Decision Algorithms for Fibonacci-Automatic Words, I: Basic Results

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

April 10, 2015

Abstract

We implement a decision procedure for answering questions about a class of infinite words that might be called (for lack of a better name) "Fibonacci-automatic". This class includes, for example, the famous Fibonacci word $\mathbf{f} = 01001010\cdots$, the fixed point of the morphism $0 \to 01$ and $1 \to 0$. We then recover many results about the Fibonacci word from the literature (and improve some of them), such as assertions about the occurrences in $\mathbf{f}$ of squares, cubes, palindromes, and so forth.

1 Decidability

As is well-known, the logical theory $\mathrm{Th}(\mathbb{N}, +)$, sometimes called Presburger arithmetic, is decidable [43, 44]. Büchi [10] showed that if we add the function $V_k(n) = k^e$, for some fixed integer $k \geq 2$, where $e = \max\{i \ : \ k^i \mid n\}$, then the resulting theory is still decidable. This theory is powerful enough to define finite automata; for a survey, see [9].

As a consequence, we have the following theorem (see, e.g., [50]):

Theorem 1. *There is an algorithm that, given a proposition phrased using only the universal and existential quantifiers, indexing into one or more k-automatic sequences, addition, subtraction, logical operations, and comparisons, will decide the truth of that proposition.*

Here, by a k-automatic sequence, we mean a sequence $\mathbf{a}$ computed by deterministic finite automaton with output (DFAO) $M = (Q, \Sigma_k, \Delta, \delta, q_0, \kappa)$. Here $\Sigma_k := \{0, 1, \ldots, k-1\}$ is the input alphabet, Δ is the output alphabet, and outputs are associated with the states given by the map $\kappa : Q \to \Delta$ in the following manner: if $(n)_k$ denotes the canonical expansion of n in base k, then $\mathbf{a}[n] = \kappa(\delta(q_0, (n)_k))$. The prototypical example of an automatic sequence

[1]School of Computer Science, University of Waterloo, Waterloo, ON N2L 3G1, Canada; 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 .

is the Thue-Morse sequence $\mathbf{t} = t_0 t_1 t_2 \cdots$, the fixed point (starting with 0) of the morphism $0 \to 01$, $1 \to 10$.

It turns out that many results in the literature about properties of automatic sequences, for which some had only long and involved proofs, can be proved purely mechanically using a decision procedure. It suffices to express the property as an appropriate logical predicate, convert the predicate into an automaton accepting representations of integers for which the predicate is true, and examine the automaton. See, for example, the recent papers [2, 31, 33, 32, 34]. Furthermore, in many cases we can explicitly enumerate various aspects of such sequences, such as subword complexity [13].

Beyond base k, more exotic numeration systems are known, and one can define automata taking representations in these systems as input. It turns out that in the so-called Pisot numeration systems, addition is computable [29, 30], and hence a theorem analogous to Theorem 1 holds for these systems. See, for example, [8]. It is our contention that the power of this approach has not been widely appreciated, and that many results, previously proved using long and involved ad hoc techniques, can be proved with much less effort by phrasing them as logical predicates and employing a decision procedure.

We have implemented a decision algorithm for one such system; namely, Fibonacci representation. In this paper we report on our results obtained using this implementation. We have reproved many results in the literature purely mechanically, as well as obtained new results, using this implementation. In this paper we focus on results on the infinite and finite Fibonacci words.

The paper is organized as follows. In Section 2, we briefly recall the details of Fibonacci representation. In Section 3 we report on our mechanical proofs of properties of the infinite Fibonacci word; we reprove many old results and we prove some new ones. In Section 4 we apply our ideas to prove results about the finite Fibonacci words. Some details about our implementation are given in the last section.

This paper, and the two companion papers [22, 23], represent the full version of a paper presented at the Journées Montoises on September 26 2014.

2 Fibonacci representation

Let the Fibonacci numbers be defined, as usual, by $F_0 = 0$, $F_1 = 1$, and $F_n = F_{n-1} + F_{n-2}$ for $n \geq 2$. (We caution the reader that some authors use a different indexing for these numbers.)

It is well-known, and goes back to Ostrowski [41], Lekkerkerker [39], and Zeckendorf [51], that every non-negative integer can be represented, in an essentially unique way, as a sum of Fibonacci numbers $(F_i)_{i \geq 2}$, subject to the constraint that no two consecutive Fibonacci numbers are used. For example, $43 = F_9 + F_6 + F_2$. Also see [11, 25].

Such a representation can be written as a binary string $a_1 a_2 \cdots a_n$ representing the integer $\sum_{1 \leq i \leq n} a_i F_{n+2-i}$. For example, the binary string 10010001 is the Fibonacci representation of 43.

For $w = a_1 a_2 \cdots a_n \in \Sigma_2^*$, we define $[a_1 a_2 \cdots a_n]_F := \sum_{1 \leq i \leq n} a_i F_{n+2-i}$, even if $a_1 a_2 \cdots a_n$ has leading zeroes or consecutive 1's. By $(n)_F$ we mean the *canonical* Fibonacci representa-

tion for the integer n, having no leading zeroes or consecutive 1's. Note that $(0)_F = \epsilon$, the empty string. The language of all canonical representations of elements of $\mathbb{N}$ is $\epsilon + 1(0+01)^*$.

Just as Fibonacci representation is the analogue of base-k representation, we can define the notion of *Fibonacci-automatic sequence* as the analogue of the more familiar notation of k-automatic sequence [16, 3]. We say that an infinite word $\mathbf{a} = (a_n)_{n\geq 0}$ is Fibonacci-automatic if there exists an automaton with output $M = (Q, \Sigma_2, q_0, \delta, \kappa, \Delta)$ that $a_n = \kappa(\delta(q_0, (n)_F))$ for all $n \geq 0$. An example of a Fibonacci-automatic sequence is the infinite Fibonacci word,

$$\mathbf{f} = f_0 f_1 f_2 \cdots = 01001010\cdots$$

which is generated by the following 2-state automaton:

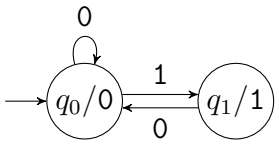

Figure 1: Canonical Fibonacci representation DFAO generating the Fibonacci word

To compute f_i, we express i in canonical Fibonacci representation, and feed it into the automaton. Then f_i is the output associated with the last state reached (denoted by the symbol after the slash). Another characterization of Fibonacci-automatic sequences can be found in [49].

A basic fact about Fibonacci representation is that addition can be performed by a finite automaton. To make this precise, we need to generalize our notion of Fibonacci representation to r-tuples of integers for $r \geq 1$. A representation for $(x_1, x_2, \ldots, x_r)$ consists of a string of symbols z over the alphabet Σ_2^r, such that the projection $\pi_i(z)$ over the i'th coordinate gives a Fibonacci representation of x_i. Notice that since the canonical Fibonacci representations of the individual x_i may have different lengths, padding with leading zeroes will often be necessary. A representation for $(x_1, x_2, \ldots, x_r)$ is called canonical if it has no leading $[0, 0, \ldots 0]$ symbols and the projections into individual coordinates have no occurrences of 11. We write the canonical representation as $(x_1, x_2, \ldots, x_r)_F$. Thus, for example, the canonical representation for $(9, 16)$ is $[0,1][1,0][0,0][0,1][0,0][1,0]$.

Thus, our claim about addition in Fibonacci representation is that there exists a deterministic finite automaton (DFA) M_{add} that takes input words of the form $[0,0,0]^*(x, y, z)_F$, and accepts if and only if $x+y = z$. Thus, for example, M_{add} accepts $[0,0,1][1,0,0][0,1,0][1,0,1]$, since the three strings obtained by projection are $0101, 0010, 1001$, which represent, respectively, 4, 2, and 6 in Fibonacci representation. This result is apparently originally due to Berstel [5]; also see [6, 27, 28, 1].

Since this automaton does not appear to have been given explicitly in the literature and it is essential to our implementation, we give it here. The states of M_{add} are $Q = \{0, 1, 2, \ldots, 16\}$, the input alphabet is $\Sigma_2 \times \Sigma_2 \times \Sigma_2$, the final states are $F = \{1, 7, 11\}$,

the initial state is $q_0 = 1$, and the transition function δ is given below. The automaton is incomplete, with any unspecified transitions going to a non-accepting dead state that transitions to itself on all inputs. This automaton actually works even for non-canonical expansions having consecutive 1's; an automaton working only for canonical expansions can easily be obtained by intersection with the appropriate regular languages. The state 0 is a "dead state" that can safely be ignored.

	[0,0,0]	[0,0,1]	[0,1,0]	[0,1,1]	[1,0,0]	[1,0,1]	[1,1,0]	[1,1,1]
0	0	0	0	0	0	0	0	0
1	1	2	3	1	3	1	0	3
2	4	5	6	4	6	4	7	6
3	0	8	0	0	0	0	0	0
4	5	0	4	5	4	5	6	4
5	0	0	0	0	0	0	9	0
6	2	10	1	2	1	2	3	1
7	8	11	0	8	0	8	0	0
8	3	1	0	3	0	3	0	0
9	0	0	5	0	5	0	4	5
10	0	0	9	0	9	0	12	9
11	6	4	7	6	7	6	13	7
12	10	14	2	10	2	10	1	2
13	0	15	0	0	0	0	0	0
14	0	0	0	0	0	0	16	0
15	0	3	0	0	0	0	0	0
16	0	0	0	0	0	0	5	0

Table 1: Transition table for M_{add} for Fibonacci addition

We briefly sketch a proof of the correctness of this automaton. States can be identified with certain sequences, as follows: if x, y, z are the identical-length strings arising from projection of a word that takes M_{add} from the initial state 1 to the state t, then t is identified with the integer sequence $([x0^n]_F + [y0^n]_F - [z0^n]_F)_{n\geq 0}$. With this correspondence, we can verify the following table by a tedious induction. In the table L_n denotes the familiar Lucas numbers, defined by $L_n = F_{n-1} + F_{n+1}$ for $n \geq 0$ (assuming $F_{-1} = 1$). If a sequence $(a_n)_{n\geq 0}$ is the sequence identified with a state t, then t is accepting iff $a_0 = 0$.

state	sequence
1	$\mathbf{0}$
2	$(-F_{n+2})_{n\geq 0}$
3	$(F_{n+2})_{n\geq 0}$
4	$(-F_{n+3})_{n\geq 0}$
5	$(-F_{n+4})_{n\geq 0}$
6	$(-F_{n+1})_{n\geq 0}$
7	$(F_n)_{n\geq 0}$
8	$(F_{n+1})_{n\geq 0}$
9	$(-L_{n+2})_{n\geq 0}$
10	$(-2F_{n+2})_{n\geq 0}$
11	$(-F_n)_{n\geq 0}$
12	$(-2F_{n+1})_{n\geq 0}$
13	$(L_{n+1})_{n\geq 0}$
14	$(-3F_{n+2})_{n\geq 0}$
15	$(2F_{n+1})_{n\geq 0}$
16	$(-2F_n - 3L_n)_{n\geq 0}$

Table 2: Identification of states with sequences

Note that the state 0 actually represents a set of sequences, not just a single sequence. The set corresponds to those representations that are so far "out of synch" that they can never "catch up" to have $x + y = z$, no matter how many digits are appended.

