic. The base-$(-k)$ DFAO for $(c_n)_{n\in\mathbb{Z}}$ can be computed using the new reverse-split command.

```
rsplit CR[+] C;
```

Also similar to the split command, the reverse-split command can produce the automaton for $(c_{-n})_{n\in\mathbb{Z}}$ and be used on DFAO with any number of inputs. For example, given a base-k DFAO, `C2`, with two inputs, we may use reverse-split as follows:

```
rsplit C2R[+] [-] C2;
```

When $x, -y \in \mathbb{N}$, `C2R` outputs on (x, y) in base $(-k)$ the same output as `C2` does on $(x, -y)$ in base k. Otherwise `C2R` outputs zero.

6.4. The `join` command

Given a list of DFAOs, the `join` command produces the DFAO which outputs the first nonzero value of the input DFAO. For example, given two base-k DFAOs, `B2` and `C2`, both with two inputs, the `join` command can be used as follows.

```
join BCJ B2[x] [y] C2[x] [y];
```

`BCJ` outputs on (x, y) the first nonzero output of `B2` or `C2` at (x, y). If both `B2` and `C2` output zero on (x, y), `BCJ` will output zero on (x, y).

6.5. Examples

As an illustration of these new `Walnut` commands, we show how to obtain the automata in Figure 1 that extend the one-sided Thue-Morse word $\mathbf{t}$ to the two bi-infinite sequences $\mathbf{t}'$ and $\mathbf{t}''$ previously introduced in Section 3.

```
def tmn "T[n-1]=@1":
combine T2 tmn:

rsplit T21 [+] T:
rsplit T22 [-] T2:
join TM21 T21[x] T22[x]:

def tmn2 "T[n-1]=@0":
combine T3 tmn2:

rsplit T23 [-] T3:
join TM22 T21[x] T23[x]:
```

Here we create the first automaton, `TM21`, by joining two automata, one for Thue-Morse on the non-negative integers, and one for Thue-Morse on the negative integers. The second automaton, `TM22`, is constructed similarly, except we use $\overline{\mathbf{t}}$ for the negative integers.

To see that these two infinite words $\mathbf{t}'$ and $\mathbf{t}''$ are good generalizations of $\mathbf{t}$ to the bi-infinite case, we can prove the following theorem.

Theorem 6.1. *A finite word x is a factor of $\mathbf{t}$ iff it is a factor of $\mathbf{t}'$, iff it is a factor of $\mathbf{t}''$.*

Proof. We can prove this with `Walnut` as follows:

```
def tm21faceq "?msd_neg_2 At (t>=0 & t<n) => TM21[i+t]=TM21[j+t]":
def tm22faceq "?msd_neg_2 At (t>=0 & t<n) => TM22[i+t]=TM22[j+t]":
eval tmtest1 "?msd_neg_2 Ai,n Ej (j>=0) & $tm21faceq(i,j,n)":
eval tmtest2 "?msd_neg_2 Ai,n Ej (j>=0) & $tm22faceq(i,j,n)":
```

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

We now use the automaton for $\mathbf{t}'$ to reprove a result of Shur [24], namely

Theorem 6.2. *There exists a bi-infinite binary word that avoids $(\frac{7}{3} + \epsilon)$-powers, and has unequal frequencies of letters.*

Let us recall the relevant definitions. We say a finite word $x = x[1..n]$ of length n has *period* p if $x[i] = x[i + p]$ for $1 \leq i \leq n - p$. Let $p > q \geq 1$ be integers. We say that a word x is a (p/q)-power if x is of length p and has period q. Let $\alpha > 1$ be a real number. We say an infinite word $\mathbf{x}$ is α-power-free if $\mathbf{x}$ has no finite factor that

is a (p/q)-power, for $p/q \geq \alpha$, and we say it is α^+-power-free if $\mathbf{x}$ has no finite factor that is a (p/q)-power, for $p/q > \alpha$. For example, an overlap-free word is 2^+-power-free.

Proof. The idea is to use the morphism given in [21, Theorem 33]; it was called g there, but we call it ξ here:

$$\xi(0) = 0110010011010010110100110 01$$
$$\xi(1) = 0110010011010010110011010 01$$
$$\xi(2) = 0110010011010011001011010 01$$

First we create a bi-infinite ternary word **vtm2** from the Thue-Morse word; this is a generalization of the so-called *variant Thue-Morse* word **vtm**. Then we apply ξ to it.

Here are additional details. The word

$$\mathbf{vtm} = v_0 v_1 v_2 \cdots = 2102012101202102012021 \cdots$$

is defined in terms of the Thue-Morse sequence by the relation

$$v_i = t_{i+1} - t_i + 1 \tag{6.1}$$

for $i \geq 0$; see, for example, [4]. We extend this to a bi-infinite word **vtm2** by using the relation (6.1) and applying it to the word in equation (3.1).

