Decision Algorithms for Fibonacci-Automatic Words, III: Enumeration and Abelian Properties

Chen Fei Du[1], Hamoon Mousavi[1], Luke Schaeffer[2], and Jeffrey Shallit[1]

February 19, 2016

Abstract

We continue our study of the class of Fibonacci-automatic words. These are infinite words whose nth term is defined in terms of a finite-state function of the Fibonacci representation of n. In this paper, we show how enumeration questions (such as counting the number of squares of length n in the Fibonacci word) can be decided purely mechanically, using a decision procedure. We reprove some known results, in a unified way, using our technique, and we prove some new results. We also examine abelian properties of these words. As a consequence of our results on abelian properties, we get the result that every nontrivial morphic image of the Fibonacci word is Fibonacci-automatic.

1 Introduction

This is the third in a series of three papers on the ramifications of a decision procedure, based on first-order logic, for the Fibonacci-automatic words. These are infinite sequences $(a_n)_{n\geq 0}$ over a finite alphabet generated by finite automata that take, as input, the Fibonacci (or "Zeckendorf") representation of n and output a_n. The canonical example of a Fibonacci-automatic word is the famous Fibonacci word

$$\mathbf{f} = 01001010\cdots,$$

which is the unique fixed point of the map sending $0 \rightarrow 01$ and $1 \rightarrow 0$.

In our first paper [19], we explained this decision procedure in detail and gave a large number of consequences of it. In our second paper [10], we focussed on the application of the method to proving new avoidability results in words.

[1]School of Computer Science, University of Waterloo, Waterloo, ON N2L 3G1, Canada; `cfdu@uwaterloo.ca`, `sh2mousa@uwaterloo.ca`, `shallit@uwaterloo.ca` .

[2]Computer Science and Artificial Intelligence Laboratory, The Stata Center, MIT Building 32, 32 Vassar Street, Cambridge, MA 02139 USA; `lrschaeffer@gmail.com` .

In this, the third paper, we show that we can also use this decision procedure to solve two other kinds of problems dealing with these words: enumeration of the number of factors obeying various properties, and questions involving abelian properties.

There are four main goals of the paper. The first is to make the application of this logic-based method to enumeration and abelian problems more widely known to researchers. In many cases, the method can easily reprove results in the literature, replacing tedious inductions and long case-based arguments with a brief computation [21]. Towards this goal, we explain the method in some detail in this paper. Furthermore, our implementation of the decision procedure is called `Walnut`, and is available for free download at

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

We encourage researchers to make use of it.

The second goal is to point out how the method can apply equally well to questions about the *finite* Fibonacci words. Again, we can easily reprove known results, and also prove wide-ranging generalizations of them.

The third goal is to stress the usefulness of determining the finiteness of matrix semigroups (or monoids) in answering questions about sequences. This technique is explored in Section 4.

Finally, the fourth goal is to explain how our method can be used to solve basic questions about abelian properties of some (but not all) Fibonacci-automatic sequences. These questions were recently studied by Fici and Mignosi [11].

2 Basics

If x is a finite or infinite word, then by $x[i]$ we mean the i'th symbol of x, and by $x[i..j]$ we mean the factor beginning at position i and ending at position j. By Σ_k we mean the alphabet $\{0, 1, \ldots, k-1\}$.

Let the Fibonacci numbers be defined by $F_0 = 0$, $F_1 = 1$, and $F_n = F_{n-1} + F_{n-2}$ for $n \geq 2$. Recall [17, 22] that every integer $n \geq 0$ can be uniquely represented in the form $\sum_{2 \leq i \leq j} a_i F_i$, where $a_j = 1$ and $a_i a_{i+1} = 0$ for $2 \leq i < j$. We define $(n)_F$ to be the binary string $a_j a_{j-1} \cdots a_2$ (starting with the most significant digit). Similarly, for a word $w = b_1 b_2 \cdots b_j$ we define its interpretation in "base Fibonacci" $[w]_F = \sum_{1 \leq i \leq j} b_i F_{j+2-i}$. This representation can be extended to k-tuples of integers using the alphabet Σ_2^k and padding with leading zeros, if necessary. Thus, for example, the representation of $(17, 11)$ in base Fibonacci is $[1,0][0,1][0,0][1,1][0,0][1,0]$, where the first coordinates spell out 100101 and the second coordinates spell out 010100. An infinite word $\mathbf{a} = a_0 a_1 a_2 \cdots$ is called *Fibonacci-automatic* if there is a deterministic finite automaton with output (DFAO) that, on input $(n)_F$, ends in a state with output a_n.

Mimicking the base-k ideas previously explored in [6, 14], we can mechanically enumerate many aspects of Fibonacci-automatic words. More precisely, if a property P of the factors $\mathbf{a}[i..i+n-1]$ of a Fibonacci-automatic word $\mathbf{a}$ is expressible in Presburger arithmetic (together with indexing into the word), then there exists an algorithm to translate P into an automaton A accepting the Fibonacci representation of those pairs (i, n) for which the property holds, and for which i is as small as possible. The number of paths with second component corresponding to $(n)_F$ in this automaton in A then counts the number of distinct factors of length n for which P holds.

This gives the concept of *Fibonacci-regular sequence* as previously studied in [1]. Roughly speaking, a sequence $(a(n))_{n\geq 0}$ taking values in $\mathbb{N}$ is Fibonacci-regular if the set of sequences

$$\{(a([xw]_F)_{w\in\Sigma_2^*} \ : \ x \in \Sigma_2^*\}$$

is finitely generated. Here we assume that $a([xw]_F)$ evaluates to 0 if xw is an invalid Fibonacci representation, that is, if it contains the factor 11. Every Fibonacci-regular sequence $(a(n))_{n\geq 0}$ has a *linear representation* of the form (u, μ, v) where u and v are row and column vectors, respectively, and $\mu : \Sigma_2 \to \mathbb{N}^{d\times d}$ is a matrix-valued morphism, where $\mu(0) = M_0$ and $\mu(1) = M_1$ are $d \times d$ matrices for some $d \geq 1$, such that

$$a(n) = u \cdot \mu(x) \cdot v$$

whenever $[x]_F = n$. The *rank* of the representation is the integer d.

3 Subword complexity

Recall that if $\mathbf{x}$ is an infinite word, then the *subword complexity function* $\rho_{\mathbf{x}}(n)$ counts the number of distinct factors of length n. Then, in analogy with [6, Thm. 27], we have

Theorem 1. *If $\mathbf{x}$ is Fibonacci-automatic, then the subword complexity function $\rho_{\mathbf{x}}(n)$ is Fibonacci-regular.*

Proof. The idea is to use the predicate

$$\forall i' < i \ \mathbf{x}[i..i+n-1] \neq \mathbf{x}[i'..i'+n-1],$$

which expresses the assertion that the factor of length n beginning at position i has never appeared previously in $\mathbf{f}$. Then, for each n, the number of corresponding i gives $\rho_{\mathbf{f}}(n)$.

Next, we use the decision procedure mentioned previously to construct an automaton A from the predicate accepting the Fibonacci representation of pairs (i, n) for which the predicate holds.