Remark 2. We note that, in the spirit of the paper, this adder itself can, in principle, be checked mechanically (in $\mathrm{Th}(\mathbb{N}, 0)$, of course!), as follows:

First we show the adder $\mathcal{A}$ is specifying a function of x and y. To do so, it suffices to check that

$$\forall x\ \forall y\ \exists z\ \mathcal{A}(x, y, z)$$

and

$$\forall x\ \forall y\ \forall z\ \forall z'\ \mathcal{A}(x, y, z) \wedge \mathcal{A}(x, y, z')\ \Longrightarrow\ z = z'.$$

The first predicate says that there is at least one sum of x and y and the second says that there is at most one.

If both of these are verified, we know that $\mathcal{A}$ computes a function $A = A(x, y)$.

Next, we verify associativity, which amounts to checking that

$$\forall x\ \forall y\ \forall z\ A(A(x, y), z) = A(x, A(y, z)).$$

We can do this by checking that

$$\forall x\ \forall y\ \forall z\ \forall w\ \forall r\ \forall s\ \forall t\ (\mathcal{A}(x, y, r)\ \wedge\ \mathcal{A}(r, z, t)\ \wedge\ \mathcal{A}(y, z, s))\ \Longrightarrow\ \mathcal{A}(x, s, t).$$

Finally, we ensure that $\mathcal{A}$ is an adder by induction. First, we check that $\forall x\ A(x,0) = x$, which amounts to

$$\forall x\ \forall y\ \mathcal{A}(x,0,y) \iff x = y.$$

Second, we check that if $A(x,1) = y$ then $x < y$ and there does not exist z such that $x < z < y$. This amounts to

$$\forall x, y, \mathcal{A}(x,1,y) \implies ((x < y) \wedge \neg\exists z\ (x < z) \wedge (z < y)).$$

This last condition shows that $A(x,1) = x{+}1$. By associativity $A(x, y{+}1) = A(x, A(y,1)) = A(A(x,y),1) = A(x,y) + 1$. By induction, $A(x,y) = A(x,0) + y = x + y$, so we are done.

Another basic fact about Fibonacci representation is that, for canonical representations containing no two consecutive 1's or leading zeroes, the radix order on representations is the same as the ordinary ordering on $\mathbb{N}$. It follows that a very simple automaton can, on input $(x,y)_F$, decide whether $x < y$.

Putting this all together, we get the analogue of Theorem 1:

Procedure 3 (Decision procedure for Fibonacci-automatic words)**.**
Input: $m, n \in \mathbb{N}$, m DFAOs witnessing Fibonacci-automatic words $\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_m$, a first-order proposition with n free variables $\varphi(v_1, v_2, \ldots, v_n)$ using constants and relations definable in $\mathrm{Th}(\mathbb{N}, 0, 1, +)$ and indexing into $\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_m$.
Output: DFA with input alphabet Σ_2^n accepting $\{(k_1, k_2, \ldots, k_n)_F\ :\ \varphi(k_1, k_2, \ldots, k_n)\ \text{holds}\}$.

We remark that there was substantial skepticism that any implementation of a decision procedure for Fibonacci-automatic words would be practical, for two reasons:

- first, because the running time is bounded above by an expression of the form

  $$2^{2^{\cdot^{\cdot^{\cdot^{2^{p(N)}}}}}}$$

  where p is a polynomial, N is the number of states in the original automaton specifying the word in question, and the number of exponents in the tower is one less than the number of quantifiers in the logical formula characterizing the property being checked.

- second, because of the complexity of checking addition (15 states) compared to the analogous automaton for base-k representation (2 states).

Nevertheless, we were able to carry out nearly all the computations described in this paper in a matter of a few seconds on an ordinary laptop.

3 Mechanical proofs of properties of the infinite Fibonacci word

Recall that a word x, whether finite or infinite, is said to have period p if $x[i] = x[i+p]$ for all i for which this equality is meaningful. Thus, for example, the English word `alfalfa` has

period 3. The *exponent* of a finite word x, written $\exp(x)$, is $|x|/P$, where P is the smallest period of x. Thus $\exp(\texttt{alfalfa}) = 7/3$.

If $\mathbf{x}$ is an infinite word with a finite period, we say it is *ultimately periodic*. An infinite word $\mathbf{x}$ is ultimately periodic if and only if there are finite words u, v such that $x = uv^\omega$, where $v^\omega = vvv\cdots$.

A nonempty word of the form xx is called a *square*, and a nonempty word of the form xxx is called a *cube*. More generally, a nonempty word of the form x^n is called an n'th power. By the *order* of a square xx, cube xxx, or n'th power x^n, we mean the length $|x|$.

The infinite Fibonacci word $\mathbf{f} = 01001010\cdots = f_0 f_1 f_2 \cdots$ can be described in many different ways. In addition to our definition in terms of automata, it is also the fixed point of the morphism $\varphi(0) = 01$ and $\varphi(1) = 0$. This word has been studied extensively in the literature; see, for example, [4, 6].

In the next subsection, we use our implementation to prove a variety of results about repetitions in $\mathbf{f}$.

3.1 Repetitions

Theorem 4. *The word* $\mathbf{f}$ *is not ultimately periodic.*

Proof. We construct a predicate asserting that the integer $p \geq 1$ is a period of some suffix of $\mathbf{f}$:

$$(p \geq 1) \ \wedge \ \exists n \ \forall i \geq n \ \mathbf{f}[i] = \mathbf{f}[i+p].$$

(Note: unless otherwise indicated, whenever we refer to a variable in a predicate, the range of the variable is assumed to be $\mathbb{N} = \{0, 1, 2, \ldots\}$.) From this predicate, using our program, we constructed an automaton accepting the language

$$L = 0^* \ \{(p)_F \ : \ (p \geq 1) \ \wedge \ \exists n \ \forall i \geq n \ \mathbf{f}[i] = \mathbf{f}[i+p]\}.$$

This automaton accepts the empty language, and so it follows that $\mathbf{f}$ is not ultimately periodic.

Here is the log of our program:

```
p >= 1 with 4 states, in 60ms
 i >= n with 7 states, in 5ms
  F[i] = F[i + p] with 12 states, in 34ms
   i >= n => F[i] = F[i + p] with 51 states, in 15ms
    Ai i >= n => F[i] = F[i + p] with 3 states, in 30ms
     p >= 1 & Ai i >= n => F[i] = F[i + p] with 2 states, in 0ms
      En p >= 1 & Ai i >= n => F[i] = F[i + p] with 2 states, in 0ms
overall time: 144ms
```

The largest intermediate automaton during the computation had 63 states.

A few words of explanation are in order: here "`F`" refers to the sequence $\mathbf{f}$, and "`E`" is our abbreviation for $\exists$ and "`A`" is our abbreviation for $\forall$. The symbol "`=>`" is logical implication, and "`&`" is logical and. □

From now on, whenever we discuss the language accepted by an automaton, we will omit the 0^* at the beginning.

We recall an old result of Karhumäki [37, Thm. 2]:

Theorem 5. **f** *contains no fourth powers.*

Proof. We create a predicate for the orders of all fourth powers occurring in **f**:

$$(n > 0) \ \wedge \ \exists i \ \forall t < 3n \ \mathbf{f}[i + t] = \mathbf{f}[i + n + t].$$

The resulting automaton accepts nothing, so there are no fourth powers.

```
n > 0 with 4 states, in 46ms
 t < 3 * n with 30 states, in 178ms
  F[i + t] = F[i + t + n] with 62 states, in 493ms
   t < 3 * n => F[i + t] = F[i + t + n] with 352 states, in 39ms
    At t < 3 * n => F[i + t] = F[i + t + n] with 3 states, in 132ms
     Ei At t < 3 * n => F[i + t] = F[i + t + n] with 2 states, in 0ms
      n > 0 & Ei At t < 3 * n => F[i + t] = F[i + t + n] with 2 states, in 0ms
overall time: 888ms
```

□

The largest intermediate automaton in the computation had 952 states.

Next, we move on to a description of the orders of squares occurring in **f**. An old result of Séébold [48] (also see [36, 26]) states

Theorem 6. *All squares in* **f** *are of order F_n for some $n \geq 2$. Furthermore, for all $n \geq 2$, there exists a square of order F_n in* **f**.

Proof. We create a predicate for the lengths of squares:

$$(n > 0) \ \wedge \ \exists i \ \forall t < n \ \mathbf{f}[i + t] = \mathbf{f}[i + n + t].$$

When we run this predicate, we obtain an automaton that accepts exactly the language 10^*. Here is the log file:

```
n > 0 with 4 states, in 38ms
 t < n with 7 states, in 5ms
  F[i + t] = F[i + t + n] with 62 states, in 582ms
   t < n => F[i + t] = F[i + t + n] with 92 states, in 12ms
    At t < n => F[i + t] = F[i + t + n] with 7 states, in 49ms
     Ei At t < n => F[i + t] = F[i + t + n] with 3 states, in 1ms
      n > 0 & Ei At t < n => F[i + t] = F[i + t + n] with 3 states, in 0ms
overall time: 687ms
```

□

The largest intermediate automaton had 236 states.

We can easily get much, much more information about the square occurrences in $\mathbf{f}$. The positions of all squares in $\mathbf{f}$ were computed by Iliopoulos, Moore, and Smyth [36, § 2], but their description is rather complicated and takes 5 pages to prove. Using our approach, we created an automaton accepting the language

$$\{(n,i)_F \ : \ (n>0) \ \wedge \ \forall t < n \ \mathbf{f}[i+t] = \mathbf{f}[i+n+t]\}.$$

This automaton has only 6 states and efficiently encodes the orders and starting positions of each square in $\mathbf{f}$. During the computation, the largest intermediate automaton had 236 states. Thus we have proved

Theorem 7. *The language*

$$\{(n,i)_F \ : \ \textit{there is a square of order } n \textit{ beginning at position } i \textit{ in } \mathbf{f}\}$$

is accepted by the automaton in Figure 2.

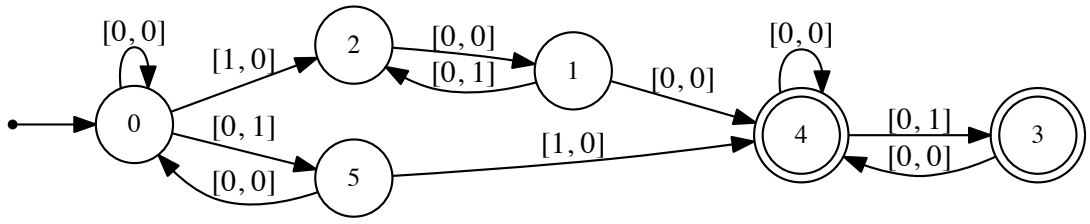

Figure 2: Automaton accepting orders and positions of all squares in $\mathbf{f}$

Next, we examine the cubes in $\mathbf{f}$. Evidently Theorem 6 implies that any cube in $\mathbf{f}$ must be of order F_n for some n. However, not every order occurs.