Now it is easy to see that the image of each letter under ξ has 14 0's and 13 1's, so the resulting word $\xi(\mathbf{vtm2})$ has unequal letter frequencies. We then assert that $\xi(\mathbf{vtm2})$ has a $(\frac{7}{3} + \epsilon)$-power with `Walnut`, as follows:

```
morphism xi "0 -> 0110010011010010110100110 01
1 -> 0110010011010010110011010 01
2 -> 0110010011010011001011010 01":
# Shur's 27-uniform morphism

def vtm0 "?msd_neg_2 TM21[n]+1=TM21[n-1]":
def vtm1 "?msd_neg_2 TM21[n]=TM21[n-1]":
def vtm2 "?msd_neg_2 TM21[n]=TM21[n-1]+1":
combine VTM2 vtm0=0 vtm1=1 vtm2=2:

image SHUR xi VTM2:
# apply xi to VTM2

eval testshur "?msd_neg_2 Ei,n (n>=1) & At (t>=i & 3*t<=3*i+4*n) =>
   SHUR[t]=SHUR[t+n]":
```

And `Walnut` returns `FALSE`, so the word is indeed $(\frac{7}{3} + \epsilon)$-free. □

This was a big calculation in `Walnut`, using 5468 seconds of CPU time and 400G of RAM.

7. Shevelev's first sequence

Now that we have extended `Walnut` to handle inputs with domain $\mathbb{Z}$ instead of $\mathbb{N}$, we are ready to solve Shevelev's problems on bi-infinite sequences.

Vladimir Shevelev [23] considered the following natural analogue of the Thue-Morse sequence for base -2: let $g(n)$ denote the number of 1's in the base-(-2) representation of n, taken modulo 2. Table 2 gives the first few values of g.

TABLE 2. Some values of g.

n	-7	-6	-5	-4	-3	-2	-1	0	1	2	3	4	5	6	7
$g(n)$	0	1	0	0	1	1	0	0	1	0	1	1	0	1	0

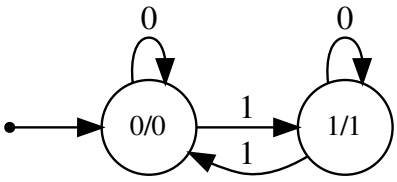

FIGURE 5. Automaton for Shevelev's g in base -2.

The (-2)-automaton for this sequence is given in Figure 5. For $n \geq 0$, this is sequence A269027 in the *On-Line Encyclopedia of Integer Sequences* (OEIS) [25].

Shevelev proved that the bi-infinite sequence $\mathbf{g} = (g(n))_{n\in\mathbb{Z}}$ contains no cubes, that is, three consecutive identical blocks of digits. His proof was quite complicated.

We can prove Shevelev's result and strengthen it as follows:

Theorem 7.1. *The sequence* $\mathbf{g}$ *contains no overlaps.*

Proof. We use the new version of `Walnut` that can use negative bases. We run the following command, which checks the assertion that there are overlaps. `Walnut` returns `FALSE`, so there are no overlaps.

```
eval shevelev "?msd_neg_2 Ei,n (n>=1) & At (t>=0 & t<=n) => G[i+t]=G[i+t+n]":
```

□

8. Shevelev's two open problems

In this section we show how, using `Walnut`, to prove Shevelev's two open problems, labeled (C) and (D) in his paper [23]. They concern two sequences v and w.

8.1. The $v(n)$ sequence

Shevelev also studied the following sequence: define $v(n)$ to be the largest integer k such that $g(i) = t(n+i)$ for $0 \leq i < k$, where $t(n)$ is the Thue-Morse sequence. This sequence is 2-synchronized in the sense of [19]; this means there is an automaton accepting precisely the base-2 representations of n and $v(n)$ in parallel, where we pad the shorter one with leading zeros.

We can construct this automaton with `Walnut` as follows: first, we extract the values of g on $\mathbb{N}$ using the new `split` command. It produces a new 2-automaton `SH` for $g(n)$. Of course `SH` can only compute $g(n)$ for non-negative values of n. We do it with the following command:

```
split SH G [+]:
```

This produces the automaton in Figure 6.

Shevelev studied the "record-setting" values of $v(n)$; these are the values $v(n)$ for which $v(n) > v(i)$ for all $i < n$. Shevelev defined the sequence of record-setting values as $(a(n))_{n\geq 1}$ and the position in which they occur in $v(n)$ as $(l(n))_{n\geq 1}$. Thus $v(l(n)) = a(n)$ for $n \geq 1$. The first few record-setting values are given in Table 3. Sequence $(a(n))_{n\geq 1}$ is A268866 in the OEIS.

