RAIRO-Theor. Inf. Appl. 50 (2016) 39–66
DOI: 10.1051/ita/2016010

Available online at:
www.rairo-ita.org

DECISION ALGORITHMS FOR FIBONACCI-AUTOMATIC WORDS, I: BASIC RESULTS

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

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

Mathematics Subject Classification. 11B85, 68R15, 11A67, 11B39, 03D05, 68Q45.

1. Decidability

As is well-known, the logical theory $\mathrm{Th}(\mathbb{N}, +)$, sometimes called Presburger arithmetic, is decidable [51,52]. Büchi [11] showed that if we add the function $V_k(n) = k^e$, for some fixed integer $k \geq 2$, where $e = \max\{i \ : \ k^i \,|\, n\}$, 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.*, [58]):

Theorem 1.1. *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 a deterministic finite automaton with output (DFAO) $M = (Q, \Sigma_k, \Delta, \delta, q_0, \kappa)$. Here $\Sigma_k := \{0, 1, \cdots, k-1\}$ 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 is the Thue-Morse sequence $\mathbf{t} = t_0 t_1 t_2 \ldots$, the fixed point (starting with 0) of the morphism $0 \to 01$, $1 \to 10$.

Keywords and phrases. Decision procedure, infinite word, Fibonacci-automatic – square, cube, palindrome, first-order logic, Fibonacci representation – quasiperiod, unbordered word, Lyndon word, uniform recurrence – linear recurrence, critical exponent.

[1] School of Computer Science, University of Waterloo, Waterloo, ON N2L 3G1, Canada.
hamoon.mousavi@gmail.com; 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

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, 34–37]. Furthermore, in many cases we can explicitly enumerate various aspects of such sequences, such as subword complexity [15].

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 [32, 33], and hence a theorem analogous to Theorem 1.1 holds for these systems. See, for example [10]. 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 [24, 25], 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 [49], Lekkerkerker [44], and Zeckendorf [59], 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 [12, 28].

Such a representation can be written as a binary string $a_1a_2 \cdots a_n$ representing the integer $\sum_{1\leq i\leq n} a_iF_{n+2-i}$. For example, the binary string 10010001 is the Fibonacci representation of 43.

For $w = a_1a_2 \cdots a_n \in \Sigma_2^*$, we define $[a_1a_2 \cdots a_n]_F := \sum_{1\leq i\leq n} a_iF_{n+2-i}$, even if $a_1a_2 \cdots a_n$ has leading zeroes or consecutive 1's. By $(n)_F$ we mean the *canonical* Fibonacci representation 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* [4] as the analogue of the more familiar notation of k-automatic sequence [3, 18]. 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 $n \geq 0$. Of course, there is a choice between reading the representation starting with either the most-significant or least-significant digit; in this paper we always use the former.

An example of a Fibonacci-automatic sequence is the infinite Fibonacci word,

$$\mathbf{f} = f_0f_1f_2 \cdots = 01001010 \cdots$$

which is generated by the following 2-state automaton:

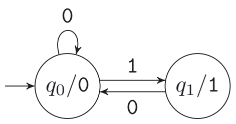

Figure 1. Canonical Fibonacci representation DFAO generating the Fibonacci word.

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 [57].

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 $(x_1, x_2, \cdots, x_r)$ consists of a string of symbols z over the alphabet Σ_2^r, such that the projection $\pi_i(z)$ over the ith 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 $(x_1, x_2, \cdots, x_r)$ is called canonical if it has no leading $[0, 0, \cdots 0]$ symbols and the projections into individual coordinates have no occurrences of 11. We write the canonical representation as $(x_1, x_2, \cdots, x_r)_F$. 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 [6]; also see [1, 7, 30, 31].

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 $Q = \{0, 1, 2, \cdots, 16\}$, 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. 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 (this latter one is used in the `Walnut` program). The state 0 is a "dead state" that can safely be ignored.

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 Table 2 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 $a_0 = 0$.

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

Let $A(x, y, z)$ denote the adder. First we check that

$$\forall x \; \forall z \; A(x, 0, z) \iff x = z.$$

Next, defining

$$\text{SUCC}(x, y) := (x < y) \; \wedge \; (\forall z \; (z \leq x) \; \vee \; (z \geq y)),$$

TABLE 1. Transition table for M_{add} for Fibonacci addition.

	[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 2. Identification of states with sequences.

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	$(-2F_n-3L_n)_{n\geq 0}$

we check that

$$\forall x\ \forall y\ \forall z\ \forall u\ \forall v\ (\textsc{succ}(y,u)\ \wedge\ \textsc{succ}(z,v)) \implies (A(x,y,z) \iff A(x,u,v)).$$

An induction on y now completes the proof.

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 $x<y$.

Quantifiers are handled as follows: for $\exists$, we use nondeterminism to "guess" and check a solution. For $\forall$, we rephrase the predicate in terms of $\exists$ and complementation.

Putting this all together, we get the analogue of Theorem 1.1.

Theorem 2.2. *There is an algorithm that, given a proposition phrased using only the universal and existential quantifiers, indexing into one or more Fibonacci-automatic sequences, addition, subtraction, logical operations, and comparisons, will decide the truth of that proposition.*

The algorithm uses a decision procedure as follows:

Procedure 2.3 (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 $\mathrm{Th}(\mathbb{N}, +)$ and indexing into $\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_m$.
Output: DFA with input alphabet Σ_2^n accepting

$$\{(k_1, k_2, \ldots, k_n)_F \ : \ \varphi(k_1, k_2, \ldots, k_n)\}.$$

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

- first, because the running time of the algorithm 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 the number of quantifier alternations in the logical formula characterizing the property being checked. This bound comes from the need to determinize a nondeterministic automaton before complementation can be done.
- second, because of the complexity of checking addition (17 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

In what follows, we examine a large number of results about the infinite Fibonacci word $\mathbf{f}$ and prove them using an implementation of the decision procedure sketched in the previous section. By $\mathbf{f}[i..j]$ we mean the factor of $\mathbf{f}$ beginning at position i and ending at position j.

The implementation, called `Walnut`, is discussed in more detail in Section 5. For the moment, we briefly mention the basic syntax: "`F`" refers to the sequence $\mathbf{f}$. "`E`" is our abbreviation for $\exists$ and "`A`" is our abbreviation for $\forall$. The symbol "`=>`" is logical implication, "`&`" is logical AND, and "`|`" is logical OR. 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. Predicates can be defined and used in later work. For example, the predicate

$$\textsc{FibEqFact}(i, j, n) := \forall t < n \ \mathbf{f}[i+t] = \mathbf{f}[j+t]$$

can be written in `Walnut` as

```
def fibeqfact "?msd_fib At (t<n) => F[i+t] = F[j+t]":
```

This defines a macro, `fibeqfact`, that takes three arguments: i, j, n (in that order) that is true iff $\mathbf{f}[i..i+n-1] = \mathbf{f}[j..j+n-1]$, that is, if the length-n factor beginning at position i in the Fibonacci word is the same as the length-n factor beginning at position j. The macro is represented by a finite automaton that takes the Fibonacci representation of (i, j, n) as input and accepts those inputs for which $\textsc{FibEqFact}(i, j, n)$ evaluates to true.

When the macro is used later in `Walnut`, it must be preceded by a dollar-sign. The order of arguments in the macro is alphabetical order of the unbound variables that appear in it when it is defined.

We provide the `Walnut` command for each predicate we discuss. By downloading the `Walnut` software and executing the appropriate commands, the reader can verify the results we present here.

3.1. Notation and definitions

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, [5, 7].