Theorem 8. *The cubes in* $\mathbf{f}$ *are of order* F_n *for* $n \geq 4$*, and a cube of each such order occurs.*

Proof. We use the predicate

$$(n>0) \ \wedge \ \exists i \ \forall t < 2n \ \mathbf{f}[i+t] = \mathbf{f}[i+n+t].$$

When we run our program, we obtain an automaton accepting exactly the language $(100)0^*$, which corresponds to F_n for $n \geq 4$.

```
n > 0 with 4 states, in 34ms
 t < 2 * n with 16 states, in 82ms
  F[i + t] = F[i + t + n] with 62 states, in 397ms
   t < 2 * n => F[i + t] = F[i + t + n] with 198 states, in 17ms
    At t < 2 * n => F[i + t] = F[i + t + n] with 7 states, in 87ms
     Ei At t < 2 * n => F[i + t] = F[i + t + n] with 5 states, in 1ms
      n > 0 & Ei At t < 2 * n => F[i + t] = F[i + t + n] with 5 states, in 0ms
overall time: 618ms
```

□

The largest intermediate automaton had 674 states.

Next, we encode the orders and positions of all cubes. We build a DFA accepting the language

$$\{(n,i)_F \ : \ (n>0) \ \wedge \ \forall t < 2n \ \mathbf{f}[i+t] = \mathbf{f}[i+n+t]\}.$$

Theorem 9. *The language*

$$\{(n,i)_F \ : \ \textit{there is a cube of order } n \textit{ beginning at position } i \textit{ in } \mathbf{f}\}$$

is accepted by the automaton in Figure 3.

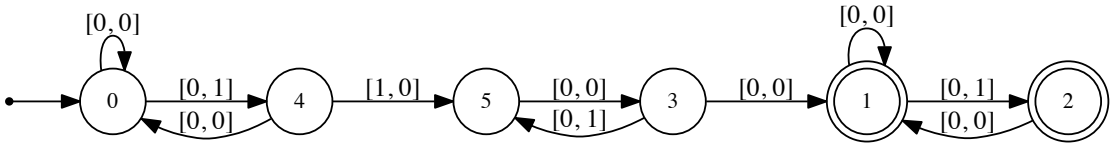

Figure 3: Automaton accepting orders and positions of all cubes in $\mathbf{f}$

Finally, we consider all the maximal repetitions in $\mathbf{f}$. Let $p(x)$ denote the length of the least period of x. If $\mathbf{x} = a_0 a_1 \cdots$, by $\mathbf{x}[i..j]$ we mean $a_i a_{i+1} \cdots a_j$. Following Kolpakov and Kucherov [38], 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 $p(\mathbf{f}[i..i+n-1]) < p(\mathbf{f}[i-1..i+n-1])$.

Theorem 10. *The factor* $\mathbf{f}[i..i+n-1]$ *is a maximal repetition of* $\mathbf{f}$ *iff* $(n,i)_F$ *is accepted by the automaton depicted in Figure 4.*

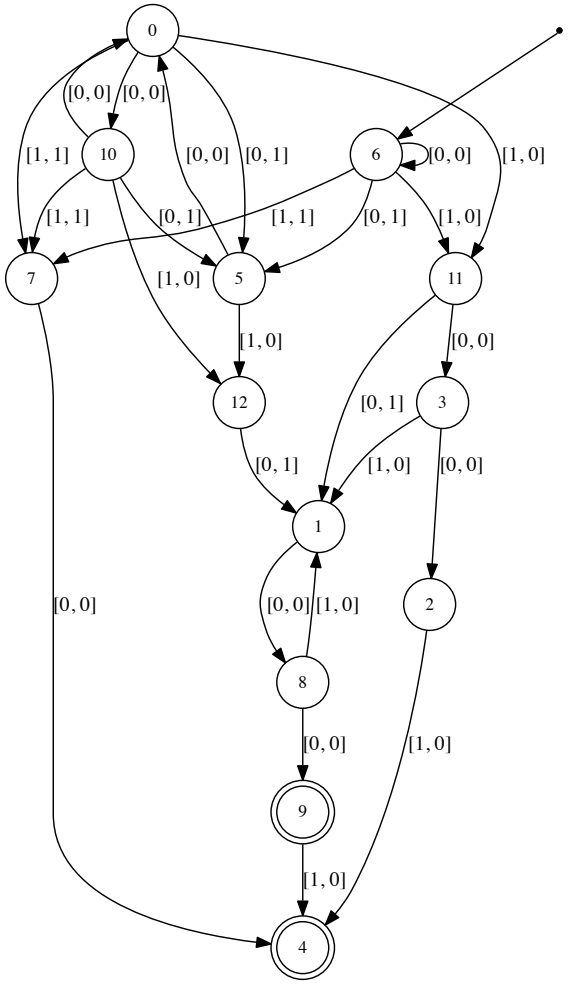

Figure 4: Automaton accepting occurrences of maximal repetitions in $\mathbf{f}$

An *antisquare* is a nonempty word of the form $x\overline{x}$, where $\overline{x}$ denotes the complement of x (1's changed to 0's and vice versa). Its order is $|x|$. For a new (but small) result we prove

Theorem 11. *The Fibonacci word* $\mathbf{f}$ *contains exactly four antisquare factors:* $01, 10, 1001$, *and* 10100101.

Proof. The predicate for having an antisquare of length n is

$$\exists i\ \forall k < n\ \mathbf{f}[i+k] \neq \mathbf{f}[i+k+n].$$

When we run this we get the automaton depicted in Figure 5, specifying that the only possible orders are 1, 2, and 4, which correspond to words of length 2, 4, and 8.

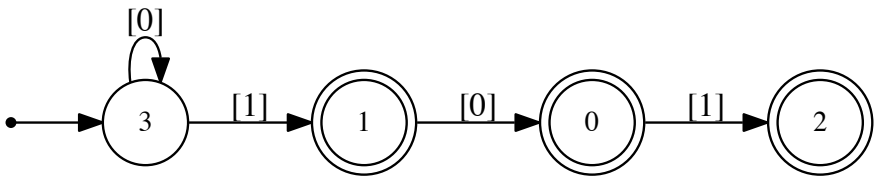

Figure 5: Automaton accepting orders of antisquares in $\mathbf{f}$

Inspection of the factors of these lengths proves the result. □

3.2 Palindromes and antipalindromes

We now turn to a characterization of the palindromes in $\mathbf{f}$. Using the predicate

$$\exists i \ \forall j < n \ \mathbf{f}[i+j] = \mathbf{f}[i+n-1-j],$$

we specify those lengths n for which there is a palindrome of length n. Our program then recovers the following result of Chuan [15]:

Theorem 12. *There exist palindromes of every length* ≥ 0 *in* $\mathbf{f}$.

We could also characterize the positions of all nonempty palindromes. The resulting 21-state automaton is not particularly enlightening, but is included here to show the kind of complexity that can arise.

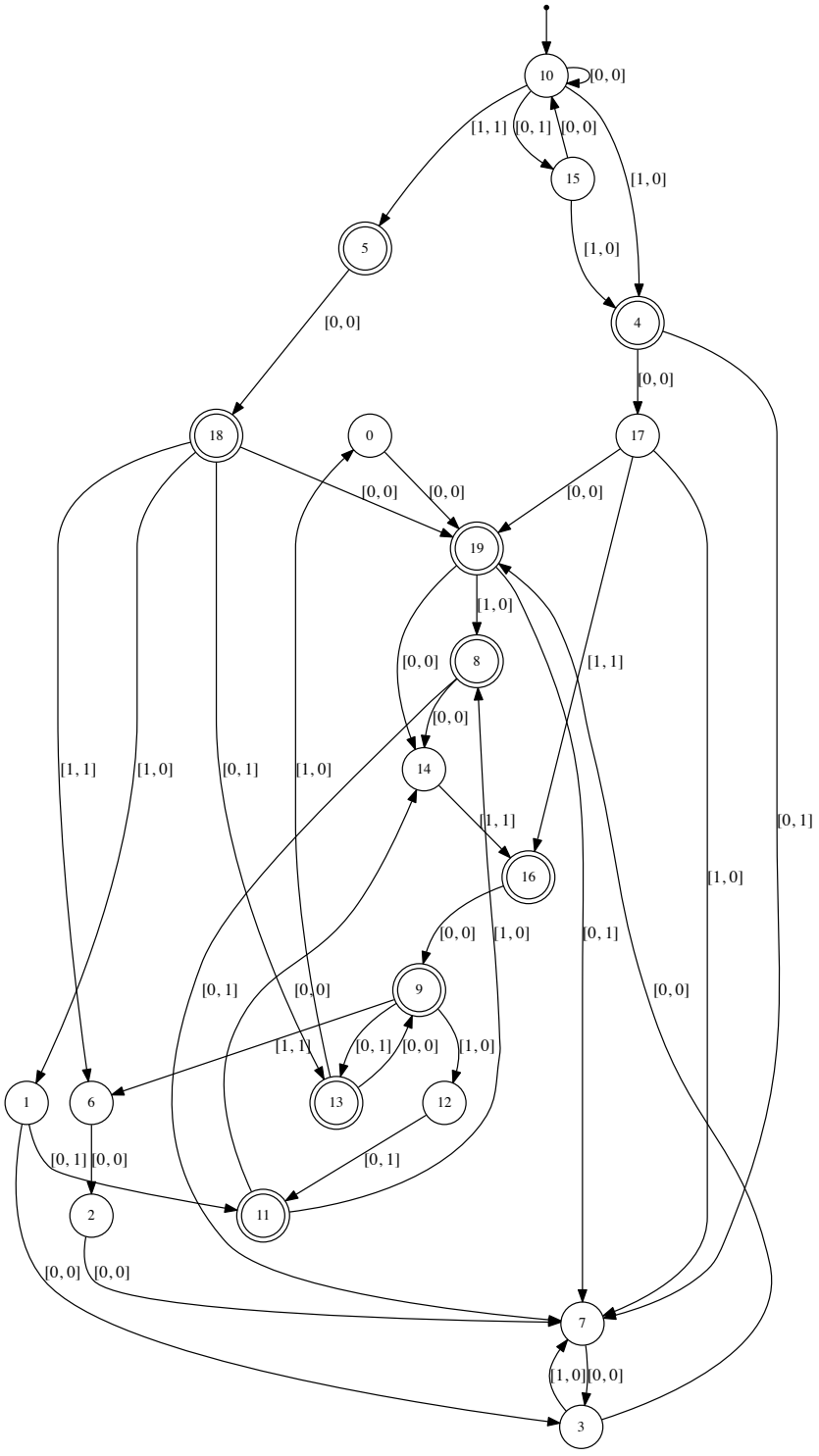

Figure 6: Automaton accepting orders and positions of all nonempty palindromes in **f**

Although the automaton in Figure 6 encodes all palindromes, more specific information is a little hard to deduce from it. For example, let's prove a result of Droubay [21]:

Theorem 13. *The Fibonacci word* $\mathbf{f}$ *has exactly one palindromic factor of length* n *if* n *is even, and exactly two palindromes of length* n *if* n *odd.*

Proof. First, we obtain an expression for the lengths n for which there is exactly one palindromic factor of length n.

$$\exists i\ (\forall t < n\ \mathbf{f}[i+t] = \mathbf{f}[i+n-1-t]) \wedge \\ \forall j\ (\forall s < n\ \mathbf{f}[j+s] = \mathbf{f}[j+n-1-s]) \implies (\forall u < n\ \mathbf{f}[i+u] = \mathbf{f}[j+u])$$

The first part of the predicate asserts that $\mathbf{f}[i..i+n-1]$ is a palindrome, and the second part asserts that any palindrome $\mathbf{f}[j..j+n-1]$ of the same length must in fact be equal to $\mathbf{f}[i..i+n-1]$.