The pairs $(a(n), l(n))$ are 2-synchronized. We can demonstrate this by creating an automaton accepting the pairs $(a(n), l(n))_2$ in parallel.

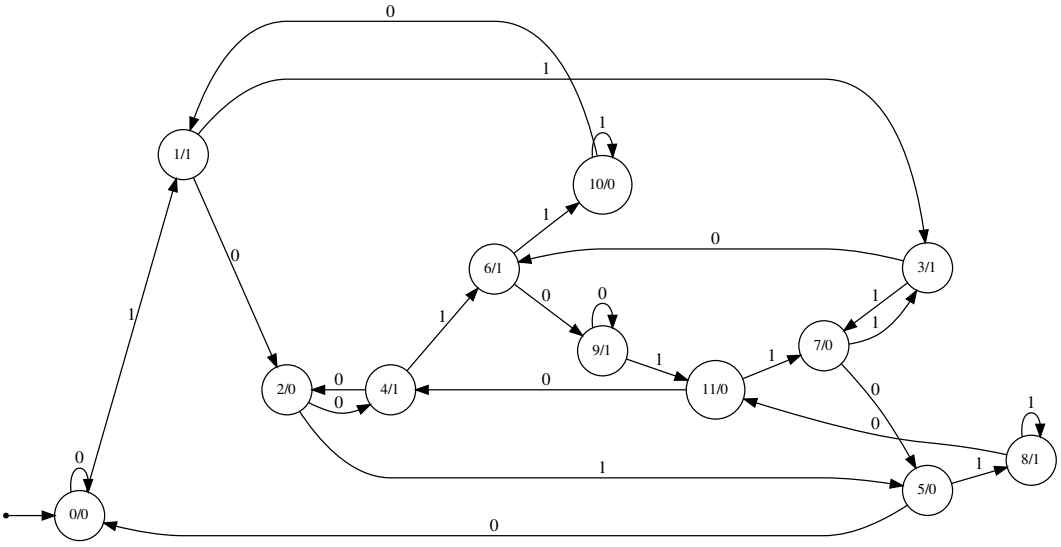

FIGURE 6. 2-automaton for $g(n)$.

TABLE 3. Record-setters for $v(n)$.

n	$a(n)$	$l(n)$
0	2	0
1	3	3
2	22	10
3	38	58
4	342	170
5	598	938

```
def agree "Ai (i<k) => SH[i]=T[n+i]":
def vseq "$agree(k,n) & ~$agree(k+1,n)":
def recordv "$vseq(k,n) & Aj,r (r<n & $vseq(j,r)) => (j<k)":
```

The resulting automaton `recordv` is displayed in Figure 7.

We can now examine the acceptance paths in this automaton. Ignoring leading zeros, they are of four types:

$$[1,0][0,0]$$
$$[1,1][1,1]$$
$$[1,0][0,1]([1,0][0,1][1,0][0,1])^i[1,0][1,1][0,0]$$
$$[1,1][0,1][0,1][1,0]([0,1][1,0][0,1][1,0])^i[1,1][0,0]$$

for $i \geq 0$, representing the numbers

$$(2,0)$$
$$(3,3)$$
$$(2^{4i+6}+2)/3, (2^{4i+5}-2)/3)$$
$$((4+7\cdot 2^{4i+5})/6, (11\cdot 2^{4i+5}-4)/6).$$

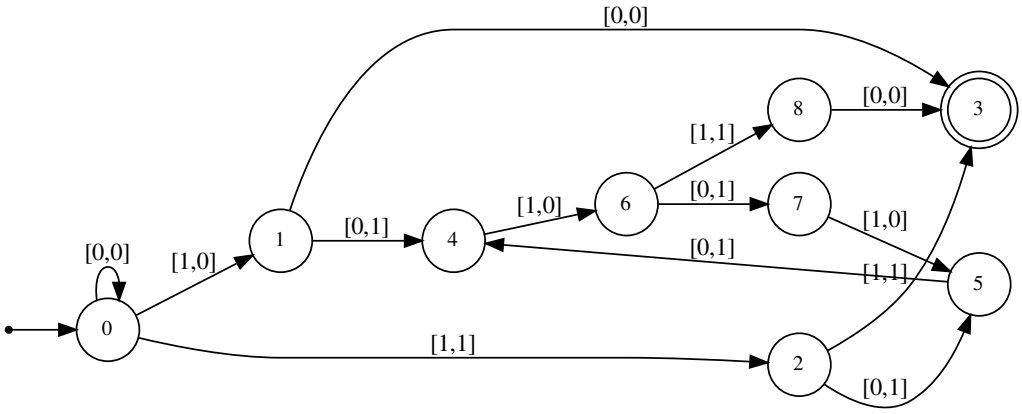

FIGURE 7. 2-automaton `recordv`.