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

3.2. Repetitions

Theorem 3.1. *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 $\mathbf{f}$:

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

In `Walnut` this is

```
eval fibrep "?msd_fib ((p >= 1) & Ai (i >= n) => (F[i]=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 $\mathbb{N} = \{0, 1, 2, \cdots\}$). From this predicate, using our program, we construct 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 has 3 states: 3ms
i>=n has 6 states: 1ms
F[i]=F[(i+p)] has 11 states: 75ms
(i>=n=>F[i]=F[(i+p)]) has 41 states: 53ms
(A i (i>=n=>F[i]=F[(i+p)])) has 2 states: 96ms
(p>=1&(A i (i>=n=>F[i]=F[(i+p)]))) has 1 states: 1ms
total computation time: 234ms
```

□

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 ([42], Thm. 2):

Theorem 3.2. $\mathbf{f}$ *contains no fourth powers.*

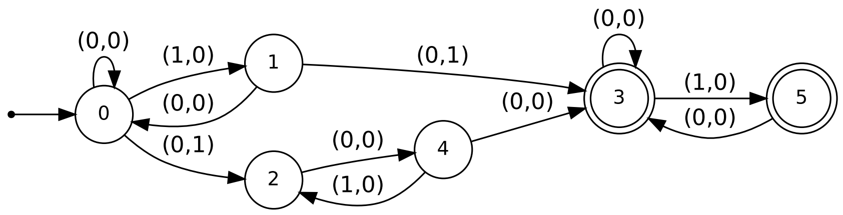

FIGURE 2. Automaton accepting orders and positions of all squares in $\mathbf{f}$.

Proof. We create a predicate for the orders of all fourth powers occurring in $\mathbf{f}$:

$$(n > 0) \ \wedge \ \exists i \ \text{FibEqFact}(i, i+n, 3n).$$

In `Walnut` this is

```
eval fib4 "?msd_fib (n>0) & Ei $fibeqfact(i,i+n,3*n)":
```

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

Next, we move on to a description of the orders of squares occurring in $\mathbf{f}$. An old result of Séébold [56] (also see [29, 41]) states

Theorem 3.3. *All squares in* $\mathbf{f}$ *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* $\mathbf{f}$*.*

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

$$(n > 0) \ \wedge \ \exists i \ \text{FibEqFact}(i, i+n, n).$$

In `Walnut` this is

```
eval fibsquarelen "?msd_fib (n>0) & Ei $fibeqfact(i,i+n,n)":
```

When we run this predicate, we obtain an automaton that accepts exactly the language 10^*. □

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 *et al.* ([41], Sect. 2), but their description is rather complicated and takes 5 pages to prove. Using our approach, we created an automaton accepting the language, which gives the starting position and lengths of all squares.

$$\{(i, n)_F \ : \ (n > 0) \ \wedge \ \text{FibEqFact}(i, i+n, n)\}.$$

In `Walnut` this is

```
eval fibsquareinfo "?msd_fib (n>0) & $fibeqfact(i,i+n,n)":
```

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

Theorem 3.4. *The language*

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

is accepted by the automaton in Figure 2.

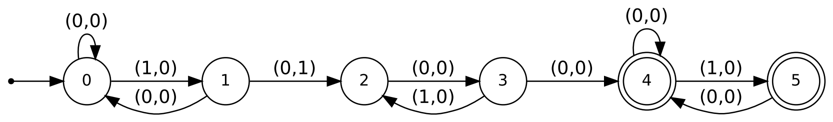

FIGURE 3. Automaton accepting orders and positions of all cubes in $\mathbf{f}$.

Next, we examine the cubes in $\mathbf{f}$. Evidently Theorem 3.3 implies that any cube in $\mathbf{f}$ must be of order F_n for some n. However, not every order occurs.

Theorem 3.5. *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

$$(n > 0) \ \wedge \ \exists i \ \ \text{FIBEQFACT}(i, i+n, 2n).$$

In `Walnut` this is

```
eval fibcube "?msd_fib (n>0) & Ei $fibeqfact(i,i+n,2*n)"
```

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

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

$$\{(i,n)_F \ : \ (n > 0) \ \wedge \ \forall t < 2n \ \ \text{FIBEQFACT}(i, i+n, 2n)\},$$

which is implemented in `Walnut` with

```
eval fibcubeinfo "?msd_fib (n>0) & $fibeqfact(i,i+n,2*n)":
```

and we get the following new result:

Theorem 3.6. *The language*

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

is accepted by the automaton in Figure 3.

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 3.7. *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

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

or, in `Walnut`,

```
eval fibantisquare "?msd_fib Ei (Ak (k<n) => (F[i+k] != F[i+k+n]))":
```

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

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

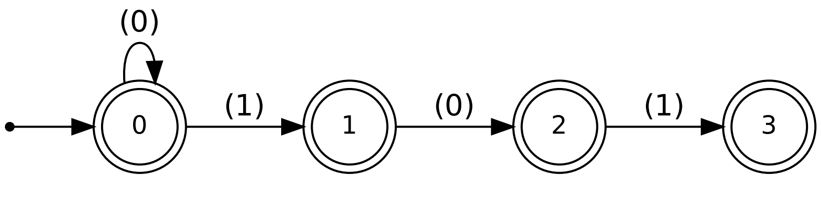

FIGURE 4. Automaton accepting orders of antisquares in $\mathbf{f}$.

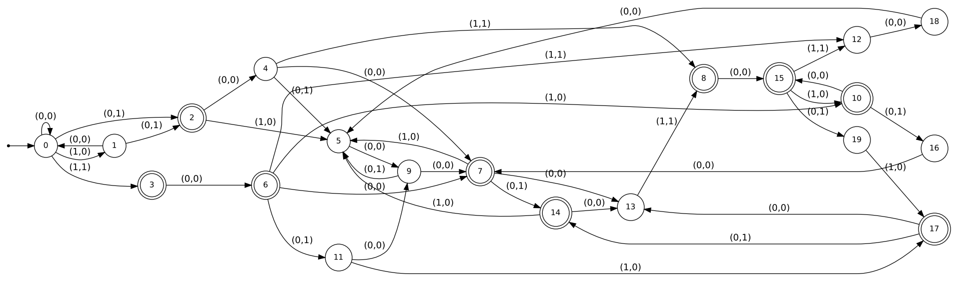

FIGURE 5. Automaton accepting orders and positions of all nonempty palindromes in $\mathbf{f}$.

3.3. PALINDROMES AND ANTIPALINDROMES

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

First, we define a predicate stating that $\mathbf{f}[i..i+n-1]$ is a nonempty palindrome:

$$\text{FIBPAL}(i,n) := (n>0) \wedge \; \forall t<n \; \mathbf{f}[i+t] = \mathbf{f}[i+n-1-t].$$

In `Walnut` this is

```
def fibpal "?msd_fib (n>0) & At (t<n) => F[i+t] = F[i+n-1-t]":
```

The resulting 20-state machine characterizes the positions and lengths of all nonempty palindromes in $\mathbf{f}$. It is not particularly enlightening, but is given to show the kind of complexity that can result from even simple queries (Fig. 5).

Using the predicate

$$\exists i \;\; \text{FIBPAL}(i,n)$$

we specify those lengths n for which there is a palindrome of length n. In `Walnut` this is

```
eval fibpallen "?msd_fib Ei $fibpal(i,n)":
```

Our program then recovers the following result of Chuan [17].

Theorem 3.8. *There exist nonempty palindromes of every length ≥ 1 in $\mathbf{f}$.*

Next, we prove a result of Droubay [23].

Theorem 3.9. *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 is odd.*

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

$$\exists i \;\; \text{FIBPAL}(i,n) \;\; \wedge \;\; \forall j \;\; \text{FIBPAL}(j,n) \implies \text{FIBEQFACT}(i,j,n)$$

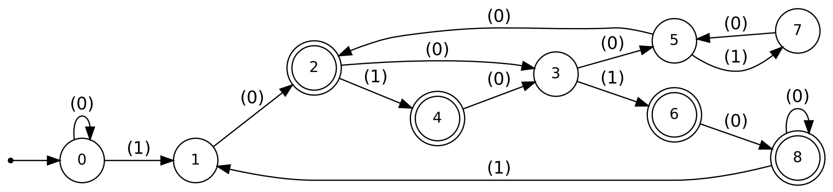

Figure 6. Automaton accepting lengths with exactly one palindrome.