The corresponding linear representation (u', μ', v') can now be constructed as follows: for each $a \in \Sigma_2$ the entry in row k and column ℓ of $\mu'(a)$ is the number of transitions in A from state k to state ℓ with second entry equal to a. The vector v' is a vector with 1's corresponding to the final states of A, and 0's elsewhere, and u' is similarly a vector with a single 1 corresponding to the initial state of A. Actually, to be precise, we need to replace u' by $u'M_0^{d-1}$ (if M_0 is a $d \times d$ matrix) in order to handle the possibility that representations need leading zeroes at the beginning to be counted. $\square$

As an example, of the technique, we exhibit a rank-6 linear representation for the sequence $a(n) = n + 1$:

$$u = [1\ 2\ 2\ 3\ 3\ 2]$$
$$M_0 = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$
$$M_1 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$
$$v = [1\ 0\ 0\ 0\ 0\ 0]^T.$$

This can be proved by a simple induction on the claim that

$$u \cdot \mu(x) = [x_F + 1\ (1x)_F + 1\ (10x)_F - x_F\ (100x)_F - x_F\ (101x)_F - (1x)_F\ (1001x)_F - (101x)_F]$$

for strings x.

Using our implementation, we can obtain a linear representation of the subword complexity function for the Fibonacci word **f**. In `Walnut` this can be done with the command

```
def fibsc "?msd_fib Aip (ip<i) => Ej (j<n) & F[i+j] != F[ip+j]":
```

which implements the predicate above for the specific case of the Fibonacci word.

(We briefly mention the basic syntax of `Walnut`: "`F`" refers to the sequence **f**. "`E`" is our abbreviation for $\exists$ and "`A`" is our abbreviation for $\forall$. The symbol "`=>`" is logical implication, "`&`" is logical AND, "`|`" is logical OR, "`~`" is logical NOT, and "`!=`" is logical NEQ. Constant values of sequences can be specified by preceding with the `@` sign. The "`?msd_fib`" asks the software to evaluate the predicate in Fibonacci representation. The order of free variables in a predicate is alphabetical order of the unbound variables that appear in it when it is defined. Once defined, a predicate can be referred to by preceding its name with the `$` sign.)

The resulting automaton has 10 states and is depicted in Figure 1. (We omit the "dead state" to which all non-displayed transitions go.)

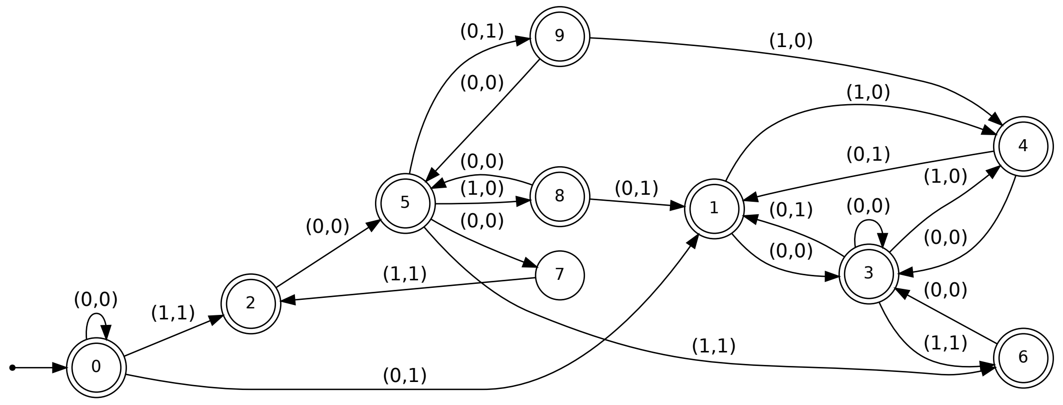

Figure 1: Automaton accepting $(i, n)_F$ for which $\mathbf{f}[i..i+n-1]$ is the first occurrence of that factor in $\mathbf{f}$

When we carry out our construction for the automaton above, we obtain the following linear representation (u', μ', v') of rank 10:

$$u' = [1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0]$$

$$\mu'(0) = M_0' = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$\mu'(1) = M_1' = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$v' = [1\ 1\ 1\ 1\ 1\ 1\ 1\ 0\ 1\ 1]^T$$

It remains to see that this linear representation indeed represents the function $n+1$. There are a variety of techniques to do this. One way, although not particularly computationally efficient, is to compare the values of the linear representations (u, μ, v) and (u', μ', v') for all strings of length at most $10 + 6 = 16$ (using [2, Corollary 3.6]). After checking this, we have now reproved in a purely mechanical way the following classic theorem of Morse and Hedlund [18]:

Theorem 2. *The subword complexity function of* $\mathbf{f}$ *is* $n+1$.

4 Applications to the finite Fibonacci words

One feature of the logic-based approach is that it applies not only to properties of the infinite Fibonacci word $\mathbf{f}$, but also to the finite Fibonacci words X_n. Recall that these words are defined as follows:

$$X_n = \begin{cases} \epsilon, & \text{if } n = 0; \\ 1, & \text{if } n = 1; \\ 0, & \text{if } n = 2; \\ X_{n-1}X_{n-2}, & \text{if } n > 2. \end{cases}$$

Note that $|X_n| = F_n$ for $n \geq 1$. Since the word X_n (for $n \neq 1$) is the prefix of length F_n of $\mathbf{f}$, our method can prove claims about the words X_n by rephrasing those claims in terms of prefixes of $\mathbf{f}$. In so doing our method gives a bonus it proves characterizations for *all* such prefixes of $\mathbf{f}$, not just those of a particular length.

As our first example, we now turn to a result of Fraenkel and Simpson [12]. They computed the exact number of squares (blocks of the form xx for x a nonempty word) appearing in the words X_n; this was previously estimated by Crochemore [8].

There are two variations of the problem: we can either count the number of distinct squares in X_n, or we can count what Fraenkel and Simpson called the number of "repeated squares" in X_n (i.e., the total number of *occurrences* of squares in X_n).

We can easily write down predicates for squares in a prefix of $\mathbf{f}$ of length n. The first represents the number of distinct squares in $\mathbf{f}[0..n-1]$:

$$L_{\text{ds}} := \{(i,j,n)_F \ : \ (j \geq 1) \text{ and } (i+2j \leq n) \text{ and } \mathbf{f}[i..i+j-1] = \mathbf{f}[i+j..i+2j-1] \\ \text{and } \forall i' < i \ \mathbf{f}[i'..i'+2j-1] \neq \mathbf{f}[i..i+2j-1]\},$$

or, in Walnut,

```
def fibeqfact "?msd_fib At (t<n) => F[i+t] = F[j+t]":
def fibds "?msd_fib (j>=1) & (i+2*j<=n) & $fibeqfact(i,i+j,j) & Aip (ip<i) =>
$fibeqfact(ip,i,2*j)":
```

This predicate asserts that $\mathbf{f}[i..i+2j-1]$ is a square occurring in $\mathbf{f}[0..n-1]$ and that furthermore it is the first occurrence of this particular string in $\mathbf{f}[0..n-1]$.

The second predicate represents the total number of occurrences of squares in $\mathbf{f}[0..n-1]$:

$$L_{\text{dos}} := \{(i,j,n)_F \ : \ (j \geq 1) \text{ and } (i+2j \leq n) \text{ and } \mathbf{f}[i..i+j-1] = \mathbf{f}[i+j..i+2j-1]\},$$

or, in Walnut,

```
def fibdos "?msd_fib (j>=1) & (i+2*j<=n) & $fibeqfact(i,i+j,j)":
```

This predicate asserts that $\mathbf{f}[i..i+2j-1]$ is a square occurring in $\mathbf{f}[0..n-1]$.