Set $i = (n-2)/2$ for $n \geq 2$ even, and $i = (n-3)/2$ for $n \geq 3$ odd. Then from above we have

$$\begin{aligned}
\ell(0) &= 0 \\
\ell(1) &= 3 \\
\ell(n) &= (2^{2n+1} - 2)/3 \text{ for } n \geq 2 \text{ even, and} \\
\ell(n) &= (11 \cdot 2^{2n-1} - 4)/6 \text{ for } n \geq 3 \text{ odd.}
\end{aligned}$$

Similarly,

$$\begin{aligned}
a(0) &= 2 \\
a(1) &= 3 \\
a(n) &= (2^{2n+2} + 2)/3 \text{ for } n \geq 2 \text{ even, and} \\
a(n) &= (7 \cdot 2^{2n-1} + 4)/6 \text{ for } n \geq 3 \text{ odd.}
\end{aligned}$$

These statements are equivalent to the statement of open problem (C) in Shevelev's paper, which is now proved.

8.2. The $w(n)$ sequence

Shevelev also studied a "dual problem". Define $w(n)$ to be the largest integer k such that $g(i) \neq t(n+i)$ for $0 \leq i < k$, where again $t(n)$ is the Thue-Morse sequence. He then studied the "record-setting" values of $w(n)$; these are the values $w(n)$ such that $w(n) > w(i)$ for all $i < n$. Enumerate the sequence of record-setting values as $(b(n))_{n\geq 1}$ and the position where they occur in $v(n)$ as $(m(n))_{n\geq 1}$. The first few record-setting values are given in Table 4. Sequence $(b(n))_{n\geq 1}$ is sequence A269341 in the OEIS. We can find formulas for these numbers just as we did in Section 8.1.

```
def disagree "Ai (i<k) => SH[i]!=T[n+i]":
def wseq "$disagree(k,n) & ~$disagree(k+1,n)":
def recordw "$wseq(k,n) & Aj,r (r<n & $wseq(j,r)) => (j<k)":
```

The resulting automaton `recordw` is displayed in Figure 8.

We can now examine the acceptance paths in this automaton. Ignoring leading zeros, they are of four types:

$$[0,0]$$

TABLE 4. Record-setters for $w(n)$.

n	$b(n)$	$m(n)$
0	0	0
1	1	1
2	6	2
3	10	14
4	86	42
5	150	234

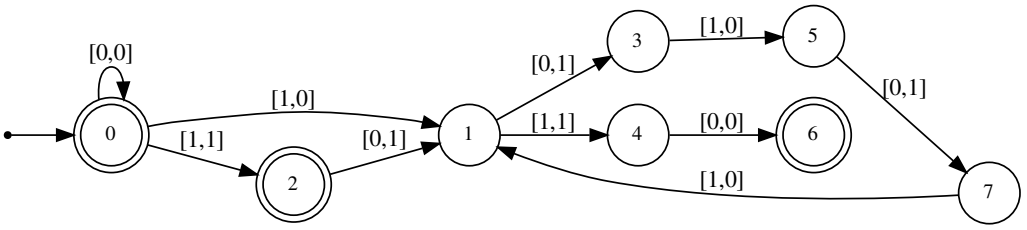

FIGURE 8. 2-automaton `recordw`.

$$
\begin{aligned}
&[1,1]\\
&[1,0]([0,1][1,0][0,1][1,0])^i[1,1][0,0]\\
&[1,1][0,1]([0,1][1,0][0,1][1,0])^i[1,1][0,0]
\end{aligned}
$$

for $i \geq 0$, representing (in binary) the pairs

$$
\begin{aligned}
&(0,0)\\
&(1,1)\\
&(2^{4i+4}+2)/3, (2^{4i+3}-2)/3)\\
&((7\cdot 2^{4i+3}+4)/6, (11\cdot 2^{4i+3}-4)/3).
\end{aligned}
$$

Set $i = (n-2)/2$ for $n \geq 2$ even, and $i = (n-3)/2$ for $n \geq 3$ odd. Then from above we have

$$
\begin{aligned}
&m(0) = 0\\
&m(1) = 1\\
&m(n) = (2^{2n-1}-2)/3 \text{ for } n \geq 2 \text{ even, and}\\
&m(n) = (11\cdot 2^{2n-3}-4)/6 \text{ for } n \geq 3 \text{ odd.}
\end{aligned}
$$

Similarly,

$$
\begin{aligned}
&b(0) = 0\\
&b(1) = 3\\
&b(n) = (2^{2n}+2)/3 \text{ for } n \geq 2 \text{ even, and}\\
&b(n) = (7\cdot 2^{2n-3}+4)/6 \text{ for } n \geq 3 \text{ odd.}
\end{aligned}
$$

These two formulas are equivalent to Shevelev's open problem (D), which is now proved.