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 $\mathbf{f}[i..i+n-1]$. In `Walnut` this is

```
eval fibonepalfac "?msd_fib Ei ($fibpal(i,n) & Aj $fibpal(j,n)
=> $fibeqfact(i,j,n))":
```

When we run this predicate through our program we get the automaton depicted below in Figure 6.

It may not be obvious, but this automaton accepts exactly the Fibonacci representations of the positive even numbers. The easiest way to check this is to use our program on the predicate $\exists i\ (i > 0)\ \wedge\ n = 2i$ and verify that the resulting automaton is isomorphic to that in Figure 6.

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.

$$\exists i\ \exists j\ \ \text{FibPal}(i,n)\ \wedge\ \text{FibPal}(j,n)\ \wedge\ (\neg\, \text{FibEqFact}(i,j,n))\ \wedge$$
$$(\forall u\ \ \text{FibPal}(u,n)\ \implies\ (\text{FibEqFact}(i,u,n)\ \vee\ \text{FibEqFact}(j,u,n)))$$

In `Walnut` this is

```
eval fibtwopal "?msd_fib Ei Ej $fibpal(i,n) & $fibpal(j,n) &
(~$fibeqfact(i,j,n)) & (Au $fibpal(u,n) =>
($fibeqfact(i,u,n)|$fibeqfact(j,u,n)))":
```

The interpretation of this predicate is that there are two indices i and j at which a palindrome of length n starts; these palindromes are distinct; and every palindrome of length n is equal to one or the other.

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 nonempty prefixes are palindromes:

Theorem 3.10. *The prefix* $\mathbf{f}[0..n-1]$ *of length* $n > 0$ *is a palindrome if and only if* $n = F_i - 2$ *for some* $i \geq 4$.

Proof. We use the predicate

$$\text{FibPal}(0, n)$$

or, in `Walnut`,

```
eval fibpalpre "?msd_fib $fibpal(0,n)":
```

obtaining an automaton accepting $1 + 10(10)^*(0 + 01)$, which are precisely the representations of $F_i - 2$ for $i \geq 4$. ☐

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

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

In Walnut this is

```
eval fibmirror "?msd_fib Ai Ej At (t < n) => F[i+t]=F[j+n-1-t]":
```

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 has 6 states: 0ms
F[(i+t)]=F[(((j+n)-1)-t)] has 263 states: 58582ms
(t<n=>F[(i+t)]=F[(((j+n)-1)-t)]) has 194 states: 27ms
(A t (t<n=>F[(i+t)]=F[(((j+n)-1)-t)])) has 34 states: 34ms
(E j (A t (t<n=>F[(i+t)]=F[(((j+n)-1)-t)]))) has 4 states: 1ms
(A i (E j (A t (t<n=>F[(i+t)]=F[(((j+n)-1)-t)])))) has 2 states: 0ms
total computation time: 58785ms
```

The time complexity arises from the large number of variables needed to make the assertion.

Thus we have reproved an old result (see, *e.g.*, Fischler [27]).

Theorem 3.11. *The word* **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 **f**:

Theorem 3.12. *The only nonempty antipalindromes in* **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:

$$\textsc{FibAp}(i,n) = (n > 0) \;\wedge\; \forall j < n \; \mathbf{f}[i+j] \neq \mathbf{f}[i+n-1-j],$$

or, in Walnut,

```
def fibap "?msd_fib (n>0) & (Aj (j<n) => (F[i+j] != F[i+n-1-j]))":
```

The resulting 6-state automaton encodes the starting positions and lengths of all nonempty antipalindromes.

Next, we write a predicate for $\mathbf{f}[i..i+n-1]$ to be an antipalindrome, with i the position of its first occurrence:

$$(n > 0) \;\wedge\; \textsc{FibAp}(i,n) \;\wedge\; \forall i' < i \; \neg \textsc{FibEqFact}(i,i',n),$$

or, in Walnut,

```
eval fibapfirst "?msd_fib (n>0) & $fibap(i,n) & (Aip (ip<i) =>
~$fibeqfact(i,ip,n))":
```

When we run this through our program, the language of $(i,n)_F$ satisfying this predicate is accepted by the following automaton (Fig. 7).

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

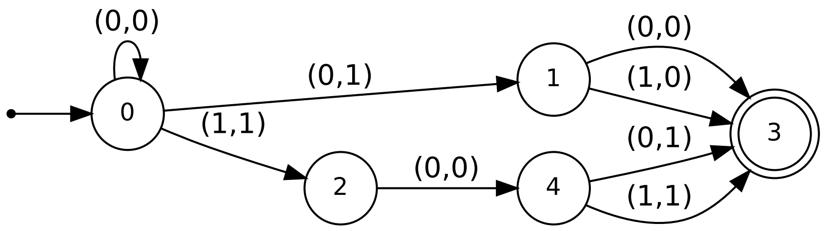

FIGURE 7. Automaton accepting orders and positions of first occurrences of nonempty antipalindromes in **f**.

3.4. SPECIAL FACTORS

Next we turn to special factors. It is well-known that **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 first write a predicate that expresses the assertion that the factor $\mathbf{f}[i..i+n-1]$ is a special factor:

$$\textsc{FibSpec}(i,n) := \exists j \;\; \textsc{FibEqFact}(i,j,n) \;\wedge\; \mathbf{f}[i+n] \neq \mathbf{f}[j+n],$$

or, in `Walnut`,

```
def fibspec "?msd_fib Ej ($fibeqfact(i,j,n) & (F[i+n]!=F[j+n]))":
```

Next, we check uniqueness by writing a predicate that asserts that there are two distinct special factors of length n:

$$\exists i \; \exists j \;\; \textsc{FibSpec}(i,n) \;\wedge\; \textsc{FibSpec}(j,n) \;\wedge\; \neg\,\textsc{FibEqFact}(i,j,n),$$

or, in `Walnut`,

```
eval fibspeccheck "?msd_fib Ei Ej $fibspec(i,n) & $fibspec(j,n)
& (~$fibeqfact(i,j,n))":
```

This automaton accepts nothing, so every special factor of **f** is unique.

Finally, we can create an automaton that says that $\mathbf{f}[i..i+n-1]$ is a special factor and i is the earliest occurrence of this factor:

$$\textsc{FibSpec}(i,n) \;\wedge\; \forall i' < i \;\neg\,\textsc{FibSpec}(i',n),$$

or, in `Walnut`,

```
eval fibspecfirst "?msd_fib $fibspec(i,n) & (Aip (ip < i) =>
~$fibspec(ip,n))":
```

Thus we have proved the following new result.

Theorem 3.13. *The automaton depicted below in Figure 8 accepts the language*

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

Furthermore it is known (*e.g.*, [50], Lem. 5) that

Theorem 3.14. *The unique special factor of length n is $\mathbf{f}[0..n-1]^R$.*

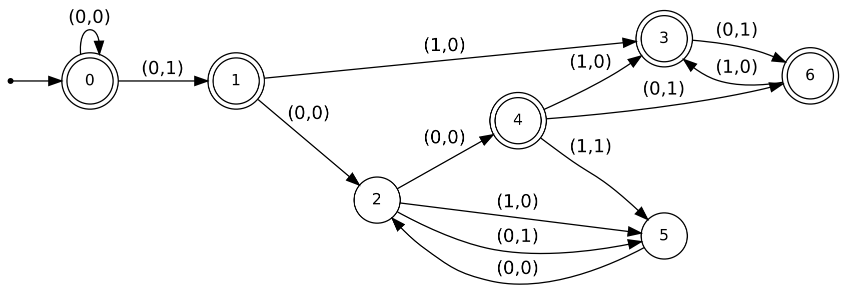

Figure 8. Automaton accepting first positions and lengths of special factors in $\mathbf{f}$.

Proof. We create a predicate that says that if a length-n factor is special then it matches $\mathbf{f}[0..n-1]^R$:

$$\forall i\, \text{FibSpec}(i,n) \implies (\forall j < n\ \mathbf{f}[i+j] = \mathbf{f}[n-1-j]),$$

or, in `Walnut`,