We apply our method to the second example, leaving the first to the reader. Let $b(n)$ denote the number of occurrences of squares in $\mathbf{f}[0..n-1]$. First, we use our method to find a DFA M accepting L_{dos}. This (incomplete) DFA has 27 states.

This gives us a linear representation of the sequence $b(n)$. Now let $B(n)$ denote the number of square occurrences in the finite Fibonacci word X_n. This corresponds to considering the Fibonacci representation of the form 10^{n-2}; that is, $B(n+1) = b([10^{n-1}]_F)$. The matrix M_0 in the linear representation is the following 27×27 array

$$M_0 = \left[\begin{array}{ccccccccccccccccccccccccccc}
1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&1&0&1&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0\\
0&0&0&0&0&0&1&0&1&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0\\
0&0&0&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&1&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&1\\
0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0\\
0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&1\\
0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0
\end{array}\right] \tag{1}$$

and has minimal polynomial

$$X^4(X-1)^2(X+1)^2(X^2-X-1)^2.$$

It now follows from the theory of linear recurrences that there are constants $c_1, c_2, \ldots, c_8$ such that

$$B(n+1) = (c_1 n + c_2)\alpha^n + (c_3 n + c_4)\beta^n + c_5 n + c_6 + (c_7 n + c_8)(-1)^n$$

for $n \geq 3$, where $\alpha = (1+\sqrt{5})/2$, $\beta = (1-\sqrt{5})/2$ are the roots of $X^2 - X - 1$. We can find these constants by computing $B(4), B(5), \ldots, B(11)$ (using the linear representation) and then solving for the values of the constants $c_1, \ldots, c_8$.

When we do so, we find

$$\begin{aligned}
c_1 &= \frac{2}{5} & \qquad c_2 &= -\frac{2}{25}\sqrt{5} - 2 \\
c_3 &= \frac{2}{5} & \qquad c_4 &= \frac{2}{25}\sqrt{5} - 2 \\
c_5 &= 1 & \qquad c_6 &= 1 \\
c_7 &= 0 & \qquad c_8 &= 0
\end{aligned}$$

A little simplification, using the fact that $F_n = (\alpha^n - \beta^n)/(\alpha - \beta)$, leads to

Theorem 3. *Let $B(n)$ denote the number of square occurrences in X_n. Then*

$$B(n+1) = \frac{4}{5}nF_{n+1} - \frac{2}{5}(n+6)F_n - 4F_{n-1} + n + 1$$

for $n \geq 3$.

This statement corrects a small error in Theorem 2 in [12] (the coefficient of F_{n-1} was wrong; note that their F and their Fibonacci words are indexed differently from ours), which was first pointed out to us by Kalle Saari. A corrigendum has now been published [13].

In a similar way, we can count the cube occurrences in X_n. These are blocks of the form xxx, where x is a nonempty word. Using analysis exactly like the square case, we easily obtain the following new result:

Theorem 4. *Let $C(n)$ denote the number of cube occurrences in the Fibonacci word X_n. Then for $n \geq 3$ we have*

$$C(n) = (d_1 n + d_2)\alpha^n + (d_3 n + d_4)\beta^n + d_5 n + d_6$$

where

$$\begin{aligned} d_1 &= \frac{3-\sqrt{5}}{10} & d_2 &= \frac{17}{50}\sqrt{5} - \frac{3}{2} \\ d_3 &= \frac{3+\sqrt{5}}{10} & d_4 &= -\frac{17}{50}\sqrt{5} - \frac{3}{2} \\ d_5 &= 1 & d_6 &= -1. \end{aligned}$$

We now turn to a question of Chuan and Droubay. Let us consider the prefixes of $\mathbf{f}$. For each prefix of length n, form all of its n shifts, and let us count the number of these shifts that are palindromes; call this number $d(n)$. (Note that in the case where a prefix is a power, two different shifts could be identical; we count these with multiplicity.)

Chuan [7, Thm. 7, p. 254] proved

Theorem 5. *For $i > 2$ we have $d(F_i) = 0$ iff $i \equiv 0 \pmod 3$.*

Our approach is first to write a predicate expressing that the shift by i of the prefix of $\mathbf{f}$ of length n is a palindrome, for $0 \leq i < n$.

If $i \leq n/2$, then this is equivalent to the assertion that

$$\mathbf{f}[i+t] = \mathbf{f}[i-1-t] \text{ for } 0 \leq t < i \text{ and } \mathbf{f}[2i+t] = \mathbf{f}[n-1-t] \text{ for } 0 \leq t < n-2i.$$

If $i \geq n/2$, then this is equivalent to the assertion that

$$\mathbf{f}[i+t] = \mathbf{f}[i-1-t] \text{ for } 0 \leq t < n-i \text{ and } \mathbf{f}[t] = \mathbf{f}[2i-n-t-1] \text{ for } 0 \leq t < 2i-n.$$

So let $\textsc{PalSP}(i,n)$ be the predicate

$$\begin{aligned}&(i<n) \wedge \\ &((2i<n) \implies ((\forall t<i\ \mathbf{f}[i+t]=\mathbf{f}[i-1-t]) \ \wedge (\forall t<n-2i\ \mathbf{f}[2i+t]=\mathbf{f}[n-1-t]))) \wedge \\ &((2i \geq n) \implies ((\forall t<n-i\ \mathbf{f}[i+t]=\mathbf{f}[i-1-t]) \ \wedge (\forall t<2i-n\ \mathbf{f}[t]=\mathbf{f}[2i-n-t-1]))).\end{aligned}$$

In `Walnut` we can write this as

```
def palsp "?msd_fib (i<n) & ((2*i < n) => (At (t<i) => (F[i+t]=F[i-1-t])) &
(At (t < n-2*i) => (F[2*i+t] = F[n-1-t]))) & ((2*i >= n) => (At (t < n-i) =>
(F[i+t]=F[i-1-t])) & (At (t < 2*i-n) => (F[t] = F[2*i-n-t-1])))":
```

Proof. We prove a stronger result than that of Chuan, characterizing $d(n)$ for *all* n in terms of finite automata, not just those n equal to a Fibonacci number.

We start by showing that $d(n)$ takes only three values: 0, 1, and 2. To do this, we construct an automaton accepting the language

$$\{(i,n)_F \ : \ (0 \leq i < n) \ \wedge \ \mathbf{f}[i..n-1]\mathbf{f}[0..i-1] \text{ is a palindrome } \}$$

using the predicate above. From this automaton we construct the linear representation (u, M_0, M_1, v) of $d(n)$ as discussed above; it has rank 27.

The range of d is finite if the monoid $\mathcal{M} = \langle M_0, M_1 \rangle$ is finite. This can be checked with a simple queue-based algorithm, and $\mathcal{M}$ turns out to have cardinality 151. Now a short computation proves that

$$\{uMv \ : \ M \in \mathcal{M}\} = \{0,1,2\},$$

and so our claim about the range of d follows.