When we run this predicate through our program we get the automaton depicted below in Figure 7.

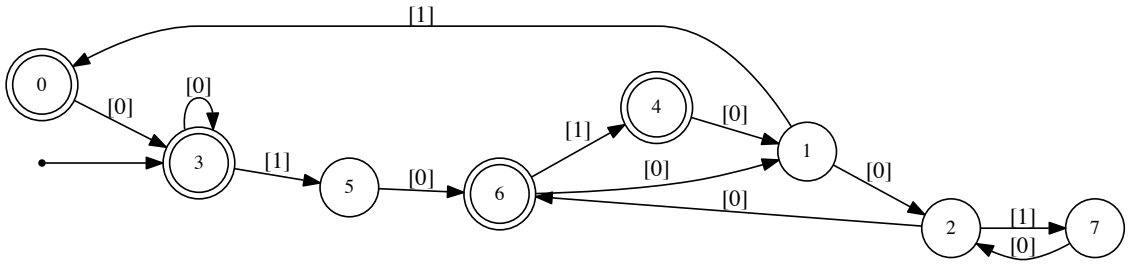

Figure 7: Automaton accepting lengths with exactly one palindrome

It may not be obvious, but this automaton accepts exactly the Fibonacci representations of the even numbers. The easiest way to check this is to use our program on the predicate $\exists i\ n = 2i$ and verify that the resulting automaton is isomorphic to that in Figure 7.

Next, we write down a predicate for the existence of exactly two distinct palindromes of length n. The predicate asserts the existence of two palindromes $\mathbf{x}[i..i+n-1]$ and $\mathbf{x}[j..j+n-1]$ that are distinct and for which any palindrome of the same length must be equal to one of them.

$$\exists i\ \exists j\ (\forall t < n\ \mathbf{f}[i+t] = \mathbf{f}[i+n-1-t])\ \wedge\ (\forall s < n\ \mathbf{f}[j+s] = \mathbf{f}[j+n-1-s]) \wedge \\ (\exists m < n\ \mathbf{f}[i+m] \neq \mathbf{f}[j+m]) \wedge \\ (\forall u(\forall k < n\ \mathbf{f}[u{+}k] = \mathbf{f}[u{+}n{-}1{-}k]) \implies ((\forall l < n\ \mathbf{f}[u{+}l] = \mathbf{f}[i{+}l]) \vee (\forall p < n\ \mathbf{f}[u{+}p] = \mathbf{f}[j{+}p])))$$

Again, running this through our program gives us an automaton accepting the Fibonacci representations of the odd numbers. We omit the automaton. □

The prefixes are factors of particular interest. Let us determine which prefixes are palindromes:

Theorem 14. *The prefix* $\mathbf{f}[0..n-1]$ *of length* n *is a palindrome if and only if* $n = F_i - 2$ *for some* $i \geq 3$.

Proof. We use the predicate

$$\forall i < n \; \mathbf{f}[i] = \mathbf{f}[n-1-i]$$

obtaining an automaton accepting $\epsilon + 1 + 10(10)^*(0 + 01)$, which are precisely the representations of $F_i - 2$. □

Next, we turn to the property of "mirror invariance". We say an infinite word $\mathbf{w}$ is mirror-invariant if whenever x is a factor of $\mathbf{w}$, then so is x^R. We can check this for $\mathbf{f}$ by creating a predicate for the assertion that for each factor x of length n, the factor x^R appears somewhere else:

$$\forall i \geq 0 \; \exists j \text{ such that } \mathbf{f}[i..i+n-1] = \mathbf{f}[j..j+n-1]^R.$$

When we run this through our program we discover that it accepts the representations of all $n \geq 0$. Here is the log:

```
t < n with 7 states, in 99ms
 F[i + t] = F[j + n - 1 - t] with 264 states, in 7944ms
  t < n => F[i + t] = F[j + n - 1 - t] with 185 states, in 89ms
   At t < n => F[i + t] = F[j + n - 1 - t] with 35 states, in 182ms
    Ej At t < n => F[i + t] = F[j + n - 1 - t] with 5 states, in 2ms
     Ai Ej At t < n => F[i + t] = F[j + n - 1 - t] with 3 states, in 6ms
overall time: 8322ms
```

Thus we have proved:

Theorem 15. *The word* $\mathbf{f}$ *is mirror invariant.*

An *antipalindrome* is a word x satisfying $x = \overline{x^R}$. For a new (but small) result, we determine all possible antipalindromes in $\mathbf{f}$:

Theorem 16. *The only nonempty antipalindromes in* $\mathbf{f}$ *are* 01, 10, $(01)^2$, *and* $(10)^2$.

Proof. Let us write a predicate specifying that $\mathbf{f}[i..i+n-1]$ is a nonempty antipalindrome, and further that it is a first occurrence of such a factor:

$$(n > 0) \; \wedge \; (\forall j < n \; \mathbf{f}[i+j] \neq \mathbf{f}[i+n-1-j]) \; \wedge \; (\forall i' < i \; \exists j < n \; \mathbf{f}[i'+j] \neq \mathbf{f}[i+j]).$$

When we run this through our program, the language of $(n, i)_F$ satisfying this predicate is accepted by the following automaton:

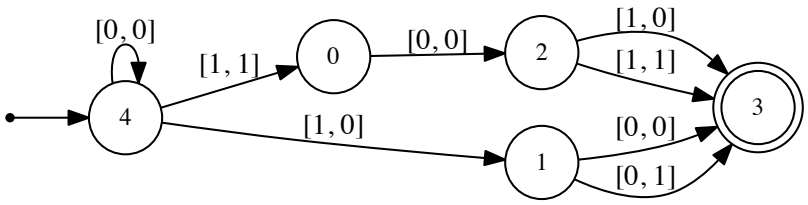

Figure 8: Automaton accepting orders and positions of first occurrences of nonempty antipalindromes in $\mathbf{f}$

It follows that the only (n, i) pairs accepted are $(2, 0), (2, 1), (4, 3), (4, 4)$, corresponding, respectively, to the strings 01, 10, $(01)^2$, and $(10)^2$. □

3.3 Special factors

Next we turn to special factors. It is well-known that $\mathbf{f}$ has exactly $n + 1$ distinct factors of length n for each $n \geq 0$. This implies that there is exactly one factor x of each length n with the property that both $x0$ and $x1$ are factors. Such a factor is called *right-special* or sometimes just *special*. We can write a predicate that expresses the assertion that the factor $\mathbf{f}[i..i + n - 1]$ is the unique special factor of length n, and furthermore, that it is the first occurrence of that factor, as follows:

$$(\forall i' < i \; \exists s < n \; \mathbf{f}[i' + s] \neq \mathbf{f}[i + s]) \; \wedge \; \exists j \; \exists k \; ((\forall t < n \; \mathbf{f}[j + t] = \mathbf{f}[i + t]) \\ \wedge \; (\forall u < n \; \mathbf{f}[k + u] = \mathbf{f}[i + u]) \; \wedge \; (\mathbf{f}[j + n] \neq \mathbf{f}[k + n])).$$

Theorem 17. *The automaton depicted below in Figure 9 accepts the language*

$\{(i, n)_F$: *the factor* $\mathbf{f}[i..i+n-1]$ *is the first occurrence of the unique special factor of length* $n\}$.

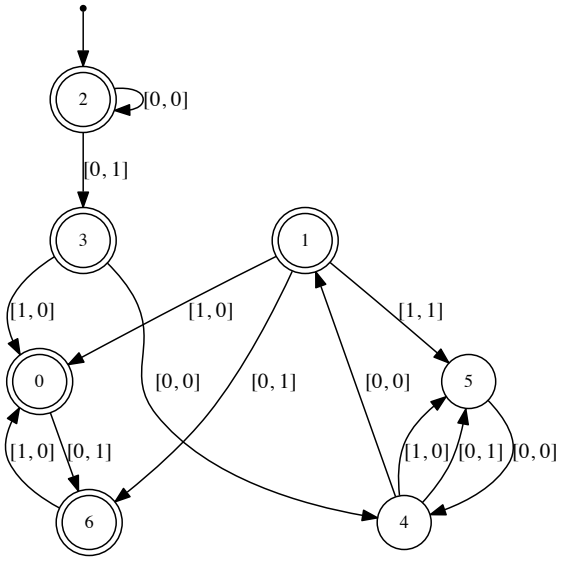

Figure 9: Automaton accepting first positions and lengths of special factors in $\mathbf{f}$

Furthermore it is known (e.g., [42, Lemma 5]) that

Theorem 18. *The unique special factor of length n is $\mathbf{f}[0..n-1]^R$.*

Proof. We create a predicate that says that if a factor is special then it matches $\mathbf{f}[0..n-1]^R$. When we run this we discover that all lengths are accepted. □

3.4 Least periods

We now turn to least periods of factors of $\mathbf{f}$; see [45] and [24] and [19, Corollary 4].

Let P denote the assertion that n is a period of the factor $\mathbf{f}[i..j]$, as follows:

$$\begin{aligned} P(n,i,j) &= \mathbf{f}[i..j-n] = \mathbf{f}[i+n..j] \\ &= \forall\ t \ \text{ with } i \le t \le j-n \text{ we have } \mathbf{f}[t] = \mathbf{f}[t+n]. \end{aligned}$$

Using this, we can express the predicate LP that n is the least period of $\mathbf{f}[i..j]$:

$$LP(n,i,j) = P(n,i,j) \text{ and } \forall n' \text{ with } 1 \le n' < n\ \neg P(n',i,j).$$

Finally, we can express the predicate that n is a least period as follows

$$L(n) = \exists i,j \ge 0 \text{ with } 0 \le i+n \le j-1\ LP(n,i,j).$$

Using an implementation of this, we can reprove the following theorem of Saari [45, Thm. 2]:

Theorem 19. *If a word w is a nonempty factor of the Fibonacci word, then the least period of w is a Fibonacci number F_n for $n \geq 2$. Furthermore, each such period occurs.*

Proof. We ran our program on the appropriate predicate and found the resulting automaton accepts 10^+, corresponding to F_n for $n \geq 2$. □

Furthermore, we can actually encode information about all least periods. The automaton depicted in Figure 10 accepts triples (n, p, i) such that p is a least period of $\mathbf{f}[i..i+n-1]$.