9. NegaFibonacci representation

Previous sections have discussed representation in base-$(-k)$. In this section we discuss a more exotic numeration system, called negaFibonacci representation.

In the ordinary Fibonacci numeration system (also called the Zeckendorf numeration system), a natural number n is expressed as a sum

$$n = \sum_{2 \le i \le t} e_i F_i,$$

where $F_0 = 0$, $F_1 = 1$, $F_n = F_{n-1} + F_{n-2}$ are the Fibonacci numbers and $e_i \in \{0, 1\}$. Such a representation is unique (up to leading zeros) provided $e_i e_{i+1} \neq 1$ for $2 \le i < t$. The binary word $e_t e_{t-1} \cdots e_2$ is called the *canonical Fibonacci representation* for n and is denoted $(n)_F$. See, for example, [13, 27]. `Walnut` can do calculations with numbers represented in this numeration system; see [17] for some examples.

As an example, consider the infinite (one-sided) Fibonacci word

$$\mathbf{f} = f_0 f_1 f_2 \cdots = 0100101001001 \cdots,$$

which is the fixed point of the morphism $0 \to 01$, $1 \to 0$ [5]. It is known that for $i \ge 0$, f_i is equal to the least significant digit of the Fibonacci representation of i, and hence $\mathbf{f}$ is Fibonacci-automatic.

As is well-known, $\mathbf{f}$ is also a Sturmian characteristic word; more specifically, if we define

$$\mathbf{s}_\theta = s_\theta(1) s_\theta(2) s_\theta(3) \cdots$$

where $0 < \theta < 1$ is a real irrational number and

$$\mathbf{s}_\theta(n) = \lfloor (n+1)\theta \rfloor - \lfloor n\theta \rfloor,$$

then

$$f_i = s_\gamma(i+1) \tag{9.1}$$

for $\gamma = (3 - \sqrt{5})/2$ and $i \ge 0$.

We now consider an extension of the one-sided word $\mathbf{f}$ to a two-sided (bi-infinite) word. There are essentially two different ways to do this; see [8, 12, 18].

One natural extension of $\mathbf{f}$ to a two-sided or bi-infinite word is to use the equality (9.1) for *all* integers i, not just for $i \ge 0$. Setting $g_i = s_\gamma(i+1)$ for all $i \in \mathbb{Z}$ gives us the following infinite word:

$$\mathbf{g} = (g_i)_{i \in \mathbb{Z}} = \cdots 010100100101001010.01001010010010100 \cdots .$$

Notice that, from the definition, the words $\mathbf{f}$ and $\mathbf{g}$ coincide on non-negative indices.

On the other hand, the Fibonacci numeration system can be extended to a representation for all integers, not just non-negative integers. This extension, sometimes called the negaFibonacci system, expresses n as a sum

$$n = \sum_{1 \le i \le t} e_i (-1)^{i+1} F_i.$$

Bunder [7] proved that this is a unique representation for all integers (up to leading zeros), provided $e_i e_{i+1} \neq 1$. In this case the binary word $e_t e_{t-1} \cdots e_1$ is called the *canonical negaFibonacci representation* for n and is denoted $(n)_{-F}$. Table 9 gives some examples of this representation.

TABLE 5. Examples of negaFibonacci representation.

n	$(n)_{-F}$	$(-n)_{-F}$
0	ϵ	ϵ
1	1	10
2	100	1001
3	101	1000
4	10010	1010
5	10000	100101
6	10001	100100
7	10100	100001
8	10101	100000

An alternative extension of **f** to a two-sided word could then be defined as the least significant bit of the negaFibonacci representation of n: namely,

$$\mathbf{h} = (h_i)_{i\in\mathbb{Z}} = \cdots 00101001001010010.0101001010010010 10 \cdots .$$

Here the relationship between **f** and **h** is slightly less apparent.

Our goal is to show the relationship between the infinite words **f**, **g**, and **h**. Namely,

Theorem 9.1. *We have*

(a) $g_i = g_{-3-i}$ *for* $i \notin \{-2, -1\}$;
(b) $h_i = h_{1-i}$ *for* $i \notin \{0, 1\}$;
(c) $g_{i-1} = h_{i+1}$ *for* $i \notin \{-1, 0\}$;
(d) $g_i = h_{-1-i}$ *for all* i.

Proof. We start with the Fibonacci-synchronized automaton `phin` given in [19] for $(n, \lfloor \alpha n \rfloor)$, $n \geq 0$, where $\alpha = (1 + \sqrt{5})/2$.