Now that we know the range of d, we can create predicates $P_0(n), P_1(n), P_2(n)$ asserting that (a) there are no length-n shifts that are palindromes (b) there is exactly one shift that is a palindrome and (c) more than one shift is a palindrome, as follows:

$$\begin{aligned}&P_0 : \neg\exists i \ \textsc{PalSP}(i,n)\\ &P_1 : \exists i \ \textsc{PalSP}(i,n) \ \wedge \ \forall j \ (\textsc{PalSP}(j,n)) \implies (i=j))\\ &P_2 : \exists i \ \exists j \ \textsc{PalSP}(i,n) \ \wedge \ \textsc{PalSP}(j,n) \ \wedge \ (i \neq j),\end{aligned}$$

or, in `Walnut`,

```
eval palsp0 "?msd_Fib ~(Ei $palsp(i,n))":
eval palsp1 "?msd_fib Ei ($palsp(i,n) & (Aj $palsp(j,n) => (i=j)))":
eval palsp2 "?msd_fib Ei Ej $palsp(i,n) & $palsp(j,n) & (i!=j)":
```

For these predicates, we can compute finite automata characterizing the Fibonacci representations of those n for which $d(n)$ equals, respectively, 0, 1, and 2. They turn out to have $32, 20$, and 23 states, respectively.

For example, we computed the automaton corresponding to P_0, and it is displayed in Figure 2.

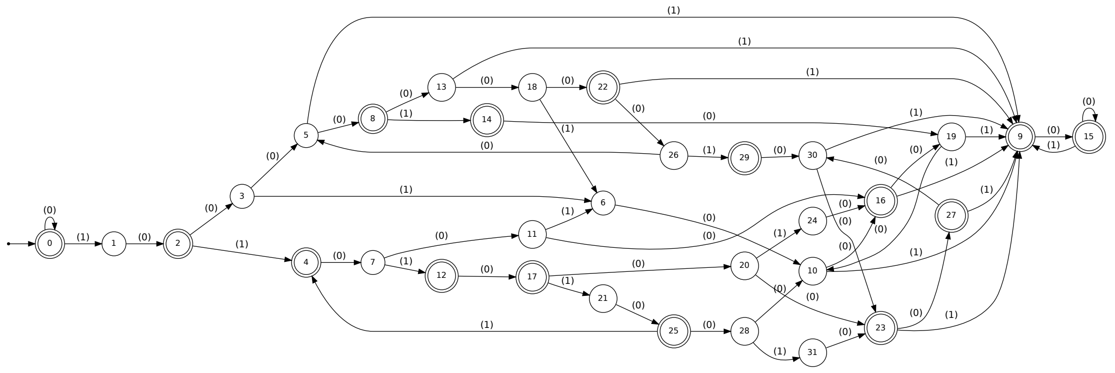

Figure 2: Automaton accepting lengths of prefixes for which no shifts are palindromes

By tracing the path labeled 10^* starting at the initial state labeled 0, we see that the "finality" of the states encountered (namely $(1, 2, 3, 5, 8, 13, 18, 6, 3, 5, 8, 13, 18, 6, \ldots)$) is ultimately periodic with period 3, thus proving Theorem 5. $\square$

To finish this section, we reprove a result concerning maximal repetitions in $\mathbf{f}$, which we previously examined in [19]. We recall that an integer p is called a *period* of a string x if $x[i] = x[i + p]$ for all integers i for which the index is meaningful. Let $p(x)$ denote the least period of x. Following Kolpakov and Kucherov [16], we say that $\mathbf{f}[i..i + n - 1]$ is a *maximal repetition* if

(a) $p(\mathbf{f}[i..i + n - 1]) \leq n/2$;

(b) $p(\mathbf{f}[i..i + n - 1]) < p(\mathbf{f}[i..i + n])$;

(c) If $i > 0$ then $p(\mathbf{f}[i..i + n - 1]) < p(\mathbf{f}[i - 1..i + n - 1])$.

They proved the following result on the number $\textsc{mr}(F_n)$ of occurrences of maximal repetitions in the prefix of $\mathbf{f}$ of length F_n:

Theorem 6. *For $n \geq 5$ we have $\textsc{mr}(F_n) = 2F_{n-2} - 3$.*

Proof. We create an automaton for the language

$$\{(i, j, n)_F \ : \ 0 \leq i \leq j < n \text{ and } \mathbf{f}[i..j] \text{ is a maximal repetition of } \mathbf{f}[0..n - 1]\},$$

using the predicate

$$\begin{gathered} \textsc{FibMR}(i, j, n) := (i \leq j) \ \wedge \ (j < n) \ \wedge \ \exists p \text{ with } 1 \leq p \leq (j + 1 - i)/2 \text{ such that} \\ ((\forall k \leq j - i - p \ \mathbf{f}[i + k] = \mathbf{f}[i + k + p]) \ \wedge \\ (i \geq 1) \implies (\forall q \text{ with } 1 \leq q \leq p \ \exists \ell \leq j - i - q + 1 \ \mathbf{f}[i - i + \ell] \neq \mathbf{f}[i - 1 + \ell + q]) \ \wedge \\ (j + 1 \leq n - 1) \implies (\forall r, \ 1 \leq r \leq p \ \exists m \leq j + 1 - r - i \ \mathbf{f}[i + m] \neq \mathbf{f}[i + m + r])). \end{gathered}$$

Here the second line of the predicate specifies that there is a period p of $\mathbf{f}[i..j]$ corresponding to a repetition of exponent at least 2. The third line specifies that no period q of $\mathbf{f}[i-1..j]$ (when this makes sense) can be $\leq p$, and the fourth line specifies that no period r of $\mathbf{f}[i..j+1]$ (when $j+1 \leq n-1$) can be $\leq p$.

In Walnut this is

```
eval fibmr "?msd_fib (i<=j)&(j<n)&Ep ((p>=1)&(2*p+i<=j+1)&
(Ak (k+i+p<=j) => (F[i+k]=F[i+k+p]))&((i>=1) => (Aq ((1<=q)&(q<=p)) =>
(El (l+i+q<=j+1)&(F[i+l-1]!=F[i+l+q-1]))))&((j+2<=n) => (Ar ((1<=r)&(r<=p)) =>
(Em (m+r+i<=j+1)&(F[i+m]!=F[i+m+r])))))":
```

From the automaton we deduce a linear representation (u, μ, v) of rank 59. Since $(F_n)_F = 10^{n-2}$, it suffices to compute the minimal polynomial of $M_0 = \mu(0)$. When we do this, we discover it is $X^4(X^2-X-1)(X-1)^2(X+1)^2$. It follows from the theory of linear recurrences that

$$\text{MR}(F_n) = e_1\alpha^n + e_2\beta^n + e_3 n + e_4 + (e_5 n + e_6)(-1)^n$$

for constants $e_1, e_2, e_3, e_4, e_5, e_6$ and $n \geq 6$. When we solve for $e_1, \ldots, e_6$ by using the first few values of $\text{MR}(F_n)$ (computed from the linear representation or directly) we discover that $e_1 = (3\sqrt{5}-5)/5$, $e_2 = (-3\sqrt{5}-5)/5$, $e_3 = e_5 = e_6 = 0$, and $e_4 = -3$. From this the result immediately follows. $\square$

In fact, we can prove even more.