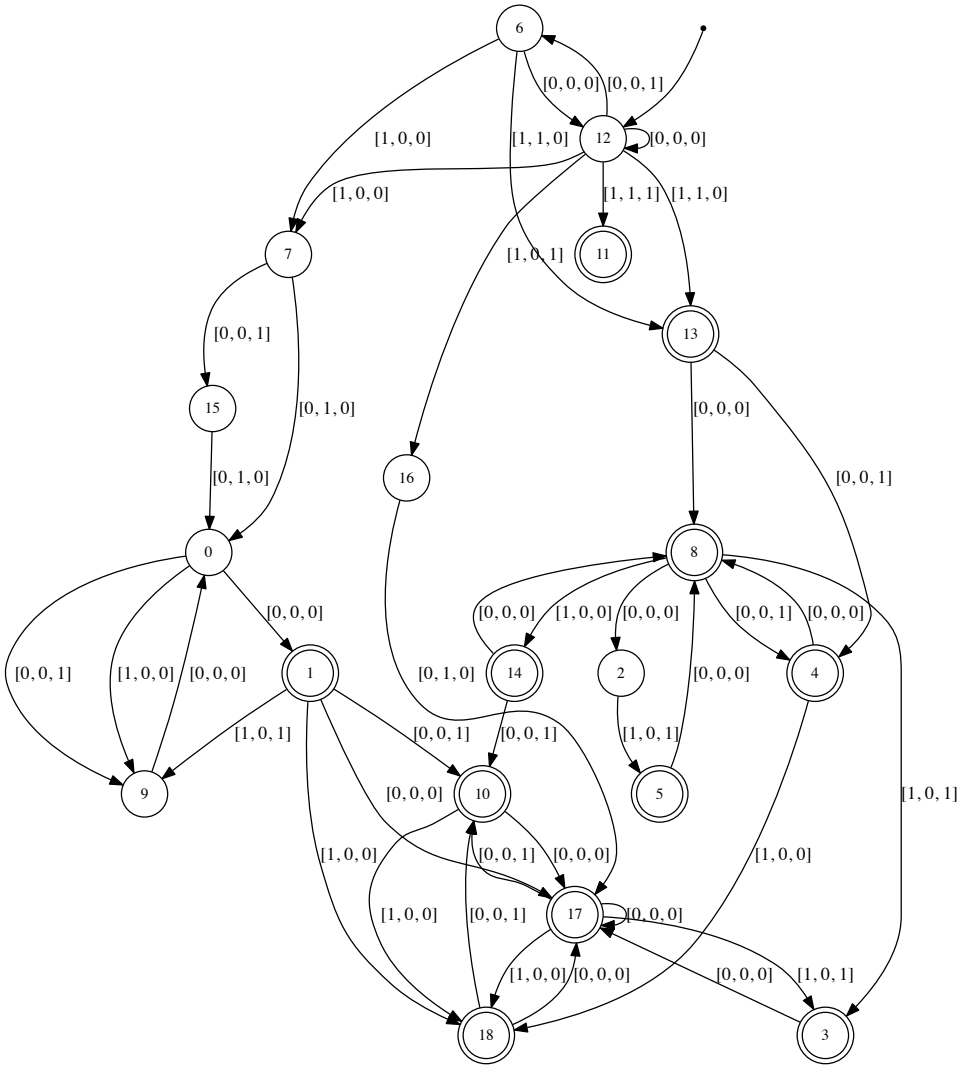

Figure 10: Automaton encoding least periods of all factors in $\mathbf{f}$

We also have the following result, which seems to be new.

Theorem 20. *Let $n \geq 1$, and define $\ell(n)$ to be the smallest integer that is the least period of some length-n factor of* $\mathbf{f}$*. Then $\ell(n) = F_j$ for $j \geq 1$ if $L_j - 1 \leq n \leq L_{j+1} - 2$, where L_j is the j'th Lucas number defined in Section 2.*

Proof. We create an automaton accepting $(n, p)_F$ such that (a) there exists at least one length-n factor of period p and (b) for all length-n factors x, if q is a period of x, then $q \geq p$. This automaton is depicted in Figure 11 below.

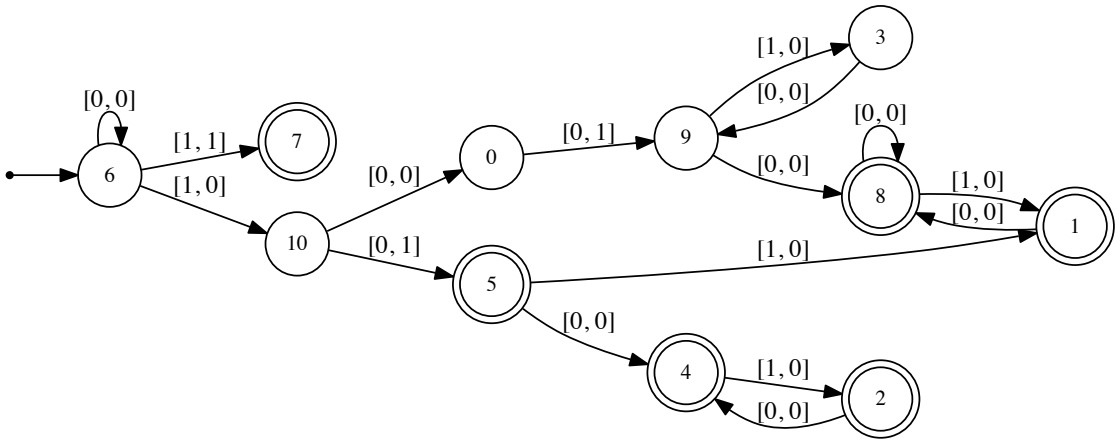

Figure 11: Automaton encoding smallest period over all length-n factors in $\mathbf{f}$

The result now follows by inspection and the fact that $(L_j - 1)_F = 10(01)^{(j-2)/2}$ if $j \geq 2$ is even, and $100(10)^{(j-3)/2}$ if $j \geq 3$ is odd. □

3.5 Quasiperiods

We now turn to quasiperiods. An infinite word $\mathbf{a}$ is said to be *quasiperiodic* if there is some finite nonempty word x such that $\mathbf{a}$ can be completely "covered" with translates of x. Here we study the stronger version of quasiperiodicity where the first copy of x used must be aligned with the left edge of $\mathbf{w}$ and is not allowed to "hang over"; these are called *aligned covers* in [14]. More precisely, for us $\mathbf{a} = a_0 a_1 a_2 \cdots$ is quasiperiodic if there exists x such that for all $i \geq 0$ there exists $j \geq 0$ with $i - n < j \leq i$ such that $a_j a_{j+1} \cdots a_{j+n-1} = x$, where $n = |x|$. Such an x is called a *quasiperiod*. Note that the condition $j \geq 0$ implies that, in this interpretation, any quasiperiod must actually be a prefix of $\mathbf{a}$.

The quasiperiodicity of the Fibonacci word $\mathbf{f}$ was studied by Christou, Crochemore, and Iliopoulos [14], where we can (more or less) find the following theorem:

Theorem 21. *A nonempty length-n prefix of* $\mathbf{f}$ *is a quasiperiod of* $\mathbf{f}$ *if and only if n is not of the form $F_n - 1$ for $n \geq 3$.*

In particular, the following prefix lengths are not quasiperiods: 1, 2, 4, 7, 12, and so forth.

Proof. We write a predicate for the assertion that the length-n prefix is a quasiperiod:

$$\forall i \geq 0 \; \exists j \text{ with } i - n < j \leq i \text{ such that } \forall t < n \; \mathbf{f}[t] = \mathbf{f}[j + t].$$

When we do this, we get the automaton in Figure 12 below. Inspection shows that this DFA accepts all canonical representations, except those of the form $1(01)^*(\epsilon + 0)$, which are precisely the representations of $F_n - 1$.

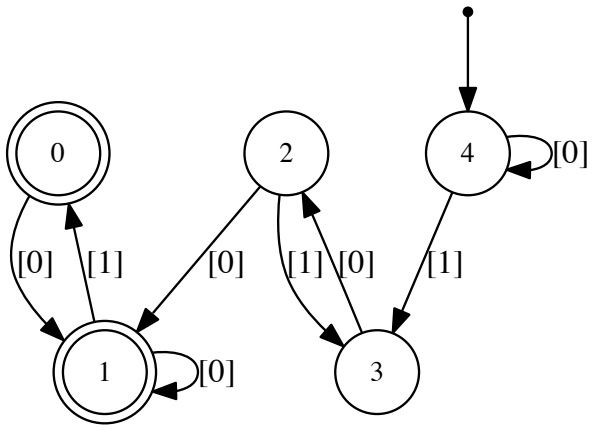

Figure 12: Automaton accepting lengths of prefixes of $\mathbf{f}$ that are quasiperiods

□

3.6 Unbordered factors

Next we look at unbordered factors. A word y is said to be a *border* of x if y is both a nonempty proper prefix and suffix of x. A word x is *bordered* if it has at least one border. It is easy to see that if a word y is bordered iff it has a border of length ℓ with $0 < \ell \leq |y|/2$.

Theorem 22. *The only unbordered nonempty factors of* $\mathbf{f}$ *are of length* F_n *for* $n \geq 2$*, and there are two for each such length. For* $n \geq 3$ *these two unbordered factors have the property that one is a reverse of the other.*

Proof. We can express the property of having an unbordered factor of length n as follows

$$\exists i \; \forall j, 1 \leq j \leq n/2, \; \exists t < j \; \mathbf{f}[i + t] \neq \mathbf{f}[i + n - j + t].$$

Here is the log:

```
j >= 1 with 4 states, in 155ms
 2 * j <= n with 16 states, in 91ms
  j >= 1 & 2 * j <= n with 21 states, in 74ms
   t < j with 7 states, in 17ms
    F[i + t] != F[i + n - j + t] with 321 states, in 10590ms
     t < j & F[i + t] != F[i + n - j + t] with 411 states, in 116ms
      Et t < j & F[i + t] != F[i + n - j + t] with 85 states, in 232ms
       j >= 1 & 2 * j <= n => Et t < j & F[i + t] != F[i + n - j + t] with 137 states, in 19ms
        Aj j >= 1 & 2 * j <= n => Et t < j & F[i + t] != F[i + n - j + t] with 7 states, in 27ms
         Ei Aj j >= 1 & 2 * j <= n => Et t < j & F[i + t] != F[i + n - j + t] with 3 states, in 0ms
overall time: 11321ms
```

The automaton produced accepts the Fibonacci representation of 0 and F_n for $n \geq 2$.

Next, we make the assertion that there are exactly two such factors for each appropriate length. We can do this by saying there is an unbordered factor of length n beginning at position i, another one beginning at position k, and these factors are distinct, and for every unbordered factor of length n, it is equal to one of these two. When we do this we discover that the representations of all F_n for $n \geq 2$ are accepted.

Finally, we make the assertion that for any two unbordered factors of length n, either they are equal or one is the reverse of the other. When we do this we discover all lengths except length 1 are accepted. (That is, for all lengths other than F_n, $n \geq 2$, the assertion is trivially true since there are no unbordered factors; for $F_2 = 1$ it is false since 0 and 1 are the unbordered factors and one is not the reverse of the other; and for all larger F_i the property holds.) □

3.7 Recurrence, uniform recurrence, and linear recurrence

We now turn to various questions about recurrence. A factor x of an infinite word $\mathbf{w}$ is said to be *recurrent* if it occurs infinitely often. The word $\mathbf{w}$ is recurrent if every factor that occurs at least once is recurrent. A factor x is *uniformly recurrent* if there exists a constant $c = c(x)$ such that any factor $\mathbf{w}[i..i + c]$ is guaranteed to contain an occurrence of x. If all factors are uniformly recurrent then $\mathbf{w}$ is said to be uniformly recurrent. Finally, $\mathbf{w}$ is *linearly recurrent* if the constant $c(x)$ is $O(|x|)$.

Theorem 23. *The word* $\mathbf{f}$ *is recurrent, uniformly recurrent, and linearly recurrent.*

Proof. A predicate for all length-n factors being recurrent:

$$\forall i \geq 0\ \forall j \geq 0\ \exists k > j\ \forall t < n\ \mathbf{f}[i + t] = \mathbf{f}[k + t].$$

This predicate says that for every factor $z = \mathbf{f}[i..i + n - 1]$ and every position j we can find another occurrence of z beginning at a position $k > j$. When we run this we discover that the representations of all $n \geq 0$ are accepted. So $\mathbf{f}$ is recurrent.