From it we construct a Fibonacci-synchronized automaton `gn` for $(n, \lfloor \gamma n \rfloor)$ for $n \geq 0$. For $n \geq 1$ we have

$$\lfloor \gamma n \rfloor = \lfloor (2 - \alpha)n \rfloor = \lfloor 2n - \alpha n \rfloor = 2n + \lfloor -\alpha n \rfloor = 2n - 1 - \lfloor \alpha n \rfloor,$$

where we have used the fact that $\lfloor x \rfloor + \lfloor -x \rfloor = -1$ if x is not an integer.

We also construct a Fibonacci-synchronized automaton `gmn` for $(n, -\lfloor -\gamma n \rfloor)$ for $n \geq 0$.

Next, we convert these two Fibonacci automata into two negaFibonacci automata, one for the non-negative values of n and one for the negative values.

We then join the two negaFibonacci automata into one called `GG` for $(n, \lfloor n\gamma \rfloor)$ that covers the entire range of n: positive, negative, and zero.

From this we get another negaFibonacci automaton computing $\lfloor (n + 1)\gamma \rfloor - \lfloor n\gamma \rfloor$ for all $n \in \mathbb{Z}$.

We create a negaFibonacci DFAO G2 for g and H2 for h, and then verify the identities (a)–(d). Here is the `Walnut` code:

```
reg shift {0,1} {0,1} "([0,0]|[0,1][1,1]*[1,0])*":
def phin "?msd_fib (s=0 & n=0) | Ex $shift(n-1,x) & s=x+1":
# Fibonacci synchronized automaton for (n, floor(n*alpha))

def gn "?msd_fib (n>=1 & Ex $phin(n,x) & s+x+1=2*n) | (n=0&s=0)":
# Fibonacci synchronized automaton for (n, floor(n*gamma))
```

```
combine GN gn:
def gmn "?msd_fib (n=0&s=0) | (n>=1) & $gn(n,s-1)":
combine GMN gmn:

rsplit GN2 [+] [+] GN:
rsplit GMN2 [-] [-] GMN:

join GG GN2[x][y] GMN2[x][y]:
# GG[x][y] is negaFibonacci synchronized DFAO for (n, floor(n*gamma))

def st "?msd_neg_fib Ex,y GG[n][x]=@1 & GG[n+1][y]=@1 & y=x+1":
# st[n] returns true if sturmian word at position n
# in negaFibonacci representation is equal to 1

def fb "?msd_neg_fib $st(n+1)":
# n'th position of the Fibonacci word, generalized to negaFibonacci
combine G2 fb:

eval parta "?msd_neg_fib Ai (G2[i]=G2[_(i+3)]) <=> (i!=_1 & i!=_2)":

reg ends1 msd_neg_fib "0*(0|10)*1":
combine H2 ends1:
eval partb "?msd_neg_fib Ai (H2[i]=H2[1-i]) <=> (i!=0 & i!=1)":

eval partc "?msd_neg_fib Ai (G2[i-1]=H2[i+1]) <=> (i!=0 & i!=_1)":

eval partd "?msd_neg_fib Ai G2[i]=H2[_(i+1)]":
```

`Walnut` returns `TRUE` when it evaluates `parta, partb, partc`. This completes the proof. □

Here is another way to see that **g** and **h** are generalizations of **f**:

Theorem 9.2. *A word is a factor of* **f** *if and only if it is a factor of* **g** *if and only if it is a factor of* **h**.

Proof. Recalling that $\mathbf{f}[0..\infty) = \mathbf{g}[0..\infty)$, we can use `Walnut` to verify these claims:

```
def gfactoreq "?msd_neg_fib At (t>=0 & t<n) => G2[i+t]=G2[j+t]":
def ghfactoreq "?msd_neg_fib At (t>=0 & t<n) => H2[i+t]=G2[j+t]":
eval test1 "?msd_neg_fib Ai,n Ej (j>=0) & $gfactoreq(i,j,n)":
eval test2 "?msd_neg_fib Ai,n Ej (j>=0) & $ghfactoreq(i,j,n)":
```

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

We now turn to an application of these ideas. We say that a finite word w is a (weak) quasiperiod of an infinite word **x** if w covers **x** by translates, where we allow w to partially "fall off" the left edge of **x**. Levé and Richomme [14] determined the (weak) quasiperiods of the (one-sided) Fibonacci word **f**. We can find a simpler but equivalent characterization of the (weak) quasiperiods using the bi-infinite Fibonacci word **g** as follows:

Theorem 9.3. *Suppose $n \geq 1$ and $F_{j-1} \leq n < F_j$ for some natural number j. The length-n word x is a quasiperiod of the bi-infinite Fibonacci word* **g** *if and only if there exists an index i, $0 \leq i \leq F_j - n - 2$, such that $x = \mathbf{g}[i..i+n-1]$.*