Theorem 7. *For $n \geq 0$ the difference $\text{MR}(n+1) - \text{MR}(n)$ is either 0 or 1. Furthermore there is a finite automaton with 10 states that accepts $(n)_F$ precisely when $\text{MR}(n+1) - \text{MR}(n) = 1$.*

Proof. Every maximal repetition $\mathbf{f}[i..j]$ of $\mathbf{f}[0..n-1]$ is either a maximal repetition of $\mathbf{f}[0..n]$ with $j \leq n-1$, or is a maximal repetition with $j = n-1$ that, when considered in $\mathbf{f}[0..n]$, can be extended one character to the right to become one with $j = n$. So the only maximal repetitions of $\mathbf{f}[0..n]$ not (essentially) counted by $\text{MR}(n)$ are those such that

$$\mathbf{f}[i..n] \text{ is a maximal repetition of } \mathbf{f}[0..n] \text{ and } \mathbf{f}[i..n-1] \text{ is } \textit{not} \text{ a maximal repetition of } \mathbf{f}[0..n-1]. \quad (2)$$

We can easily create a predicate asserting this latter condition:

$$(\text{FibMR}(i, n, n+1) \& \neg \, \text{FibMR}(i, n-1, n)),$$

which in Walnut is

```
eval mrdiff "?msd_fib ($fibmr(i,n,n+1) & ~$fibmr(i,n-1,n))":
```

From this we obtain the linear representation of $\text{MR}(n+1) - \text{MR}(n)$:

$$u = [1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ ]$$

$$\mu(0) = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$\mu(1) = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$v = [0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 1\ 1\ 0]$$

We now use the trick we previously used for the proof of Theorem 5; the monoid generated by $\mu(0)$ and $\mu(1)$ has size 61 and for each matrix M in this monoid we have $uMv \in \{0, 1\}$. It follows that $\text{MR}(n+1) - \text{MR}(n) \in \{0, 1\}$ for all $n \geq 0$.

Knowing this, we can now build an automaton accepting those n for which there exists an i for which (2) holds. We do this with

$$\exists i\ (\text{FibMR}(i, n, n+1)\ \wedge\ \neg\, \text{FibMR}(i, n-1, n)),$$

which in `Walnut` is

```
eval mrd "?msd_fib Ei ($fibmr(i,n,n+1) & ~$fibmr(i,n-1,n))":
```

When we do so we get the automaton depicted in Figure 3.

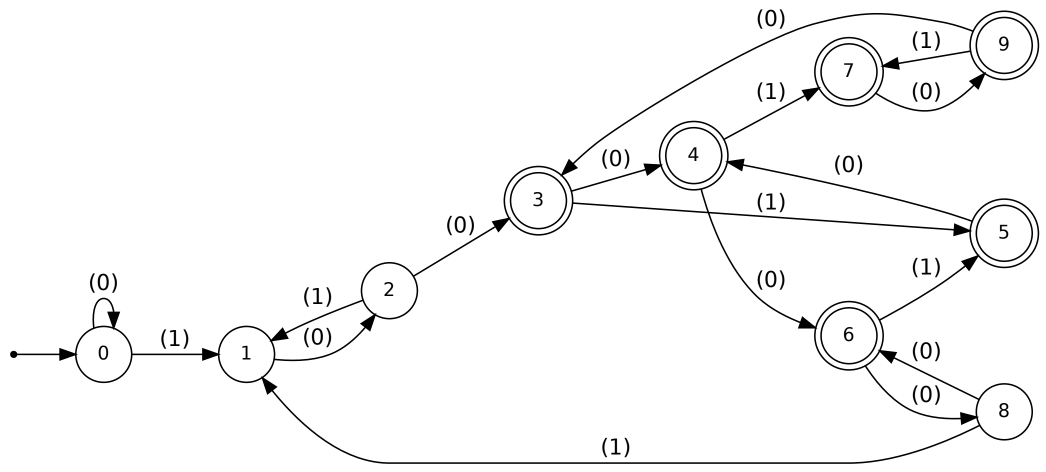

Figure 3: Automaton accepting $(n)_F$ such that $\textsc{mr}(n+1) - \textsc{mr}(n) = 1$

□

5 Abelian properties

Our decision procedure does not apply, in complete generality, to abelian properties of infinite words. This is because there is no obvious way to express assertions like $\psi(x) = \psi(x')$ for two factors x, x' of an infinite word. (Here $\psi : \Sigma^* \to \mathbb{N}^{|\Sigma|}$ is the Parikh map that sends a word to the number of occurrences of each letter.) Indeed, in the 2-automatic case it is provable that there is at least one abelian property that is inexpressible [20, §5.2].

However, the special nature of the Fibonacci word $\mathbf{f}$ allows us to mechanically prove some assertions involving abelian properties. In this section we describe how to do this.

By an *abelian square of order* n we mean a factor of the form xx' where $\psi(x) = \psi(x')$, where $n = |x|$. In a similar way we can define abelian cubes and higher powers.

We start with the elementary observation that $\mathbf{f}$ is defined over the alphabet Σ_2. Hence, to understand the abelian properties of a factor x it suffices to know $|x|$ and $|x|_0$. Next, we observe that the map that sends n to $a_n := |\mathbf{f}[0..n-1]|_0$ (that is, the number of 0's in the length-n prefix of $\mathbf{f}$), is actually *synchronized* (see [5, 3, 4, 15]). That is, there is a DFA accepting the Fibonacci representation of the pairs (n, a_n). In fact, we have the following result (which is probably not new, but we were not able to find it explicitly in the literature):

Theorem 8. *Suppose the Fibonacci representation of n is $e_1 e_2 \cdots e_i$. Then $a_n = [e_1 e_2 \cdots e_{i-1}]_F + e_i$.*

Proof. First, we observe that an easy induction on m proves that $|X_m|_0 = F_{m-1}$ for $m \geq 2$. We will use this in a moment.

The theorem's claim is easily checked for $n = 0, 1$. We prove it for $F_{m+1} \le n < F_{m+2}$ by induction on m. The base case is $m = 1$, which corresponds to $n = 1$.

Now assume the theorem's claim is true for $m - 1$; we prove it for m. Write $(n)_F = e_1 e_2 \cdots e_m$. Then, using the fact that $\mathbf{f}[0..F_{m+2} - 1] = X_{m+2} = X_{m+1} X_m$, we get

$$\begin{aligned} |\mathbf{f}[0..n-1]|_0 &= |\mathbf{f}[0..F_{m+1}-1]|_0 + |\mathbf{f}[F_{m+1}..n-1]|_0 \\ &= |X_{m+1}|_0 + |\mathbf{f}[0..n-1-F_{m+1}]|_0 \\ &= F_m + |\mathbf{f}[0..n-1-F_{m+1}|_0 \\ &= F_m + [e_2 \cdots e_{m-1}]_F + e_m \\ &= [e_1 \cdots e_{m-1}]_F + e_m, \end{aligned}$$

as desired. $\square$

In fact, the synchronized automaton for $(n, a_n)_F$ is given in the following diagram:

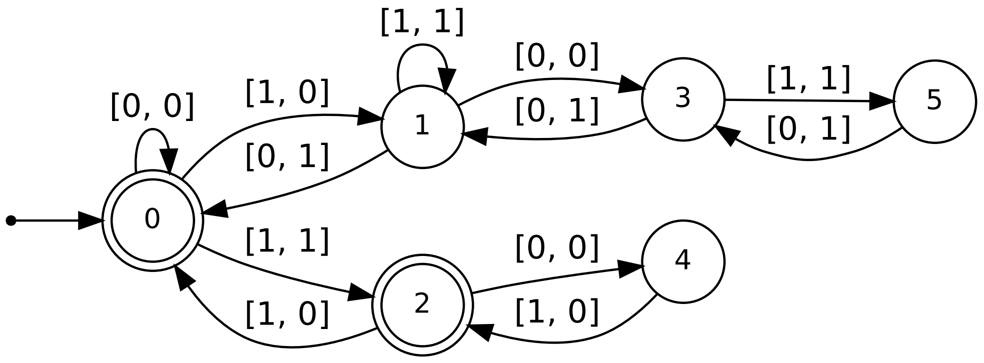

Figure 4: Automaton accepting $(n, a_n)_F$

The correctness of this automaton can be checked using our prover. Define $\text{SF}(x, y) = 1$ if $(x, y)_F$ is accepted and 0 otherwise, it suffices to check that

1. $\forall x\ \exists y\ \ \text{SF}(x, y) = 1$ (that is, for each x there is at least one corresponding y accepted);

2. $\forall x\, \forall y\, \forall z\ (\text{SF}(x, y) = \text{SF}(x, z)) \implies y = z$ (that is, for each x at most one corresponding y is accepted);

3. $\forall x\ \forall y\ ((\text{SF}(x, y) = 1)\ \wedge\ (\mathbf{f}[x] = 0)) \implies (\text{SF}(x + 1, y + 1) = 1)$;

4. $\forall x\ \forall y\ ((\text{SF}(x, y) = 1)\ \wedge\ (\mathbf{f}[x] = 1)) \implies (\text{SF}(x + 1, y) = 1)$;

In Walnut this can be checked by downloading the file sf.txt and placing it into Walnut's Word Automata library, and then executing the following commands:

```
eval sf1 "?msd_fib Ax Ey SF[x][y]=@1":
eval sf2 "?msd_fib Ax Ay Az (SF[x][y]=@1 & SF[x][z]=@1) => (y=z)":
eval sf3 "?msd_fib Ax Ay ((SF[x][y]=@1) & (F[x] = @0)) => (SF[x+1][y+1]=@1)":
eval sf4 "?msd_fib Ax Ay ((SF[x][y]=@1) & (F[x] = @1)) => (SF[x+1][y]=@1)"
```

Another useful automaton computes, on input n, i, j, the function

$$\textsc{fab}(n,i,j) := |\mathbf{f}[i..i+n-1]|_0 - |\mathbf{f}[j..j+n-1]|_0 = a_{i+n} - a_i - a_{j+n} + a_j.$$

From the known fact that the factors of $\mathbf{f}$ are "balanced" we know that $\textsc{fab}$ takes only the values $-1, 0, 1$. The automaton for $\textsc{fab}$ can be deduced from the one for $\textsc{sf}$ as follows: define

$$\textsc{fhw}(i,n,w) := \exists a\ \exists b\ \ \textsc{sf}(i,a)\ \wedge\ \textsc{sf}(i+n,b)\ \wedge\ w = a-b,$$

and then assembling $\textsc{fab}$ from the automata for

$$\exists u\ \exists v\ \ \textsc{fhw}(i,n,u)\ \wedge\ \textsc{fhw}(j,n,v)\ \wedge\ u - v = c$$

for $c \in \{-1, 0, 1\}$. However, in our case we calculated it by "guessing" the right automaton and then verifying the correctness with our prover.

The automaton for $\textsc{fab}(n,i,j)$ has 30 states, numbered from 0 to 29. Inputs are in Σ_2^3. The transitions, as well as the outputs, are given in Table 1.

q	$[0,0,0]$	$[0,0,1]$	$[0,1,0]$	$[0,1,1]$	$[1,0,0]$	$[1,0,1]$	$[1,1,0]$	$[1,1,1]$	$\tau(q)$
0	0	1	2	3	3	4	5	6	0
1	7	0	8	2	2	3	9	5	0
2	10	11	0	1	1	12	3	4	0
3	13	10	7	0	0	1	2	3	0
4	14	10	15	0	0	1	2	3	1
5	16	17	7	0	0	1	2	3	-1
6	18	17	15	0	0	1	2	3	0
7	0	1	2	3	3	19	5	20	0
8	10	11	0	1	1	21	3	19	0
9	17	22	0	1	1	12	3	4	-1
10	0	1	2	3	3	4	23	24	0
11	7	0	8	2	2	3	25	23	0
12	15	0	26	2	2	3	9	5	1
13	0	1	2	3	3	19	23	27	0
14	1	12	3	4	4	19	24	27	-1
15	1	12	3	4	4	19	6	20	-1
16	2	3	9	5	5	20	23	27	1
17	2	3	9	5	5	6	23	24	1
18	3	4	5	6	6	20	24	27	0
19	14	13	15	7	7	0	8	2	1
20	18	16	15	7	7	0	8	2	0
21	15	7	26	8	8	2	28	9	1
22	8	2	28	9	9	5	25	23	1
23	16	17	13	10	10	11	0	1	-1
24	18	17	14	10	10	11	0	1	0
25	17	22	10	11	11	29	1	12	-1
26	11	29	1	12	12	21	4	19	-1
27	18	16	14	13	13	10	7	0	0
28	17	22	0	1	1	21	3	19	-1
29	15	0	26	2	2	3	25	23	1

Table 1: Automaton to compute $\textsc{fab}$

Once we have guessed the automaton, we can verify it as follows:

1. $\forall i\ \forall j\ \ \textsc{fab}[0][i][j] = 0$. This is the basis for an induction.

2. Induction steps:

   - $\forall i\ \forall j\ \forall n\ (\mathbf{f}[i+n] = \mathbf{f}[j+n]) \implies (\textsc{fab}[n][i][j] = \textsc{fab}[n+1][i][j])$.
   - $\forall i\ \forall j\ \forall n\ ((\mathbf{f}[i+n] = 0) \wedge (\mathbf{f}[j+n] = 1)) \implies (((\textsc{fab}[n][i][j] = -1) \wedge (\textsc{fab}[n+1][i][j] = 0)) \vee ((\textsc{fab}[n][i][j] = 0) \wedge (\textsc{fab}[n+1][i][j] = 1)))$
   - $\forall i\, \forall j\, \forall n\, ((\mathbf{f}[i+n] = 0) \wedge (\mathbf{f}[j+n] = 1)) \implies (((\textsc{fab}[n][i][j] = 1) \wedge (\textsc{fab}[n+1][i][j] = 0)) \vee ((\textsc{fab}[n][i][j] = 0) \wedge (\textsc{fab}[n+1][i][j] = -1)))$.

In Walnut these four checks can be done as follows:

```
eval fab1 "?msd_fib Ai Aj FAB[0][i][j] = @0":
eval fab2 "?msd_fib Ai Aj An (F[i+n]=F[j+n]) => (FAB[n][i][j]=FAB[n+1][i][j])":
eval fab3 "?msd_fib Ai Aj An ((F[i+n]=@0)&(F[j+n]=@1)) => (((FAB[n][i][j]=@-1)&
(FAB[n+1][i][j]=@0)) | ((FAB[n][i][j]=@0)&(FAB[n+1][i][j]=@1)))":
```

```
eval fab4 "?msd_fib Ai Aj An ((F[i+n]=@1)&(F[j+n]=@0)) => (((FAB[n][i][j]=@1)&
(FAB[n+1][i][j]=@0)) | ((FAB[n][i][j]=@0)&(FAB[n+1][i][j]=@-1)))":
```

As the first application, we prove

Theorem 9. *The Fibonacci word* **f** *has abelian squares of all orders.*

Proof. We use the predicate

$$\exists i \ (\text{FAB}[n][i][i+n] = 0).$$

The resulting automaton accepts all $n \geq 0$. The total computing time was 141 ms. □

Cummings and Smyth [9] counted the total number of all occurrences of (nonempty) abelian squares in the Fibonacci words X_i. We can do this by using the predicate

$$(k > 0) \wedge (i + 2k \leq n) \wedge (\text{FAB}[k][i][i+k] = 0),$$

using the techniques in Section 3 and considering the case where $n = F_i$.