```
eval fibspecmatch "?msd_fib Ai $fibspec(i,n) => (Aj (j<n) => (F[i+j] = F[n-1-j]))":
```

When we run this we discover that all lengths are accepted. □

3.5. Least periods

We now turn to least periods of factors of $\mathbf{f}$ (see [53] and [26] and [20], Cor. 4).
Let FibPer denote the assertion that p is a period of the factor $\mathbf{f}[i..j]$, as follows:

$$\begin{aligned}\text{FibPer}(i,j,p) &:= (i \le j)\ \wedge\ (p \ge 1)\ \wedge\ (p \le j-i+1)\ \wedge\ \mathbf{f}[i..j-p] = \mathbf{f}[i+p..j] \\ &= (p \ge 1)\ \wedge\ \forall\, t,\ i \le t \le j-p\ \mathbf{f}[t] = \mathbf{f}[t+p].\end{aligned}$$

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

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

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

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

In `Walnut` this can be done as follows:

```
def fibper "?msd_fib (i <= j) & (p >= 1) & (p+i <=j+1) &
$fibeqfact(i,i+p,j+1-p-i)":
def fibleastper "?msd_fib $fibper(i,j,n) &
(Anp ((1<=np)&(np<n)) => (~$fibper(i,j,np)))":
def fiblp "?msd_fib Ei Ej $fibleastper(i,j,n)":
```

Using an implementation of this, we reprove the following theorem of Saari ([53], Thm. 2):

Theorem 3.15. *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 $n \ge 2$. 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 $n \ge 2$. □

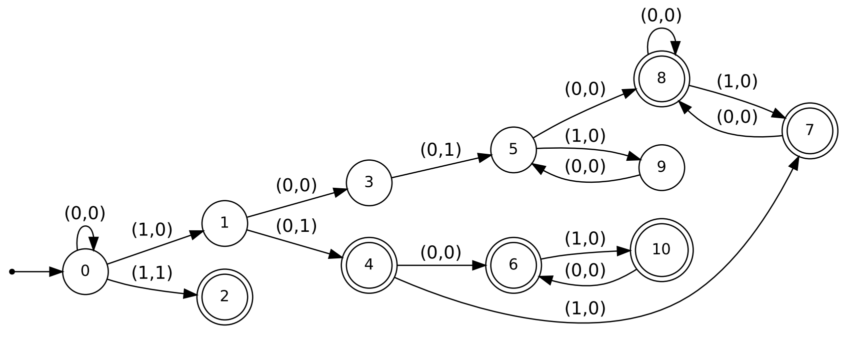

FIGURE 9. Automaton encoding smallest period over all length-n factors in $\mathbf{f}$.

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

Theorem 3.16. *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 $q \geq p$. The corresponding predicate is

$$(n \geq 1) \ \wedge \ \exists i \ \ \textsc{FibPer}(i, i+n-1, p) \ \wedge \ (\forall j \ \forall q \ \ \textsc{FibPer}(j, j+n-1, q) \implies q \geq p),$$

or, in `Walnut`,

```
eval fiblpl "?msd_fib (n>=1) & (Ei $fibper(i,i+n-1,p)) &
(Aj Aq $fibper(j,j+n-1,q) => (q >= p))":
```

This automaton is depicted in Figure 9 below.

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. □

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

Using the `Walnut` code

```
eval fibmaxrep "?msd_fib Et Eu Ev $fibleastper(i,i+n-1,t) &
$fibleastper(i,i+n,u) & $fibleastper(i-1,i+n-1,v) & (2*t <= n) &
(t < u) & ((i>0) => (t<v))":
```

we obtain the following new result:

Theorem 3.17. *The factor $\mathbf{f}[i..i+n-1]$ is a maximal repetition of $\mathbf{f}$ iff $(i,n)_F$ is accepted by the automaton depicted in Figure 10.*

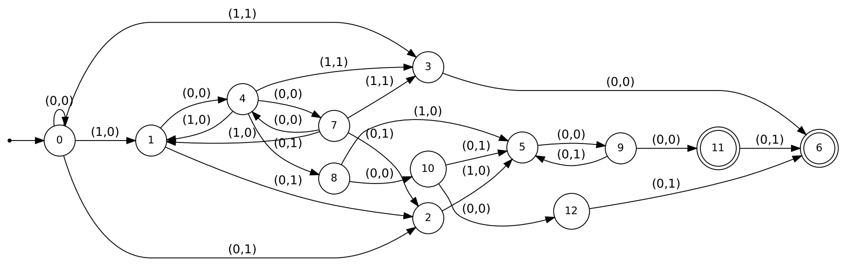

FIGURE 10. Automaton accepting occurrences of maximal repetitions in **f**.

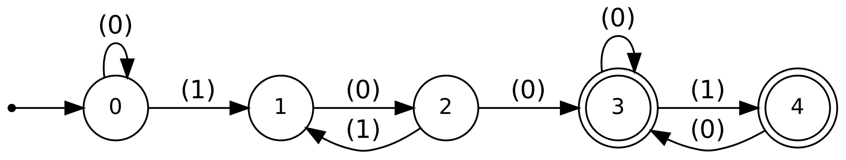

FIGURE 11. Automaton accepting lengths of prefixes of **f** that are quasiperiods.

3.6. Quasiperiods

We now turn to quasiperiods. An infinite word **a** is said to be *quasiperiodic* if there is some finite nonempty word x such that **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 **w** and is not allowed to "hang over"; these are called *aligned covers* in [16]. More precisely, for us $\mathbf{a} = a_0a_1a_2 \cdots$ 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 $a_ja_{j+1} \cdots a_{j+n-1} = x$, 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 **a**.

The quasiperiodicity of the Fibonacci word **f** was studied by Christou *et al.* [16], where we can (more or less) find the following theorem (also see [45]):

Theorem 3.18. *A nonempty length-n prefix of* **f** *is a quasiperiod of* **f** *if and only if n is not of the form $F_k - 1$ for $k \geq 3$.*

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:

$$\forall i \geq 0 \ \exists j \text{ with } i - n < j \leq i \ \ \textsc{FibEqFact}(0, j, n).$$

In `Walnut` this is

```
eval fibquasi "?msd_fib Ai (Ej ((i < j+n) & (j <= i) &
$fibeqfact(0,j,n)))":
```

When we do this, we get the automaton in Figure 11. Inspection shows that this DFA accepts all canonical representations, except those of the form

$$1(01)^*(\epsilon + 0),$$

which are precisely the representations of $F_k - 1$ for $k \geq 3$. □

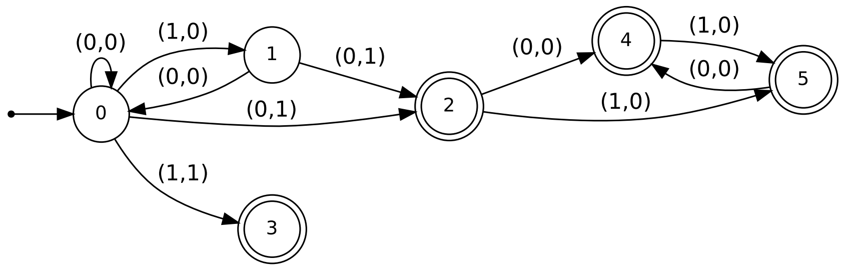

FIGURE 12. Automaton accepting positions and lengths of unbordered factors of $\mathbf{f}$.

3.7. 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 a word y is bordered iff it has a border of length ℓ with $0 < \ell \leq |y|/2$. We now prove a result due to (Harju and Kärki [38], Lem. 3).

Theorem 3.19. *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 assertion that $\mathbf{f}[i..i+n-1]$ is unbordered as follows:

$$\text{FibUnbf}(i,n) := (n>0) \ \wedge \ \forall j, 1 \leq j \leq n/2, \ \exists t < j \ \mathbf{f}[i+t] \neq \mathbf{f}[i+n-j+t].$$

In `Walnut` this is