A predicate for uniform recurrence:

$$\forall i\ \exists \ell\ \forall j\ \exists s,\ j \leq s \leq j + l - n\ \forall p < n\ \mathbf{f}[s + p] = \mathbf{f}[i + p].$$

Once again, when we run this we discover that the representations of all $n \geq 0$ are accepted. So $\mathbf{f}$ is uniformly recurrent.

A predicate for linear recurrence with constant C:

$$\forall i\ \forall j\ \exists s,\ j \leq s \leq j + Cn\ \forall p < n\ \mathbf{f}[s+p] = \mathbf{f}[i+p].$$

When we run this with $C = 4$, we discover that the representations of all $n \geq 0$ are accepted (but, incidentally, not for $C = 3$). So $\mathbf{f}$ is linearly recurrent. □

Remark 24. We can decide the property of linear recurrence for Fibonacci-automatic words even without knowing an explicit value for the constant C. The idea is to accept those pairs (n, t) such that there exists a factor of length n with two consecutive occurrences separated by distance t. Letting S denote the set of such pairs, then a sequence is linearly recurrent iff $\limsup_{(n,t)\in S} t/n < \infty$, which can be decided using an argument like that in [47, Thm. 8]. However, we do not know how to compute, in general, the exact value of the lim sup for Fibonacci representation (which we do indeed know for base-k representation), although we can approximate it arbitrarily closely.

3.8 Lyndon words

Next, we turn to some results about Lyndon words. Recall that a nonempty word x is a *Lyndon word* if it is lexicographically less than all of its nonempty proper prefixes.[1] We reprove some recent results of Currie and Saari [19] and Saari [46].

Theorem 25. *Every Lyndon factor of* $\mathbf{f}$ *is of length* F_n *for some* $n \geq 2$*, and each of these lengths has a Lyndon factor.*

Proof. Here is the predicate specifying that there is a factor of length n that is Lyndon:

$$\exists i\ \forall j, 1 \leq j < n,\ \exists t < n - j\ (\forall u < t\ \mathbf{f}[i+u] = \mathbf{f}[i+j+u])\ \wedge\ \mathbf{f}[i+t] < \mathbf{f}[i+j+t].$$

When we run this we get the representations 10^*, which proves the result. □

Theorem 26. *For* $n \geq 2$*, every length-*n *Lyndon factor of* $\mathbf{f}$ *is a conjugate of* $\mathbf{f}[0..n-1]$.

Proof. Using the predicate from the previous theorem as a base, we can create a predicate specifying that every length-n Lyndon factor is a conjugate of $\mathbf{f}[0..n-1]$. When we do this we discover that all lengths except 1 are accepted. (The only lengths having a Lyndon factor are F_n for $n \geq 2$, so all but F_2 have the desired property.) □

[1] There is also a version where "prefixes" is replaced by "suffixes".

3.9 Critical exponents

Recall from Section 3 that $\exp(w) = |w|/P$, where P is the smallest period of w. The *critical exponent* of an infinite word $\mathbf{x}$ is the supremum, over all factors w of $\mathbf{x}$, of $\exp(w)$.

A classic result of [40] is

Theorem 27. *The critical exponent of* $\mathbf{f}$ *is* $2 + \alpha$*, where* $\alpha = (1 + \sqrt{5})/2$.

Although it is known that the critical exponent is computable for k-automatic sequences [47], we do not yet know this for Fibonacci-automatic sequences (and more generally Pisot-automatic sequences). However, with a little inspired guessing about the maximal repetitions, we can complete the proof.

Proof. For each length n, the smallest possible period p of a factor is given by Theorem 20. Hence the critical exponent is given by $\lim_{j\to\infty}(L_{j+1} - 2)/F_j$, which is $2 + \alpha$. □

We can also ask the same sort of questions about the *initial critical exponent* of a word $\mathbf{w}$, which is the supremum over the exponents of all prefixes of $\mathbf{w}$.

Theorem 28. *The initial critical exponent of* $\mathbf{f}$ *is* $1 + \alpha$.

Proof. We create an automaton M_{ice} accepting the language

$$L = \{(n, p)_F \ : \ \mathbf{f}[0..n-1] \text{ has least period } p\}.$$

It is depicted in Figure 13 below. From the automaton, it is easy to see that the least period of the prefix of length $n \geq 1$ is F_j for $j \geq 2$ and $F_{j+1} - 1 \leq n \leq F_{j+2} - 2$. Hence the initial critical exponent is given by $\limsup_{j\to\infty}(F_{j+2} - 2)/F_j$, which is $1 + \alpha$.

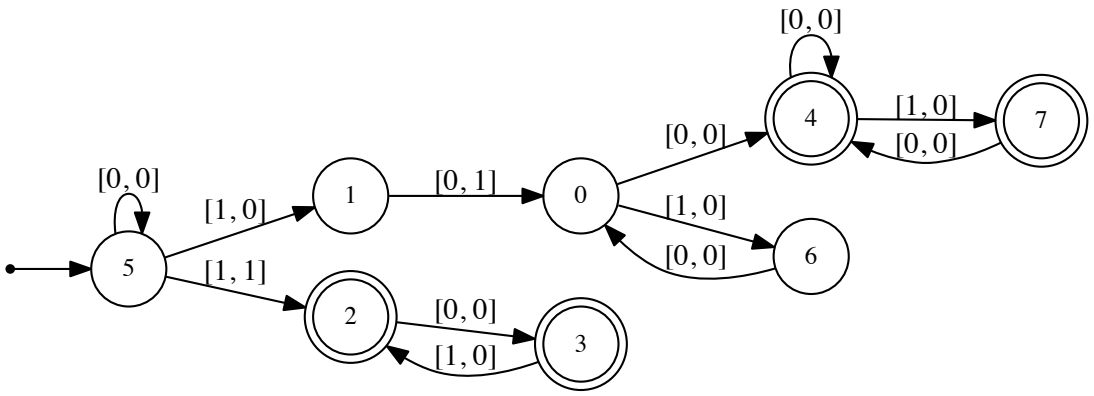

Figure 13: Automaton accepting least periods of prefixes of length n

□

3.10 The shift orbit closure

The *shift orbit closure* of a sequence $\mathbf{x}$ is the set of all sequences $\mathbf{t}$ with the property that each prefix of $\mathbf{t}$ appears as a factor of $\mathbf{x}$. Note that this set can be much larger than the set of all suffixes of $\mathbf{x}$.

The following theorem is well known [7, Prop. 3, p. 34]:

Theorem 29. *The lexicographically least sequence in the shift orbit closure of* $\mathbf{f}$ *is* $0\mathbf{f}$*, and the lexicographically greatest is* $1\mathbf{f}$*.*

Proof. We handle only the lexicographically least, leaving the lexicographically greatest to the reader.

The idea is to create a predicate $P(n)$ for the lexicographically least sequence $\mathbf{b} = b_0 b_1 b_2 \cdots$ which is true iff $b_n = 1$. The following predicate encodes, first, that $b_n = 1$, and second, that if one chooses any length-$(n+1)$ factor t of $\mathbf{f}$, then $b_0 \cdots b_n$ is equal or lexicographically smaller than t.

$$\exists j\ \mathbf{f}[j+n] = 1\ \wedge\ \forall k\ ((\forall s \le n\ \mathbf{f}[j+s] = \mathbf{f}[k+s])\ \vee \\ (\exists i \le n \text{ s.t. } \mathbf{f}[j+i] < \mathbf{f}[k+i]\ \wedge\ (\forall t < i\ \mathbf{f}[j+t] = \mathbf{f}[k+t])))$$

When we do this we get the following automaton, which is easily seen to generate the sequence $0\mathbf{f}$.

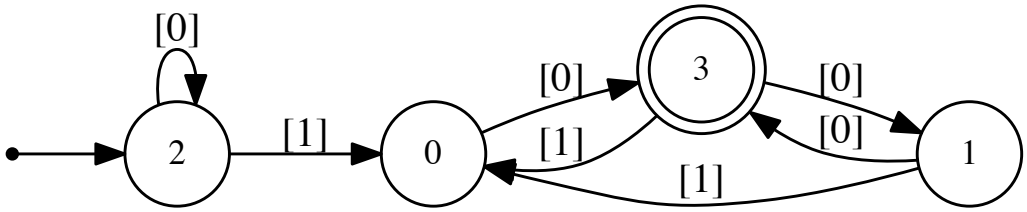

Figure 14: Automaton accepting lexicographically least sequence in shift orbit closure of $\mathbf{f}$

□

3.11 Minimal forbidden words

Let $\mathbf{x}$ be an infinite word. A finite word $z = a_0 \cdots a_n$ is said to be *minimal forbidden* if z is not a factor of $\mathbf{x}$, but both $a_1 \cdots a_n$ and $a_0 \cdots a_{n-1}$ are [18].

We can characterize all minimal forbidden words as follows: we create an automaton accepting the language

$$\{(i,n)_F \ : \ \mathbf{f}[i..i+n-1]\,\overline{\mathbf{f}[n]} \text{ is not a factor of } \mathbf{f} \text{ and}$$
$$\mathbf{f}[i+1..i+n-1]\,\overline{\mathbf{f}[n]} \text{ is a factor and } i \text{ is as small as possible } \}.$$

When we do so we find the words accepted are

$$[1,1]([0,0][1,1])^*(\epsilon + [0,0]).$$

This corresponds to the words

$$\mathbf{f}[F_n - 1..2F_n - 3]\,\overline{\mathbf{f}[2F_n - 2]}$$

for $n \geq 3$. The first few are

$$11, 000, 10101, 00100100, 1010010100101, \ldots.$$

3.12 Grouped factors

Cassaigne [12] introduced the notion of *grouped factors*. A sequence $\mathbf{a} = (a_i)_{i\geq 0}$ has grouped factors if, for all $n \geq 1$, there exists some position $m = m(n)$ such that $\mathbf{a}[m..m+\rho(n)+n-2]$ contains all the $\rho(n)$ length-n blocks of $\mathbf{a}$, each block occurring exactly once. One consequence of his result is that the Fibonacci word has grouped factors.

We can write a predicate for the property of having grouped factors, as follows:

$$\forall n \geq 1 \quad \exists m, s \geq 0 \quad \forall i \geq 0$$
$$\exists j \text{ s.t. } m \leq j \leq m+s \text{ and } \mathbf{a}[i..i+n-1] = \mathbf{a}[j..j+n-1] \text{ and}$$
$$\forall j', \ m \leq j' \leq m+s, \quad j \neq j' \text{ we have } \mathbf{a}[i..i+n-1] \neq \mathbf{a}[j'..j'+n-1].$$

The first part of the predicate says that every length-n block appears somewhere in the desired window, and the second says that it appears exactly once.

(This five-quantifier definition can be viewed as a response to the question of Homer and Selman [35], "...in what sense would a problem that required at least three alternating quantifiers to describe be natural?")

Using this predicate and our decision method, we verified that the Fibonacci word does indeed have grouped factors.