Proof. We first check whether a number in negaFibonacci representation is Fibonacci number and assert that a number is the smallest Fibonacci number that is larger than n as follows.

$$\begin{aligned} \text{isFib}(x) &:= \exists i\ x = F_i \\ \text{minFib}(n, s) &:= \text{isFib}(s) \wedge s > n \wedge \forall k\ (\text{isFib}(k) \wedge k > n) \implies k \geq s \end{aligned}$$

The statements translate into `Walnut` as follows. To generate the regular expression for isFib, we utilize the fact that the terms of the negaFibonacci system alternate in signs and $F_n = F_{n-1} + F_{n-3} + \cdots + F_1$ if n is odd.

```
reg isfib msd_neg_fib "(0*1(00)*|0*(10)*1)":
def minfib "?msd_neg_fib $isfib(s) & s>n & Ak ($isfib(k) & k>n) => k>=s":
```

We then use the following to assert quasiperiodicity in the bi-infinite Fibonacci word.

$$\begin{aligned} \text{gFactorEq}(i, j, n) &:= \forall t\ (t \geq 0 \wedge t < n) \implies \mathbf{g}[i + t] = \mathbf{g}[j + t] \\ \text{quasiBiFib}(k, n) &:= \forall i\ \exists j\ (i < j + n) \wedge (j \leq i) \wedge \text{gFactorEq}(k, j, n) \end{aligned}$$

We express the full theorem as follows.

$$\begin{aligned} &\forall k, n\ n \geq 1 \implies \\ &\quad (\text{quasiBiFib}(k, n) \iff \exists i, s\ \text{minFib}(n, s) \wedge i \geq 0 \wedge i \leq s - n - 2 \wedge \text{gFactorEq}(k, i, n)) \end{aligned}$$

Translating into `Walnut`, we have the following.

```
def gfactoreq "?msd_neg_fib At (t>=0 & t<n) => G2[i+t]=G2[j+t]":
def quasibifib "?msd_neg_fib Ai Ej i<j+n & j<=i & $gfactoreq(k,j,n)":
eval thm7bifib "?msd_neg_fib Ak,n (n>=1) => ($quasibifib(k,n)
   <=> (Ei,s $minfib(n,s) & i>=0 & i<=s-n-2 & $gfactoreq(k,i,n)))":
# 30 ms
# returns TRUE
```

This completes the proof. □

Remark 9.4. For two related papers about Fibonacci representations, see [3, 10].

10. Final remarks

`Walnut` can also be used to prove (or reprove) other results in Shevelev's paper [23]. However, these additional problems do not deal with negative bases, so they are not the focus of this paper. We simply mention briefly that his Open Problem A, also mentioned in [22], has recently been completely solved in [15].

Furthermore, his Theorem 6, dealing with the critical exponent of the sequence counting (modulo 2) the number of runs of 1's in the (ordinary) binary representation of n, can be easily proved and even improved as follows:

Theorem 10.1. *Define $r(n)$ to be the parity of the number of runs of 1's in the binary representation of n. Then the infinite word $\mathbf{r} = (r(n))_{n \geq 0}$ has no $(4 + \epsilon)$-powers, and this bound is optimal.*

Proof. It is easy to see that $\mathbf{r}[1..4] = 1111$, so clearly $\mathbf{r}$ has 4th powers. To show that $\mathbf{r}$ is 4^+-power-free, we use `Walnut`:

```
reg runs msd_2 "0*11*(00*11*00*11*)*0*":
# number of runs of 1's in binary expansion of n is odd
```

```
combine RU runs:
# turn it into a DFAO

eval has4e "Ei,n (n>=1) & At (t<=3*n) => RU[i+t]=RU[i+t+n]":
# assert it has (4+epsilon)-powers
# Walnut returns FALSE
```

□

The sequence **r** is sequence A268411 in the OEIS.

Acknowledgements. We are very grateful to Arseny Shur for his suggestion that we try to prove Theorem 6.2 with `Walnut`. Some of the calculations with `Walnut` were done on the CrySP RIPPLE Facility at the University of Waterloo. Thanks to Ian Goldberg for allowing us to run computations on this machine.

We also thank the referees and Robert Burns for their careful reading and several corrections.

References

[1] J.-P. Allouche and J.O. Shallit, The ubiquitous Prouhet-Thue-Morse sequence. In C. Ding, T. Helleseth, and H. Niederreiter, editors, *Sequences and Their Applications, Proceedings of SETA '98*. Springer-Verlag (1999), pp. 1–16.