```
def fibunbf "?msd_fib (n>0) & Aj (((1 <= j) & (2*j <= n)) =>
Et Er (t<j) & (F[r] != F[r+n-j]) & (r = i+t))":
```

Note: the slightly convoluted expression is needed so that the program runs to completion in a reasonable length of time. The result is depicted in Figure 12.

Once FibUnbf is defined, we can use it to obtain the lengths of all unbordered factors of $\mathbf{f}$ as follows: $\exists i \ \text{FibUnbf}(i,n)$, or in `Walnut`

```
eval fibub "?msd_fib Ei $fibunbf(i,n)":
```

The automaton produced accepts the Fibonacci representation of 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.

$$\exists i \ \exists k \ \text{FibUnbf}(i,n) \ \wedge \ \text{FibUnbf}(k,n) \ \wedge \ (\neg \text{FibEqFact}(i,k,n)) \ \wedge$$
$$(\forall j \ \text{FibUnbf}(j,n) \implies (\text{FibEqFact}(i,j,n) \ \vee \ \text{FibEqFact}(k,j,n)),$$

or in `Walnut`,

```
eval fib2unb "?msd_fib Ei Ek $fibunbf(i,n) & $fibunbf(k,n) &
(~$fibeqfact(i,k,n)) & (Aj $fibunbf(j,n) => ($fibeqfact(i,j,n) |
$fibeqfact(k,j,n)))":
```

When we do this we discover that the representations of all F_n for $n \geq 2$ are accepted.

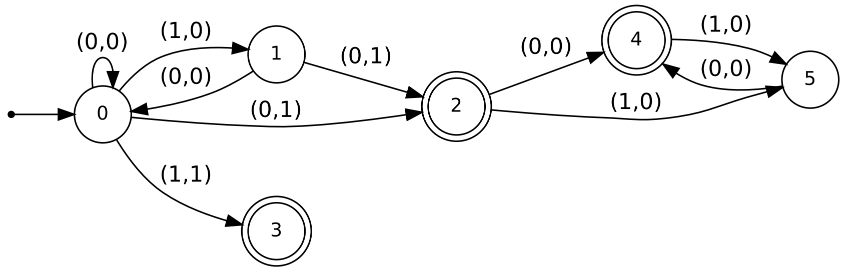

FIGURE 13. Automaton accepting positions and lengths of Lyndon factors of $\mathbf{f}$.

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.

We can achieve this with

$$\forall i\ \forall k\ ((\textsc{FibUnbf}(i,n)\ \wedge\ \textsc{FibUnbf}(k,n)) \implies \textsc{FibEqFact}(i,k,n)\ \vee\ (\forall \ell < n\ \mathbf{f}[i+\ell] = \mathbf{f}[k+n-\ell-1])),$$

or, in `Walnut`,

```
eval fib2unbrev "?msd_fib Ai Ak ($fibunbf(i,n) & $fibunbf(k,n)) =>
($fibeqfact(i,k,n) | (Al (l<n) => (F[i+l] = F[k+n-l-1])))":
```

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). ☐

3.8. 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[3]. We reprove some recent results of Currie and Saari [20] and Saari [54].

Theorem 3.20. *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 the factor $\mathbf{f}[i..i+n-1]$ is Lyndon:

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

or, in `Walnut`,

```
def fiblyndonfac "?msd_fib (n>=1) & Aj ((1 <= j) & (j < n)) =>
Et (t+j < n) & $fibeqfact(i,i+j,t) & (F[i+t]=@0) & (F[i+j+t]=@1)":
```

When we run this we get the automaton depicted in Figure 13.

Next, with the predicate $\exists i\ \textsc{FibLyndonFac}(i,n)$, or in `Walnut`,

```
eval fiblyndon "?msd_fib Ei $fiblyndonfac(i,n)":
```

we specify those lengths that have a Lyndon factor. When we run this we get the representations 10^*, which proves the result. ☐

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

Recall that two words x, w are said to be *conjugate* if one is a cyclic shift of the other.

Theorem 3.21. *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 $\mathbf{f}[i..i+n-1]$ is a conjugate of $\mathbf{f}[0..n-1]$:

$$\text{FibConjPre}(i,n) := \exists k < n\ (\text{FibEqFact}(i,k,n-k)\ \wedge\ \text{FibEqFact}(0,i+n-k,k)),$$

or, in Walnut,

```
def fibconjpre "?msd_fib Ek ((k<n) & $fibeqfact(i,k,n-k) &
$fibeqfact(0,i+n-k,k))":
```

Next, we create a predicate specifying that every length-n Lyndon factor is a conjugate of $\mathbf{f}[0..n-1]$:

$$\forall i\ \ \text{FibLyndonFac}(i,n) => \text{FibConjPre}(i,n),$$

or, in Walnut,

```
def fibchecklyndon "?msd_fib Ai $fiblyndonfac(i,n) => $fibconjpre(i,n)":
```

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.) □

3.9. 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 a recurrent factor. A factor x is *uniformly recurrent* if there exists a constant $c = c(x)$ such that the starting positions of two consecutive occurrences of x in $\mathbf{w}$ are always separated by at most c positions. If all factors are uniformly recurrent then the word $\mathbf{w}$ is said to be uniformly recurrent. Finally, $\mathbf{w}$ is *linearly recurrent* if it is uniformly recurrent, and the constant $c(x)$ is $O(|x|)$.

We now prove a well-known result (see, *e.g.*, [14]) using our method:

Theorem 3.22. *The word* $\mathbf{f}$ *is recurrent, uniformly recurrent, and linearly recurrent.*

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

$$\forall i \geq 0\ \exists j > i\ \ \text{FibEqFact}(i,j,n),$$

or, in Walnut,

```
eval fibrecur "?msd_fib Ai Ej (j>i) & $fibeqfact(i,j,n)":
```

This predicate says that for every factor $z = \mathbf{f}[i..i+n-1]$ and we can find another occurrence of z beginning at a position $j > i$. 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:

$$\forall i\ \exists \ell\ \forall j\ \exists s,\ j \leq s \leq j+l-n\ \ \text{FibEqFact}(i,s,n),$$

or, in Walnut,

```
eval fibunirecur "?msd_fib Ai El Aj Es (j<=s) & (s+n <= j+l)
& $fibeqfact(i,s,n)":
```

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

Finally, a predicate for linear recurrence with constant C:

$$\forall i\ \exists j\ (i < j \le i + Cn)\ \textsc{FibEqFact}(i, j, n),$$

or, in `Walnut`, with $C = 3$,

```
eval fiblinrec "?msd_fib Ai Ej (i < j) & (j<=i+3*n) &
$fibeqfact(i,j,n)":
```

When we run this predicate on `Walnut`, the representations of all $n \ge 1$ are accepted. However, if we change the condition $i < j \le i + Cn$ to $i < j < i + Cn$ then we discover that predicate evaluates to true only for $n \ge 2$. So $\mathbf{f}$ is linearly recurrent, with optimal recurrence constant 3. □

Remark 3.23. We can decide the property of linear recurrence for Fibonacci-automatic sequences 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 ([55], Thm. 8).

Even further, we can prove that if a Fibonacci-automatic (resp., k-automatic) sequence is uniformly recurrent, then it is linearly recurrent. To see this, note that if there is an upper bound u_n for the distance between consecutive factors of length n, then an automaton accepting $(n, u_n)_F$ (resp., $(n, u_n)_k$) exists, by the argument above, and a simple argument using the pumping lemma now shows that $u_n = O(n)$.

3.10. Critical exponents

Recall from Section 3.1 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 [46] is

Theorem 3.24. *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 [55], 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 3.16. 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 3.25. *The initial critical exponent of* $\mathbf{f}$ *is* $1 + \alpha$.

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

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

using the predicate $\textsc{FibLeastPer}(0, n-1, p)$, or, in `Walnut`,