4 Mechanical proofs of properties of the finite Fibonacci words

Although our program is designed to answer questions about the properties of the infinite Fibonacci word $\mathbf{f}$, it can also be used to solve problems concerning the finite Fibonacci words

(X_n), 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 $n \geq 1$. (We caution the reader that there exist many variations on this definition in the literature, particularly with regard to indexing and initial values.) Furthermore, we have $\varphi(X_n) = X_{n+1}$ for $n \geq 1$.

Our strategy for the the finite Fibonacci words has two parts:

(i) Instead of phrasing statements in terms of factors, we phrase them in terms of occurrences of factors (and hence in terms of the indices defining a factor).

(ii) Instead of phrasing statements about finite Fibonacci words, we phrase them instead about *all* length-n prefixes of $\mathbf{f}$. Then, since $X_i = \mathbf{f}[0..F_i - 1]$, we can deduce results about the finite Fibonacci words by considering the case where n is a Fibonacci number F_i.

To illustrate this idea, consider one of the most famous properties of the Fibonacci words, the *almost-commutative* property: letting $\eta(a_1a_2 \cdots a_n) = a_1a_2 \cdots a_{n-2}a_na_{n-1}$ be the map that interchanges the last two letters of a string of length at least 2, we have

Theorem 30. *$X_{n-1}X_n = \eta(X_nX_{n-1})$ for $n \geq 2$.*

We can verify this, and prove even more, using our method.

Theorem 31. *Let $x = \mathbf{f}[0..i-1]$ and $y = \mathbf{f}[0..j-1]$ for $i > j > 1$. Then $xy = \eta(yx)$ if and only if $i = F_n$, $j = F_{n-1}$ for $n \geq 3$.*

Proof. The idea is to check, for each $i > j > 1$, whether

$$\mathbf{f}[0..i-1]\mathbf{f}[0..j-1] = \eta(\mathbf{f}[0..j-1]\mathbf{f}[0..i-1]).$$

We can do this with the following predicate:

$$(i > j) \ \wedge \ (j \geq 2) \ \wedge \ (\forall t, \ j \leq t < i, \ \mathbf{f}[t] = \mathbf{f}[t-j]) \wedge \\ (\forall s \leq j-3 \ \mathbf{f}[s] = \mathbf{f}[s+i-j]) \ \wedge \ (\mathbf{f}[j-2] = \mathbf{f}[i-1]) \ \wedge \ (\mathbf{f}[j-1] = \mathbf{f}[i-2]).$$

The log of our program is as follows:

```
i > j with 7 states, in 49ms
 j >= 2 with 5 states, in 87ms
  i > j & j >= 2 with 12 states, in 3ms
   j <= t with 7 states, in 3ms
    t < i with 7 states, in 17ms
     j <= t & t < i with 19 states, in 6ms
      F[t] = F[t - j] with 16 states, in 31ms
       j <= t & t < i => F[t] = F[t - j] with 62 states, in 31ms
        At j <= t & t < i => F[t] = F[t - j] with 14 states, in 43ms
         i > j & j >= 2 & At j <= t & t < i => F[t] = F[t - j] with 12 states, in 9ms
```

```
          s <= j - 3 with 14 states, in 72ms
           F[s] = F[s + i - j] with 60 states, in 448ms
            s <= j - 3 => F[s] = F[s + i - j] with 119 states, in 14ms
             As s <= j - 3 => F[s] = F[s + i - j] with 17 states, in 58ms
              i > j & j >= 2 & At j <= t & t < i => F[t] = F[t - j] & As s <= j - 3 => F[s] = F[s + i - j] with 6 states, in 4ms
               F[j - 2] = F[i - 1] with 20 states, in 34ms
                i > j & j >= 2 & At j <= t & t < i => F[t] = F[t - j] & As s <= j - 3 => F[s] = F[s + i - j] & F[j - 2] = F[i - 1] with 5 states, in 1ms
                 F[j - 1] = F[i - 2] with 20 states, in 29ms
                  i > j & j >= 2 & At j <= t & t < i => F[t] = F[t - j] & As s <= j - 3 => F[s] = F[s + i - j] & F[j - 2] = F[i - 1] & F[j - 1] = F[i - 2] with 5 states,
overall time: 940ms
```

The resulting automaton accepts $[1,0][0,1][0,0]^+$, which corresponds to $i = F_n$, $j = F_{n-1}$ for $n \geq 4$. □

An old result of Séébold [48] is

Theorem 32. *If uu is a square occurring in $\mathbf{f}$, then u is conjugate to some finite Fibonacci word.*

Proof. Assertion $\mathrm{conj}(i,j,k,\ell)$ means $\mathbf{f}[i..j]$ is a conjugate of $\mathbf{f}[k..\ell]$ (assuming $j-i = \ell - k$)

$$\mathrm{conj}(i,j,k,\ell) := \exists m\ \mathbf{f}[i..i+\ell-m] = \mathbf{f}[m..\ell] \text{ and } \mathbf{f}[i+\ell-m+1..j] = \mathbf{f}[k..m-1].$$

Predicate:

$$(\mathbf{f}[i..i+n-1] = \mathbf{f}[i+n..i+2n-1]) \implies \mathrm{conj}(i, i+n-1, 0, n-1)$$

This asserts that any square uu of order n appearing in $\mathbf{f}$ is conjugate to $\mathbf{f}[0..n-1]$. When we implement this, we discover that all lengths are accepted. This makes sense since the only lengths corresponding to squares are F_n, and for all other lengths the base of the implication is false. □

We now reprove an old result of de Luca [20]. Recall that a primitive word is a non-power; that is, a word that cannot be written in the form x^n where n is an integer ≥ 2.

Theorem 33. *All finite Fibonacci words are primitive.*

Proof. The factor $\mathbf{f}[i..j]$ is a power if and only if there exists d, $0 < d < j - i + 1$, such that $\mathbf{f}[i..j-d] = \mathbf{f}[i+d..j]$ and $\mathbf{f}[j-d+1..j] = \mathbf{f}[i..i+d-1]$. Letting $\mathrm{pow}(i,j)$ denote this predicate, the predicate

$$\neg\, \mathrm{pow}(0, n-1)$$

expresses the claim that the length-n prefix $\mathbf{f}[0..n-1]$ is primitive. When we implement this, we discover that the prefix of every length is primitive, except those prefixes of length $2F_n$ for $n \geq 4$. □

A theorem of Chuan [15, Thm. 3] states that the finite Fibonacci word X_n, for $n \geq 5$, is the product of two palindromes in exactly one way: where the first factor of length $F_{n-1} - 2$ and the second of length $F_{n-2} + 2$. (Actually, Chuan claimed this was true for all Fibonacci words, but, for example, for 010 there are evidently two different factorizations of the form $(\epsilon)(010)$ and $(010)\epsilon$.) We can prove something more general using our method, by generalizing:

Theorem 34. *If the length-n prefix* $\mathbf{f}[0..n-1]$ *of* $\mathbf{f}$ *is the product of two (possibly empty) palindromes, then* $(n)_F$ *is accepted by the automaton in Figure 15 below.*

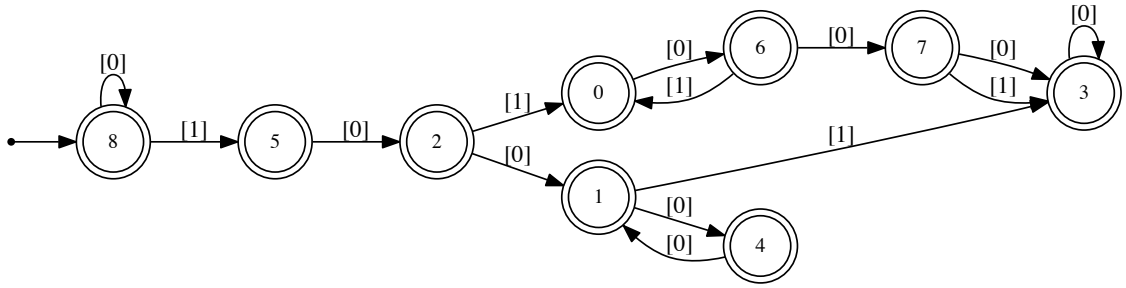

Figure 15: Automaton accepting lengths of prefixes that are the product of two palindromes

Furthermore, if the length-n prefix $\mathbf{f}[0..n-1]$ *of* $\mathbf{f}$ *is the product of two (possibly empty) palindromes in exactly one way, then* $(n)_F$ *is accepted by the automaton in Figure 16 below.*

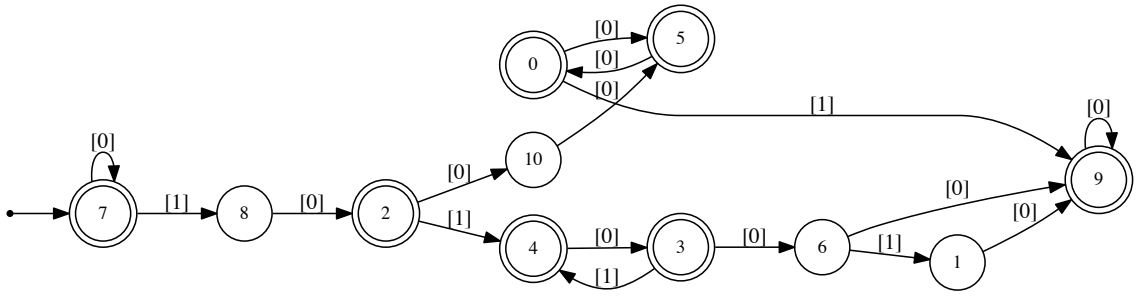

Figure 16: Automaton accepting lengths of prefixes that are the product of two palindromes in exactly one way

Evidently, this includes all n *of the form* F_j *for* $j \geq 5$.

Proof. For the first, we use the predicate

$$\exists p \leq n \ \left((\forall t < p \ \mathbf{f}[t] = \mathbf{f}[p-1-t]) \ \wedge \ (\forall u < n-p \ \mathbf{f}[p+u] = \mathbf{f}[n-1-u])\right).$$

For the second, we use the predicate

$$\exists p \le n\ ((\forall t < p\ \mathbf{f}[t] = \mathbf{f}[p-1-t])\ \wedge\ (\forall u < n-p\ \mathbf{f}[p+u] = \mathbf{f}[n-1-u]))) \wedge$$
$$(\forall q \le n\ ((\forall m < q\ \mathbf{f}[m] = \mathbf{f}[q-1-m])\ \wedge\ (\forall v < n-q\ \mathbf{f}[q+v] = \mathbf{f}[n-1-v]))\ \Longrightarrow\ p = q).$$

□

A result of Cummings, Moore, and Karhumäki [17] states that the borders of the finite Fibonacci word $\mathbf{f}[0..F_n - 1]$ are precisely the words $\mathbf{f}[0..F_{n-2k} - 1]$ for $2k < n$. We can prove this, and more:

Proof. Consider the pairs (n, m) such that $1 \le m < n$ and $\mathbf{f}[0..m-1]$ is a border of $\mathbf{f}[0..n-1]$. Their Fibonacci representations are accepted by the automaton below in Figure 17.