When we do, we get a linear representation of rank 127 that counts the total number $w(n)$ of occurrences of abelian squares in the prefix of length n of the Fibonacci word.

To recover the Cummings-Smyth result we compute the minimal polynomial of the matrix M_0 corresponding to the predicate above. It is

$$x^4(x-1)(x+1)(x^2+x+1)(x^2-3x+1)(x^2-x+1)(x^2+x-1)(x^2-x-1).$$

This means that $w(F_n)$, that is, w evaluated at 10^{n-2} in Fibonacci representation, is a linear combination of the roots of this polynomial to the n'th power (more precisely, the $(n-2)$th, but this detail is unimportant). The roots of the polynomial are

$$-1, 1, (-1 \pm i\sqrt{3})/2, (3 \pm \sqrt{5})/2, (1 \pm i\sqrt{3})/2, (-1 \pm \sqrt{5})/2, (1 \pm \sqrt{5})/2.$$

Solving for the coefficients as we did in Section 3 we get

Theorem 10. *For all $n \geq 0$ we have*

$$w(F_n) = c_1 \left(\frac{3+\sqrt{5}}{2}\right)^n + c_1 \left(\frac{3-\sqrt{5}}{2}\right)^n + c_2 \left(\frac{1+\sqrt{5}}{2}\right)^n + c_2 \left(\frac{1-\sqrt{5}}{2}\right)^n +$$
$$c_3 \left(\frac{1+i\sqrt{3}}{2}\right)^n + \overline{c_3} \left(\frac{1-i\sqrt{3}}{2}\right)^n + c_4 \left(\frac{-1+i\sqrt{3}}{2}\right)^n + \overline{c_4} \left(\frac{-1-i\sqrt{3}}{2}\right)^n + c_5(-1)^n,$$

where

$$\begin{aligned} c_1 &= 1/40 \\ c_2 &= -\sqrt{5}/20 \\ c_3 &= (1 - i\sqrt{3})/24 \\ c_4 &= i\sqrt{3}/24 \\ c_5 &= -2/15, \end{aligned}$$

and here $\overline{x}$ denotes complex conjugate. Here the parts corresponding to the constants c_3, c_4, c_5 form a periodic sequence of period 6.

Next, we turn to what is apparently a new result. Let $h(n)$ denote the total number of distinct factors (not occurrences of factors) that are abelian squares in the Fibonacci word X_n.

In this case we need the predicate

$$(k \geq 1) \wedge (i + 2k \leq n) \wedge (\text{FAB}[k][i][i+k] = 0) \wedge (\forall j < i \ (\exists t < 2k \ (\mathbf{f}[j+t] \neq \mathbf{f}[i+t]))).$$

We get the minimal polynomial

$$x^4(x+1)(x^2+x+1)(x^2-3x+1)(x^2-x+1)(x^2+x-1)(x^2-x-1)(x-1)^2.$$

Using the same technique as above we get

Theorem 11. *For $n \geq 2$ we have $h(n) = a_1 c_1^n + \cdots + a_{10} c_{10}^n$ where*

$$\begin{aligned} a_1 &= (-2+\sqrt{5})/20 & c_1 &= (3+\sqrt{5})/2 \\ a_2 &= (-2-\sqrt{5})/20 & c_2 &= (3-\sqrt{5})/2 \\ a_3 &= (5-\sqrt{5})/20 & c_3 &= (1+\sqrt{5})/2 \\ a_4 &= (5+\sqrt{5})/20 & c_4 &= (1-\sqrt{5})/2 \\ a_5 &= 1/30 & c_5 &= -1 \\ a_6 &= -5/6 & c_6 &= 1 \\ a_7 &= (1/12) - i\sqrt{3}/12 & c_7 &= (1/2) + i\sqrt{3}/2 \\ a_8 &= (1/12) + i\sqrt{3}/12 & c_8 &= (1/2) - i\sqrt{3}/2 \\ a_9 &= (1/6) + i\sqrt{3}/12 & c_9 &= (-1/2) + i\sqrt{3}/2 \\ a_{10} &= (1/6) - i\sqrt{3}/12 & c_{10} &= (-1/2) - i\sqrt{3}/2. \end{aligned}$$

For another new result, consider counting the total number $a(n)$ of distinct factors of length $2n$ of the infinite word $\mathbf{f}$ that are abelian squares.

This function is rather erratic. The following table gives the first few values:

n	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
$a(n)$	1	3	5	1	9	5	5	15	3	13	13	5	25	9	15	25	1	27	19	11

We use the predicate

$$(n \geq 1) \wedge (\text{FAB}[n][i][i+n] = 0) \wedge (\forall j < i \ (\exists t < 2n \ (\mathbf{f}[j+t] \neq \mathbf{f}[i+t]))).$$

to create the matrices and vectors.

Theorem 12. *$a(n) = 1$ infinitely often and $a(n) = 2n-1$ infinitely often. More precisely $a(n) = 1$ iff $(n)_F = 1$ or $(n)_F = (100)^i 101$ for $i \geq 0$, and $a(n) = 2n-1$ iff $(n)_F = 10^i$ for $i \geq 0$.*

Proof. For the first statement, we create a DFA accepting those $(n)_F$ for which $a(n) = 1$, via the predicate

$$\forall i\ \forall j\ ((\text{FAB}[n][i][i+n] = 0) \wedge (\text{FAB}[n][j][j+n] = 0)) \implies (\forall t < 2n\ (\mathbf{f}[j+t] = \mathbf{f}[i+t])).$$

The resulting 6-state automaton accepts the set specified.

For the second result, we first compute the minimal polynomial of the matrix M_0 of the linear representation. It is $x^5(x-1)(x+1)(x^2-x-1)$. This means that, for $n \geq 5$, we have $a(F_n) = c_1 + c_2(-1)^n + c_3\alpha^n + c_4\beta^n$ where, as usual, $\alpha = (1+\sqrt{5})/2$ and $\beta = (1-\sqrt{5})/2$. Solving for the constants, we determine that $a(F_n) = 2F_n - 1$ for $n \geq 2$, as desired.

To show that these are the only cases for which $a(n) = 2n - 1$, we use a predicate that says that there are not at least three different factors of length $2n$ that are not abelian squares. Running this through our program results in only the cases previously discussed. □

Finally, we turn to abelian cubes. Unlike the case of squares, some orders do not appear in $\mathbf{f}$.