```
eval fibice "?msd_fib $fibleastper(0,n-1,p)":
```

It is depicted in Figure 14. By inspection of the automaton, it is easy to see that the least period of the prefix of length $n \ge 1$ is F_j for $j \ge 2$ and $F_{j+1} - 1 \le n \le F_{j+2} - 2$. (Note that the Fibonacci representation of $\{F_i - 1\ :\ i \ge 3\}$ is given by the regular expression $1(01)^*(\epsilon + 0)$.) Hence the initial critical exponent is given by $\limsup_{j \to \infty} (F_{j+2} - 2)/F_j$, which is $1 + \alpha$. □

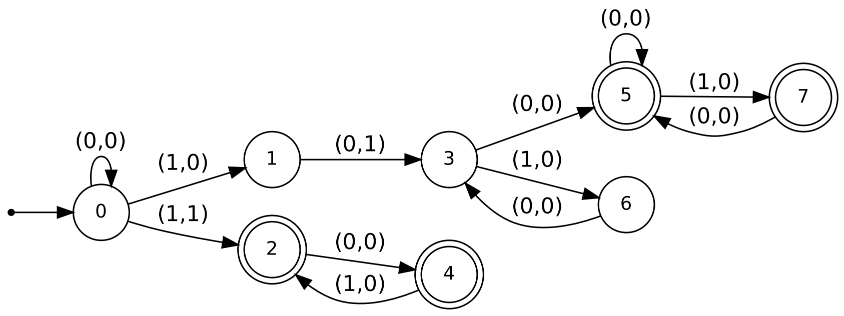

FIGURE 14. Automaton accepting least periods of prefixes of length n.

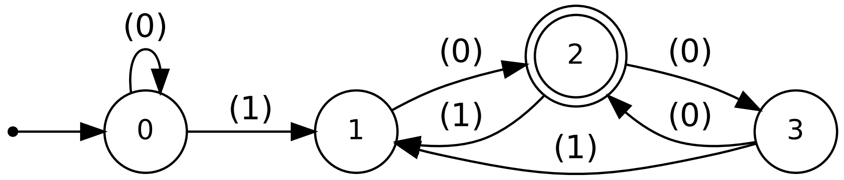

FIGURE 15. Automaton accepting lexicographically least sequence in shift orbit closure of $\mathbf{f}$.

3.11. 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 ([8], Prop. 3, p. 34):

Theorem 3.26. *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 $\mathbf{b} = b_0b_1b_2\cdots$ 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:

$$\exists j\ \mathbf{f}[j+n] = 1\ \wedge\ \forall k\ (\textsc{FibEqFact}(j,k,n+1)\ \vee\ (\exists i \le n\ \mathbf{f}[j+i] < \mathbf{f}[k+i]\ \wedge\ \textsc{FibEqFact}(j,k,i))),$$

or, in `Walnut`,

```
eval fibll "?msd_fib Ej (F[j+n] = @1) & (Ak $fibeqfact(j,k,n+1) |
(Ei (i<=n) & (F[j+i]=@0) & (F[k+i]=@1) & $fibeqfact(j,k,i)))":
```

When we do this, we get the automaton in Figure 15, which is easily seen to generate the sequence $0\mathbf{f}$, with 1 being the output associated with a final state and 0 for a non-final state (Fig. 15). □

3.12. 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 [21,47].

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

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

We achieve this with the two predicates

$$\text{FibMinf}(i,n) := (\forall k \ \ \text{FibEqFact}(i,k,n) \implies (\mathbf{f}[i+n] = \mathbf{f}[k+n])) \land \\ (\exists \ell \ \ \text{FibEqFact}(i+1,\ell+1,n-1) \ \land \mathbf{f}[i+n] \neq \mathbf{f}[\ell+n])$$

$$\text{FibMinf}(i,n) \ \land \ \forall t \ \text{FibMinf}(t,n) \implies (t \geq i),$$

or, in `Walnut`,

```
def fibminf "?msd_fib (Ak $fibeqfact(i,k,n) => (F[i+n] = F[k+n])) &
(El $fibeqfact(i+1,l+1,n-1) & (F[i+n] != F[l+n]))":
eval fibmfw "?msd_fib $fibminf(i,n) & (At $fibminf(t,n) => (t>=i))":
```

When we do so, we find the words accepted are

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

This regular expression specifies the words

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

for $n \geq 3$. The first few minimal forbidden words are therefore

$$11, 000, 10101, 00100100, 1010010100101, \cdots$$

3.13. Grouped factors

Cassaigne [13] 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 **a**, each block occurring exactly once. One consequence of his results is that the Fibonacci word has grouped factors.

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

$$\forall n \geq 1 \quad \exists m, s \geq 0 \quad \forall i \geq 0 \\ \exists j \ m \leq j \leq m+s \text{ and } \mathbf{a}[i..i+n-1] = \mathbf{a}[j..j+n-1] \text{ and} \\ \forall j', \ m \leq j' \leq m+s, \quad j \neq j' \text{ we have } \mathbf{a}[i..i+n-1] \neq \mathbf{a}[j'..j'+n-1].$$

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 [39], "··· in what sense would a problem that required at least three alternating quantifiers to describe be natural?")

In the case of **f**, the infinite Fibonacci word, we can simplify the verification somewhat by using the classical fact that **f** has subword complexity $n+1$; that is, that there are exactly $n+1$ distinct factors of length n [48].

We first create a predicate stating that there are grouped factors of length n beginning at position m:

$$\text{FibGf}(m,n) := \forall j \ \text{ with } (m \leq j \leq m+n) \ \forall k \text{ with } (m \leq k \leq m+n) \\ (j \neq k) \implies \neg\,\text{FibEqFact}(j,k,n),$$

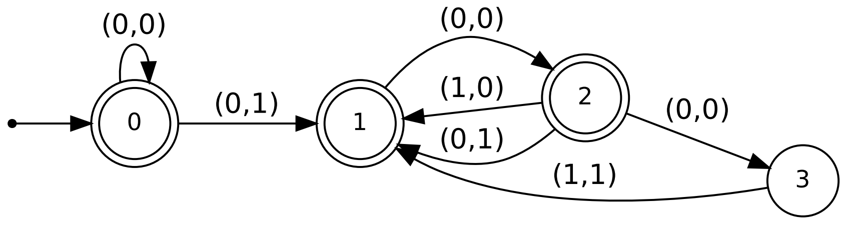

FIGURE 16. Automaton for first occurrence of grouped length-n factors of $\mathbf{f}$.

or, in `Walnut`,

```
def fibgf "?msd_fib Aj Ak (((j>=m)&(k>=m)&(j<=m+n)&(k<=m+n)&(j!=k)) =>
(~$fibeqfact(j,k,n)))":
```

Next, we verify that there are grouped factors for all n by evaluating

$$\exists m \, \textsc{FibGf}(m,n),$$

or, in `Walnut`,

```
eval fibgfcheck "?msd_fib Em $fibgf(m,n)":
```

which verifies the assertion for all n.

For a new result, we create a predicate that specifies for each n, the smallest m such that there are grouped factors beginning at position m of $\mathbf{f}$:

$$\textsc{FibGf}(m,n) \ \wedge \ \forall m' < m \, \neg \, \textsc{FibGf}(m',n)$$

or, in `Walnut`,

```
eval fibgfm "?msd_fib $fibgf(m,n) & (Amp (mp < m) => (~$fibgf(mp, n)))":
```

This gives the automaton depicted below in Figure 16.

Theorem 3.27. *The automaton in Figure 16 specifies, for each n, the smallest m such that the grouped length-n factors appear for the first time at position m.*

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 ([3], Thm. 7.1.2): 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 4.1. *$X_{n-1}X_n = \eta(X_nX_{n-1})$ for $n \geq 2$.*

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

Theorem 4.2. *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:

$$(i > j) \ \wedge \ (j > 1) \ \wedge \ \textsc{FibEqFact}(0, j, i-j) \ \wedge \\ \textsc{FibEqFact}(0, i-j, j-2) \ \wedge \ (\mathbf{f}[j-2] = \mathbf{f}[i-1]) \ \wedge \ (\mathbf{f}[j-1] = \mathbf{f}[i-2]),$$

or, in Walnut,