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

[3] H. Alpert, Differences of multiple Fibonacci numbers. *INTEGERS* **9** (2009) 745–749.

[4] J. Berstel, Sur les mots sans carré définis par un morphisme. In H.A. Maurer, editor, *Proc. 6th Int'l Conf. on Automata, Languages, and Programming (ICALP)*, Vol. 71 of *Lecture Notes in Computer Science*. Springer-Verlag (1979), pp. 16–25.

[5] J. Berstel, Mots de Fibonacci. *Séminaire d'Informatique Théorique, LITP* **6-7** (1980–81) 57–78.

[6] J. Berstel, *Axel Thue's Papers on Repetitions in Words: a Translation*. Number 20 in Publications du Laboratoire de Combinatoire et d'Informatique Mathématique. Université du Québec à Montréal (1995).

[7] M.W. Bunder, Zeckendorf representations using negative Fibonacci numbers. *Fibonacci Quart.* **30** (1992) 111–115.

[8] J. Du and X. Su, On the existence of solutions for the Frenkel-Kontorova model on quasi-crystals. *Electron. Res. Arch.* **29** (2021) 4177–4198.

[9] V. Grünwald, Intorno all'aritmetica dei sistemi numerici a base negativa con particolare riguardo al sistema numerico a base negativo-decimale per lo studio delle sue analogie coll'aritmetica (decimale). *Giornale di Matematiche di Battaglini* **23** (1885) 203–221. Errata, p. 367.

[10] P. Hajnal, A short note on Zeckendorf type numeration systems with negative digits allowed. *Bull. ICA* **97** (2023) 54–66.

[11] D.E. Knuth, *The Art of Computer Programming*, Vol. 3: Seminumerical Algorithms. Addison-Wesley, 3rd ed. (1998).

[12] S. Labbé and J. Lepšová, A numeration system for Fibonacci-like Wang shifts. In T. Lecroq and S. Puzynina, editors, *WORDS 2021*, Vol. 12847 of *Lecture Notes in Computer Science*. Springer-Verlag (2021), pp. 104–116.

[13] C.G. Lekkerkerker, Voorstelling van natuurlijke getallen door een som van getallen van Fibonacci. *Simon Stevin* **29** (1952) 190–195.

[14] F. Levé and G. Richomme, Quasiperiodic infinite words: some answers. *Bull. Eur Assoc. Theor. Comput. Sci.* **84** (2004) 128–138.

[15] J. Meleshko, P. Ochem, J. Shallit and S.L. Shan, Pseudoperiodic words and a question of Shevelev. Preprint arXiv:2207.10171 (2022).

[16] H. Mousavi, Automatic theorem proving in `Walnut`. Preprint arXiv:1603.06017 (2016).

[17] H. Mousavi, L. Schaeffer and J. Shallit, Decision algorithms for Fibonacci-automatic words, I: basic results. *RAIRO Inform. Théor. Appl.* **50** (2016) 39–66.

[18] S.W. Rosema and R. Tijdeman, The Tribonacci substitution. *Electronic J. Combinatorics* **5** (2005) #A13 (electronic).

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

[20] J. Shallit, The Logical Approach to Automatic Sequences: Exploring Combinatorics on Words with `Walnut`. Vol. 482 of *London Math. Soc. Lecture Notes Series*. Cambridge University Press (2022).

[21] J. Shallit and A. Shur, Subword complexity and power avoidance. *Theoret. Comput. Sci.* **792** (2019) 96–116.

[22] V. Shevelev, Equations of the form $t(x+a) = t(x)$ and $t(x+a) = 1 - t(x)$ for Thue-Morse sequence. Preprint arXiv:0907.0880 (2012).

[23] V. Shevelev, Two analogs of the Thue-Morse sequence. Preprint arXiv:1603.04434 (2017).

[24] A.M. Shur, The structure of the set of cube-free $\mathbb{Z}$-words in a two-letter alphabet (Russian). *Izv. Ross. Akad. Nauk Ser. Mat.* **64** (2000) 201–224. English translation in *Izv. Math.* **64** (2000) 847–871.

[25] N.J.A. Sloane et al., The on-line encyclopedia of integer sequences. Available at https://oeis.org (2022).

[26] A. Thue, Über die gegenseitige Lage gleicher Teile gewisser Zeichenreihen. *Norske vid. Selsk. Skr. Mat. Nat. Kl.* **1** (1912) 1–67. Reprinted in *Selected Mathematical Papers of Axel Thue*, T. Nagell, editor, Universitetsforlaget, Oslo (1977) pp. 413–478.

[27] E. Zeckendorf, Représentation des nombres naturels par une somme de nombres de Fibonacci ou de nombres de Lucas. *Bull. Soc. Roy. Liège* **41** (1972) 179–182.