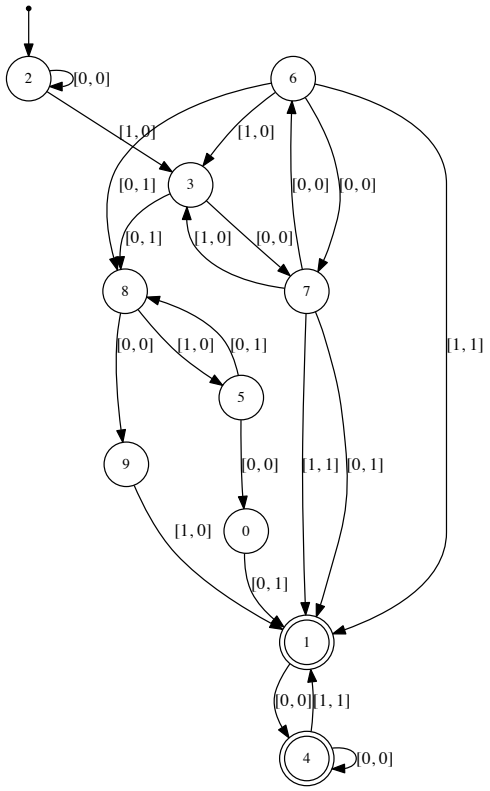

Figure 17: Automaton encoding borders of prefixes of $\mathbf{f}$

We use the predicate

$$(n > m)\ \wedge\ (m \ge 1)\ \wedge\ \forall i < m\ \mathbf{f}[i] = \mathbf{f}[n-m+i].$$

By following the paths with first coordinate of the form 10^+ we recover the result of Cummings, Moore, and Karhumäki as a special case. □

5 Details about our implementation

Our prover, called `Walnut`, was written in JAVA by Hamoon Mousavi, and was developed using the `Eclipse` development environment.[2] We used the `dk.brics.automaton` package, developed by Anders Møller at Aarhus University, for automaton minimization.[3] `Maple 15` was used to compute characteristic polynomials.[4] The `GraphViz` package was used to display automata.[5]

Our program consists of about 5000 lines of code. We used Hopcroft's algorithm for DFA minimization.

A user interface is provided to enter queries in a language very similar to the language of first-order logic. The intermediate and final result of a query are all automata. At every intermediate step, we chose to do minimization and determinization, if necessary. Each automaton accepts tuples of integers in the numeration system of choice. The built-in numeration systems are ordinary base-k representations and Fibonacci base. However, the program can be used with any numeration system for which an automaton for addition and ordering can be provided. These numeration system-specific automata can be declared in text files following a simple syntax. For the automaton resulting from a query it is always guaranteed that if a tuple t of integers is accepted, all tuples obtained from t by addition or truncation of leading zeros are also accepted. In Fibonacci representation, we make sure that the accepting integers do not contain consecutive 1's.

The program was tested against hundreds of different test cases varying in simplicity from the most basic test cases testing only one feature at a time, to more comprehensive ones with many alternating quantifiers. We also used known facts about automatic sequences and Fibonacci word in the literature to test our program, and in all those cases we were able to get the same result as in the literature. In a few cases, we were even able to find small errors in those earlier results.

The source code and manual for `Walnut` is available for free download at

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

6 Acknowledgments

We thank Narad Rampersad and Michel Rigo for useful suggestions.

References

[1] C. Ahlbach, J. Usatine, C. Frougny, and N. Pippenger. Efficient algorithms for Zeckendorf arithmetic. *Fibonacci Quart.* **51** (2013), 249–256.

[2] Available from `http://www.eclipse.org/ide/` .

[3] Available from `http://www.brics.dk/automaton/` .

[4] Available from `http://www.maplesoft.com` .

[5] Available from `http://www.graphviz.org` .

[2] J.-P. Allouche, N. Rampersad, and J. Shallit. Periodicity, repetitions, and orbits of an automatic sequence. *Theoret. Comput. Sci.* **410** (2009), 2795–2803.

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

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

[5] J. Berstel. Fonctions rationnelles et addition. In M. Blab, editor, *Théorie des Langages, École de printemps d'informatique théorique*, pp. 177–183. LITP, 1982.

[6] J. Berstel. Fibonacci words—a survey. In G. Rozenberg and A. Salomaa, editors, *The Book of L*, pp. 13–27. Springer-Verlag, 1986.

[7] J.-P. Borel and F. Laubie. Quelques mots sur la droite projective réelle. *J. Théorie Nombres Bordeaux* **5** (1993), 23–51.

[8] V. Bruyère and G. Hansel. Bertrand numeration systems and recognizability. *Theoret. Comput. Sci.* **181** (1997), 17–43.

[9] V. Bruyère, G. Hansel, C. Michaux, and R. Villemaire. Logic and p-recognizable sets of integers. *Bull. Belgian Math. Soc.* **1** (1994), 191–238. Corrigendum, *Bull. Belg. Math. Soc.* **1** (1994), 577.

[10] J. R. Büchi. Weak secord-order arithmetic and finite automata. *Zeitschrift für mathematische Logik und Grundlagen der Mathematik* **6** (1960), 66–92. Reprinted in S. Mac Lane and D. Siefkes, eds., *The Collected Works of J. Richard Büchi*, Springer-Verlag, 1990, pp. 398–424.

[11] L. Carlitz. Fibonacci representations. *Fibonacci Quart.* **6** (1968), 193–220.

[12] J. Cassaigne. Sequences with grouped factors. In *Developments in Language Theory III*, pp. 211–222. Aristotle University of Thessaloniki, 1998.

[13] E. Charlier, N. Rampersad, and J. Shallit. Enumeration and decidable properties of automatic sequences. *Internat. J. Found. Comp. Sci.* **23** (2012), 1035–1066.

[14] M. Christou, M. Crochemore, and C. S. Iliopoulos. Quasiperiodicities in Fibonacci strings. To appear in *Ars Combinatoria*. Preprint available at `http://arxiv.org/abs/1201.6162`, 2012.

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

[16] A. Cobham. Uniform tag sequences. *Math. Systems Theory* **6** (1972), 164–192.

[17] L. J. Cummings, D. Moore, and J. Karhumäki. Borders of Fibonacci strings. *J. Combin. Math. Combin. Comput.* **20** (1996), 81–87.

[18] J. D. Currie, N. Rampersad, and K. Saari. Suffix conjugates for a class of morphic subshifts. In J. Karhumäki, A. Lepistö, and L. Zamboni, editors, *WORDS 2013*, Vol. 8079 of *Lecture Notes in Computer Science*, pp. 95–106. Springer-Verlag, 2013.

[19] J. D. Currie and K. Saari. Least periods of factors of infinite words. *RAIRO Inform. Théor. App.* **43** (2009), 165–178.

[20] A. de Luca. A combinatorial property of the Fibonacci words. *Inform. Process. Lett.* **12** (1981), 193–195.

[21] X. Droubay. Palindromes in the Fibonacci word. *Inform. Process. Lett.* **55** (1995), 217–221.

[22] C. F. Du, H. Mousavi, L. Schaeffer, and J. Shallit. Decision algorithms for Fibonacci-automatic words, II: Related sequences and avoidability. Submitted, 2015.

[23] C. F. Du, H. Mousavi, L. Schaeffer, and J. Shallit. Decision algorithms for Fibonacci-automatic words, III: Enumeration and abelian properties. Submitted, 2015.

[24] D. D. A. Epple and J. Siefken. Collapse: a Fibonacci and Sturmian game. Available at `http://www.siefkenj.com/tmp/Fibonacci-4.pdf`, 2014.

[25] A. S. Fraenkel. Systems of numeration. *Amer. Math. Monthly* **92** (1985), 105–114.

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

[27] C. Frougny. Linear numeration systems of order two. *Inform. Comput.* **77** (1988), 233–259.

[28] C. Frougny. Fibonacci representations and finite automata. *IEEE Trans. Inform. Theory* **37** (1991), 393–399.

[29] C. Frougny. Representations of numbers and finite automata. *Math. Systems Theory* **25** (1992), 37–60.

[30] C. Frougny and B. Solomyak. On representation of integers in linear numeration systems. In M. Pollicott and K. Schmidt, editors, *Ergodic Theory of $\mathbb{Z}^d$ Actions (Warwick, 1993–1994)*, Vol. 228 of *London Mathematical Society Lecture Note Series*, pp. 345–368. Cambridge University Press, 1996.

[31] D. Goc, D. Henshall, and J. Shallit. Automatic theorem-proving in combinatorics on words. In N. Moreira and R. Reis, editors, *CIAA 2012*, Vol. 7381 of *Lecture Notes in Computer Science*, pp. 180–191. Springer-Verlag, 2012.

[32] 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.

[33] D. Goc, K. Saari, and J. Shallit. Primitive words and Lyndon words in automatic and linearly recurrent sequences. In A.-H. Dediu, C. Martin-Vide, and B. Truthe, editors, *LATA 2013*, Vol. 7810 of *Lecture Notes in Computer Science*, pp. 311–322. Springer-Verlag, 2013.

[34] 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.

[35] S. Homer and A. L. Selman. *Computability and Complexity Theory.* Springer-Verlag, 2nd edition, 2011.

[36] C. S. Iliopoulos, D. Moore, and W. F. Smyth. A characterization of the squares in a Fibonacci string. *Theoret. Comput. Sci.* **172** (1997), 281–291.

[37] J. Karhumäki. On cube-free ω-words generated by binary morphisms. *Disc. Appl. Math.* **5** (1983), 279–297.

[38] 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.

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

[40] F. Mignosi and G. Pirillo. Repetitions in the Fibonacci infinite word. *RAIRO Inform. Théor. App.* **26** (1992), 199–204.

[41] A. Ostrowski. Bemerkungen zur Theorie der Diophantischen Approximationen. *Abh. Math. Sem. Hamburg* **1** (1922), 77–98,250–251. Reprinted in *Collected Mathematical Papers*, Vol. 3, pp. 57–80.

[42] G. Pirillo. Fibonacci numbers and words. *Discrete Math.* **173** (1997), 197–207.

[43] M. Presburger. Über die Volständigkeit eines gewissen Systems der Arithmetik ganzer Zahlen, in welchem die Addition als einzige Operation hervortritt. In *Sparawozdanie z I Kongresu matematyków krajów slowianskich*, pp. 92–101, 395. Warsaw, 1929.

[44] M. Presburger. On the completeness of a certain system of arithmetic of whole numbers in which addition occurs as the only operation. *Hist. Phil. Logic* **12** (1991), 225–233.

[45] K. Saari. Periods of factors of the Fibonacci word. In *WORDS 07*, 2007.

[46] K. Saari. Lyndon words and Fibonacci numbers. *J. Combin. Theory. Ser. A* **121** (2014), 34–44.

[47] L. Schaeffer and J. Shallit. The critical exponent is computable for automatic sequences. *Internat. J. Found. Comp. Sci.* **23** (2012), 1611–1626.

[48] P. Séébold. *Propriétés combinatoires des mots infinis engendrés par certains morphismes (Thèse de 3e cycle)*. PhD thesis, Université P. et M. Curie, Institut de Programmation, Paris, 1985.

[49] J. O. Shallit. A generalization of automatic sequences. *Theoret. Comput. Sci.* **61** (1988), 1–16.

[50] J. Shallit. Decidability and enumeration for automatic sequences: a survey. In A. A. Bulatov and A. M. Shur, editors, *CSR 2013*, Vol. 7913 of *Lecture Notes in Computer Science*, pp. 49–63. Springer-Verlag, 2013.

[51] 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.