Theorem 13. *The Fibonacci word* $\mathbf{f}$ *contains, as a factor, an abelian cube of order* n *iff* $(n)_F$ *is accepted by the automaton in Figure 5.*

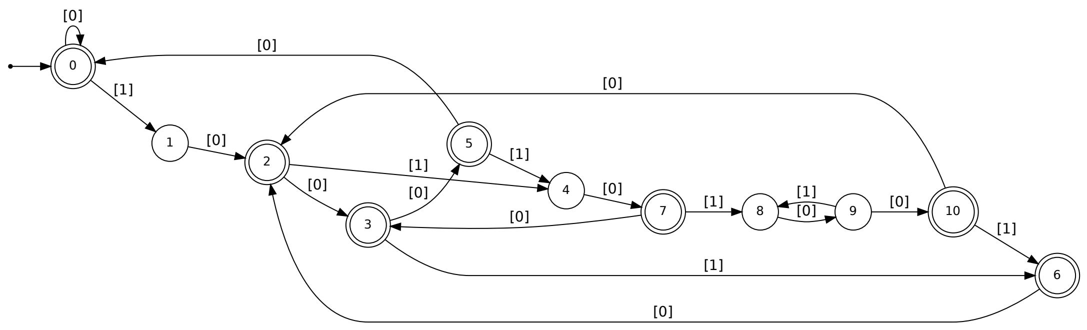

Figure 5: Automaton accepting orders of abelian cubes in $\mathbf{f}$

Theorem 8 has the following interesting corollary.

Corollary 14. *Let* $h : \{\Sigma_2\}^* \to \Delta^*$ *be an arbitrary morphism such that* $h(01) \neq \epsilon$. *Then* $h(\mathbf{f})$ *is an infinite Fibonacci-automatic word.*

Proof. From Theorem 8 we see that there is a predicate $\text{SF}(n, n')$ which is true if $n' = |\mathbf{f}[0..n-1]|_0$ and false otherwise, and this predicate can be implemented as a finite automaton taking the inputs n and n' in Fibonacci representation.

Suppose $h(0) = w$ and $h(1) = x$. Now, to show that $\mathrm{h}(\mathbf{f})$ is Fibonacci-automatic, it suffices to show that, for each letter $a \in \Delta$, the language of "fibers"

$$L_a = \{(n)_F : (h(\mathbf{f}))[n] = a\}$$

is regular.

To see this, we write a predicate for the n in the definition of L_a, namely

$$\begin{aligned}&\exists q\ \exists r_0\ \exists r_1\ \exists m\ (q \le n < q + |h(\mathbf{f}[m])|)\ \wedge\ \text{SF}(m, r_0)\ \wedge\ (r_0 + r_1 = m) \wedge \\ &\qquad (r_0|w| + r_1|x| = q)\ \wedge\ ((\mathbf{f}[m] = 0\ \wedge\ w[n-q] = a)\ \vee\ (\mathbf{f}[m] = 1\ \wedge\ x[n-q] = a)).\end{aligned}$$

Notice that the predicate looks like it uses multiplication, but this multiplication can be replaced by repeated addition since $|w|$ and $|x|$ are constants here.

Unpacking this predicate we see that it asserts the existence of m, q, r_0, and r_1 having the meaning that

- the n'th symbol of $\mathrm{h}(\mathbf{f})$ lies inside the block $h(\mathbf{f}[m])$ and is in fact the $(n-q)$'th symbol in the block (with the first symbol being symbol 0)
- $\mathbf{f}[0..m-1]$ has r_0 0's in it
- $\mathbf{f}[0..m-1]$ has r_1 1's in it
- the length of $h(\mathbf{f}[0..m-1])$ is q

Since everything in this predicate is in the logical theory $(\mathbb{N}, +, <, F)$ where F is the predicate for the Fibonacci word, the language L_a is regular. □

Remark 15. Notice that everything in this proof goes through for other numeration systems, provided the original word has the property that the Parikh vector of the prefix of length n is synchronized.

6 Acknowledgments

We thank Kalle Saari for bringing our attention to the small error in [12]. We thank Narad Rampersad and Michel Rigo for useful suggestions.

We used `Maple` to compute the minimal polynomials of matrices, and `gv` to display the automata computed by `Walnut`.

References

[1] J.-P. Allouche, K. Scheicher, and R. F. Tichy. Regular maps in generalized number systems. *Math. Slovaca* **50** (2000), 41–58.

[2] J. Berstel and C. Reutenauer. *Noncommutative Rational Series with Applications*, Vol. 137 of *Encylopedia of Mathematics and Its Applications*. Cambridge University Press, 2011.

[3] A. Carpi and V. D'Alonzo. On the repetitivity index of infinite words. *Internat. J. Algebra Comput.* **19** (2009), 145–158.

[4] A. Carpi and V. D'Alonzo. On factors of synchronized sequences. *Theoret. Comput. Sci.* **411** (2010), 3932–3937.

[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. *Internat. J. Found. Comp. Sci.* **23** (2012), 1035–1066.

[7] W.-F. Chuan. Symmetric Fibonacci words. *Fibonacci Quart.* **31** (1993), 251–255.

[8] M. Crochemore. An optimal algorithm for computing the repetitions in a word. *Inform. Process. Lett.* **12** (1981), 244–250.

[9] L. J. Cummings and W. F. Smyth. Weak repetitions in strings. *J. Combin. Math. Combin. Comput.* **24** (1997), 33–48.

[10] C. F. Du, H. Mousavi, E. Rowland, L. Schaeffer, and J. Shallit. Decision algorithms for Fibonacci-automatic words, II: Related sequences and avoidability. Preliminary version available at `https://cs.uwaterloo.ca/~shallit/Papers/part2e.pdf`, 2015.

[11] G. Fici and F. Mignosi. Words with the maximum number of abelian squares. Preprint. Available at `http://arxiv.org/abs/1506.03562`, 2015.

[12] A. S. Fraenkel and J. Simpson. The exact number of squares in Fibonacci words. *Theoret. Comput. Sci.* **218** (1999), 95–106.

[13] A. S. Fraenkel and J. Simpson. Corrigendum to "The exact number of squares in Fibonacci words" [Theoret. Comput. Sci. **218** (1) (1999) 95–106]. *Theoret. Comput. Sci.* **547** (2014), 122.

[14] D. Goc, H. Mousavi, and J. Shallit. On the number of unbordered factors. In A.-H. Dediu, C. Martin-Vide, and B. Truthe, editors, *LATA 2013*, Vol. 7810 of *Lecture Notes in Computer Science*, pp. 299–310. Springer-Verlag, 2013.

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

[16] R. Kolpakov and G. Kucherov. On maximal repetitions in words. In G. Ciobanu and G. Păun, editors, *Fundamentals of Computation Theory: FCT '99*, Vol. 1684 of *Lecture Notes in Computer Science*, pp. 374–385. Springer-Verlag, 1999.

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

[18] M. Morse and G. A. Hedlund. Symbolic dynamics II. Sturmian trajectories. *Amer. J. Math.* **62** (1940), 1–42.

[19] H. Mousavi, L. Schaeffer, and J. Shallit. Decision algorithms for Fibonacci-automatic words, I: Basic results. To appear, *RAIRO Informatique*. Preliminary version available at `https://cs.uwaterloo.ca/~shallit/Papers/part1.pdf`, 2016.

[20] L. Schaeffer. Deciding properties of automatic sequences. Master's thesis, University of Waterloo, 2013.

[21] J. Shallit. Enumeration and automatic sequences. *Pure Math. Appl.* **25** (2015), 96–106.

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