```
eval fibfinite "?msd_fib (i>j) & (j>1) & $fibeqfact(0,j,i-j) &
$fibeqfact(0,i-j,j-2) & (F[j-2]=F[i-1]) & (F[j-1]=F[i-2])":
```

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

An old result of Séébold [56] is

Theorem 4.3. *If uu is a square occurring in $\mathbf{f}$, then u is conjugate to some finite Fibonacci word.*

Proof. Recalling FibConjPre as defined in Section 3.8, we can express the assertion that any square uu of order n appearing in $\mathbf{f}$ is conjugate to $\mathbf{f}[0..n-1]$:

$$\forall i \ \textsc{FibEqFact}(i, i+n, n) \implies \textsc{FibConjPre}(i, n),$$

or, in Walnut,

```
def fibsqc "?msd_fib Ai $fibeqfact(i,i+n,n) => $fibconjpre(i,n)":
```

When we implement this, we discover that all lengths ≥ 1 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 [22]. 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.

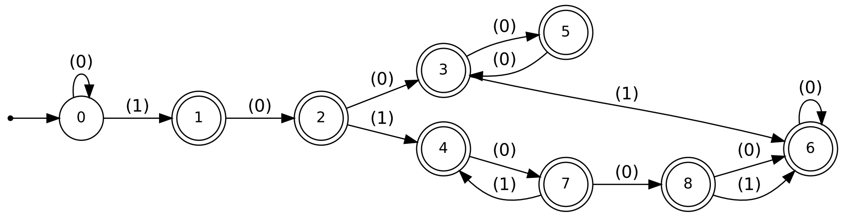

FIGURE 17. Automaton accepting lengths of prefixes that are the product of two palindromes.

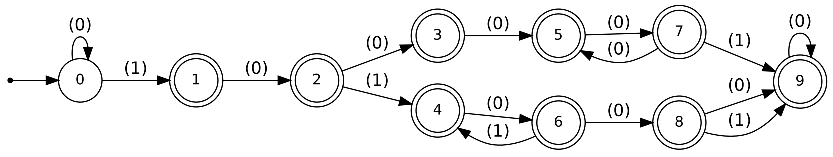

FIGURE 18. Automaton accepting lengths of prefixes that are the product of two palindromes in exactly one way.

Theorem 4.4. *All finite Fibonacci words are primitive.*

Proof. We prove somewhat more. 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 using the following two `Walnut` commands

```
def fibpow "?msd_fib Ed (0<d)&(d+i<=j)&$fibeqfact(i,i+d,j-d-i+1)&
$fibeqfact(j-d+1,i,d)":
eval fibprim "?msd_fib ~$fibpow(0,n-1)":
```

we discover something more: namely, that the prefix of every length is primitive, except those prefixes of length $2F_n$ for $n \geq 4$. Since for $k \geq 4$ we never have $2F_n = F_k$, Theorem 4.4 is a consequence. □

A theorem of Chuan ([17], 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 is of length $F_{n-1} - 2$ and the second is 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 4.5. *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 17 below. 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 18 below. Evidently, this includes all n of the form F_j for $j \geq 5$.*

Proof. For the first, we first define a predicate stating that $\mathbf{f}[i..i + n - 1]$ is a (possibly empty) palindrome:

$$\textsc{FibPale}(i, n) := \forall t < n\ \mathbf{f}[i + t] = \mathbf{f}[i + n - 1 - t].$$

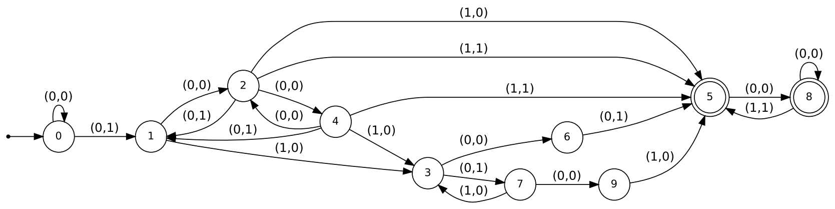

FIGURE 19. Automaton encoding borders of prefixes of $\mathbf{f}$.

Next, we define a predicate alleging that the prefix of length n is a product of two (possibly empty) palindromes, with the first being of length i:

$$\text{FIBPALPROD}(i, n) := (i < n) \ \wedge\ \text{FIBPALE}(0, i) \ \wedge\ \text{FIBPALE}(i, n - i).$$

Next, we use this predicate to compute the lengths of all prefixes that are the product of two (possibly empty) palindromes:

$$\exists i\, \text{FIBPALPROD}(i, n)$$

Finally, we define a predicate stating that the prefix of length n is a product of two palindromes, but only in one way:

$$\text{FIBUNIPALPROD}(n) := \exists p \ \wedge\ \text{FIBPALPROD}(p, n) \ \wedge\ (\forall q \ \ \text{FIBPALPROD}(q, n) => (p = q)).$$

In `Walnut` this is

```
def fibpale "?msd_fib At (t<n) => (F[i+t] = F[i+n-1-t])":
def fibpalprod "?msd_fib (i<n) & $fibpale(0,i) & $fibpale(i,n-i)":
def fibpalprodlen "?msd_fib Ei $fibpalprod(i,n)":
def fibunipalprod "?msd_fib Ep ($fibpalprod(p,n) & (Aq $fibpalprod(q,n) => (p = q)))":
```

As a result we get the automaton accepting the lengths of prefixes that are products of two palindromes (in Fig. 17) and the automaton accepting the lengths of prefixes that are products of two palindromes in only one way (in Fig. 18). □

A result of Cummings, Moore, and Karhumäki [19] 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 19.

To get this, we used the predicate

$$(n > m) \ \wedge\ (m \ge 1) \ \wedge\ \text{FIBEQFACT}(0, n - m, m),$$

or, in `Walnut`,

```
eval fibbord "?msd_fib (n>m) & (m >= 1) & $fibeqfact(0,n-m,m)":
```

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[4]. We used the package `dk.brics.automaton`, developed by Anders Møller at Aarhus University, for automaton minimization[5]. The `GraphViz` package was used to display automata[6].

Our program (not counting the `dk.brics.automaton`) package, consists of about 4800 lines of code. We used Hopcroft's algorithm for DFA minimization [40].

A user interface is provided to enter queries in a language very similar to the language of first-order logic. The intermediate and final results 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`.

All the calculations in this paper were performed on a MacBook Pro, with 2.6 GHz Intel Core i5 processor, running OS X Yosemite 10.10.5.

Acknowledgements. We thank Narad Rampersad and Michel Rigo for useful suggestions. We also thank the referees, who read this paper with care and made many helpful suggestions that improved the paper.

References

[1] C. Ahlbach, J. Usatine, C. Frougny and N. Pippenger, Efficient algorithms for Zeckendorf arithmetic. *Fibonacci Quart.* **51** (2013) 249–256.
[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.-P. Allouche, J. Shallit and G. Skordev, Self-generating sets, integers with missing blocks, and substitutions. *Discrete Math.* **292** (2005) 1–15.
[5] J. Berstel, Mots de Fibonacci. *Séminaire d'Informatique Théorique, LITP* **6-7** (1980–81) 57–78.
[6] J. Berstel, Fonctions rationnelles et addition. In *Théorie des Langages, École de printemps d'informatique théorique*, edited by M. Blab. LITP (1982) 177–183.
[7] J. Berstel, Fibonacci Words – A Survey. In *The Book of L*, edited by G. Rozenberg and A. Salomaa. Springer-Verlag (1986) 13–27.
[8] J.-P. Borel and F. Laubie, Quelques mots sur la droite projective réelle. *J. Théorie Nombres Bordeaux* **5** (1993) 23–51.
[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] V. Bruyère and G. Hansel, Bertrand numeration systems and recognizability. *Theoret. Comput. Sci.* **181** (1997) 17–43.

[4] Available from `http://www.eclipse.org/ide/`.
[5] Available from `http://www.brics.dk/automaton/`.
[6] Available from `http://www.graphviz.org`.

[11] J.R. Büchi, Weak secord-order arithmetic and finite automata. *Zeitschrift für mathematische Logik und Grundlagen der Mathematik* **6** (1960) 66–92. Reprinted in *The Collected Works of J. Richard Büchi*, edited by S. Mac Lane and D. Siefkes. Springer-Verlag (1990) 398–424.
[12] L. Carlitz, Fibonacci representations. *Fibonacci Quart.* **6** (1968) 193–220.
[13] J. Cassaigne, Sequences with grouped factors. In *Developments in Language Theory III.* Aristotle University of Thessaloniki (1998) 211–222.
[14] J. Cassaigne, Recurrence in infinite words. In *STACS 2001*, edited by A. Ferreira and H. Reichel. Vol. 2010 of *Lect. Notes Comput. Sci.* Springer-Verlag (2001) 1–11.
[15] E. Charlier, N. Rampersad and J. Shallit, Enumeration and decidable properties of automatic sequences. *Int. J. Found. Comp. Sci.* **23** (2012) 1035–1066.
[16] M. Christou, M. Crochemore and C.S. Iliopoulos, Quasiperiodicities in Fibonacci strings. Preprint arXiv:1201.6162 (2012). To appear in *Ars Combinatoria*.
[17] W.-F. Chuan, Symmetric Fibonacci words. *Fibonacci Quart.* **31** (1993) 251–255.
[18] A. Cobham, Uniform tag sequences. *Math. Systems Theory* **6** (1972) 164–192.
[19] L.J. Cummings, D. Moore and J. Karhumäki, Borders of Fibonacci strings. *J. Combin. Math. Combin. Comput.* **20** (1996) 81–87.
[20] J.D. Currie and K. Saari, Least periods of factors of infinite words. *RAIRO: ITA* **43** (2009) 165–178.
[21] J.D. Currie, N. Rampersad and K. Saari, Suffix conjugates for a class of morphic subshifts. In *WORDS 2013*, edited by J. Karhumäki, A. Lepistö and L. Zamboni. Vol. 8079 of *Lect. Notes Comput. Sci.* Springer-Verlag (2013) 95–106.
[22] A. de Luca, A combinatorial property of the Fibonacci words. *Inform. Process. Lett.* **12** (1981) 193–195.
[23] X. Droubay, Palindromes in the Fibonacci word. *Inform. Process. Lett.* **55** (1995) 217–221.
[24] C.F. Du, H. Mousavi, L. Schaeffer and J. Shallit, Decision algorithms for Fibonacci-automatic words, II: Related sequences and avoidability. In preparation (2016).
[25] C.F. Du, H. Mousavi, L. Schaeffer and J. Shallit, Decision algorithms for Fibonacci-automatic words, III: Enumeration and abelian properties. To appear in *Int. J. Found. Comput. Sci* (2016).
[26] D.D.A. Epple and J. Siefken, Collapse: A Fibonacci and Sturmian game. *Amer. Math. Monthly* **122** (2015) 515–527.
[27] S. Fischler, Palindromic prefixes and episturmian words. *J. Combin. Theory. Ser. A* **113** (2006) 1281–1304.
[28] A.S. Fraenkel, Systems of numeration. *Amer. Math. Monthly* **92** (1985) 105–114.
[29] A.S. Fraenkel and J. Simpson, The exact number of squares in Fibonacci words. *Theoret. Comput. Sci.* **218** (1999) 95–106.
[30] C. Frougny, Linear numeration systems of order two. *Inform. Comput.* **77** (1988) 233–259.
[31] C. Frougny, Fibonacci representations and finite automata. *IEEE Trans. Inform. Theory* **37** (1991) 393–399.
[32] C. Frougny, Representations of numbers and finite automata. *Math. Systems Theory* **25** (1992) 37–60.
[33] C. Frougny and B. Solomyak, On representation of integers in linear numeration systems. In *Ergodic Theory of $\mathbb{Z}^d$ Actions (Warwick, 1993–1994)*, edited by M. Pollicott and K. Schmidt. Vol. 228 of *London Math. Soc. Lect. Note Ser.* Cambridge University Press (1996) 345–368.
[34] D. Goc, D. Henshall and J. Shallit, Automatic theorem-proving in combinatorics on words. In *CIAA 2012*, edited by N. Moreira and R. Reis. Vol. 7381 of *Lect. Notes Comput. Sci.* Springer-Verlag (2012) 180–191.
[35] D. Goc, H. Mousavi and J. Shallit, On the number of unbordered factors. In *LATA 2013*, edited by A.-H. Dediu, C. Martin-Vide, and B. Truthe. Vol. 7810 of *Lect. Notes Comput. Sci.* Springer-Verlag (2013) 299–310.
[36] D. Goc, K. Saari and J. Shallit, Primitive words and Lyndon words in automatic and linearly recurrent sequences. In *LATA 2013*, edited by A.-H. Dediu, C. Martin-Vide and B. Truthe. Vol. 7810 of *Lect. Notes Comput. Sci.* Springer-Verlag (2013) 311–322.
[37] D. Goc, L. Schaeffer and J. Shallit, The subword complexity of k-automatic sequences is k-synchronized. In *DLT 2013*, edited by M.-P. Béal and O. Carton. Vol. 7907 of *Lect. Notes Comput. Sci.* Springer-Verlag (2013) 252–263.
[38] T. Harju and T. Kärki, On the number of frames of binary words. *Theoret. Comput. Sci.* **412** (2011) 5276–5284.
[39] S. Homer and A.L. Selman, Computability and Complexity Theory, 2nd edition. Springer-Verlag (2011).
[40] J.E. Hopcroft, An $n \log n$ algorithm for minimizing the states in a finite automaton. In *The Theory of Machines and Computation*, edited by Z. Kohavi. Academic Press, New York (1971) 189–196.
[41] 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.
[42] J. Karhumäki, On cube-free ω-words generated by binary morphisms. *Disc. Appl. Math.* **5** (1983) 279–297.
[43] R. Kolpakov and G. Kucherov, On maximal repetitions in words. In *Fundamentals of Computation Theory: FCT '99*, edited by G. Ciobanu and G. Păun. Vol. 1684 of *Lect. Notes Comput. Sci.* Springer-Verlag (1999) 374–385.
[44] C.G. Lekkerkerker, Voorstelling van natuurlijke getallen door een som van getallen van Fibonacci. *Simon Stevin* **29** (1952) 190–195.
[45] F. Levé and G. Richomme, Quasiperiodic infinite words: some answers. *Bull. Eur. Assoc. Theor. Comput. Sci.* **84** (2004) 128–138.
[46] F. Mignosi and G. Pirillo, Repetitions in the Fibonacci infinite word. *RAIRO: ITA* **26** (1992) 199–204.
[47] F. Mignosi, A. Restivo and M. Sciortino, Words and forbidden factors. *Theoret. Comput. Sci.* **273** (2002) 99–117.

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

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

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

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

[52] 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.

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

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

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

[56] P. Séébold, *Propriétés combinatoires des mots infinis engendrés par certains morphismes*. Ph. D. thesis, Université P. et M. Curie, Institut de Programmation, Paris (1985).

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

[58] J. Shallit, Decidability and enumeration for automatic sequences: a survey. In *CSR 2013*, edited by A.A. Bulatov and A.M. Shur. Vol. 7913 of *Lect. Notes Comput. Sci.* Springer-Verlag (2013) 49–63.

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

Communicated by D. Jamet.
Received March 24, 2016. Accepted March 29, 2016.