The Logical Approach to Automatic Sequences

Part 5: Fibonacci- and Tribonacci-Automatic Sequences... and Beyond

Jeffrey Shallit
School of Computer Science, University of Waterloo
Waterloo, Ontario N2L 3G1, Canada
shallit@cs.uwaterloo.ca
https://cs.uwaterloo.ca/~shallit

Beyond base-k expansions

Can our methods (the decision procedure to prove theorems; linear representations for enumeration) be extended further?

What ingredients do we need?

- A method to represent elements of $\mathbb{N}$ as strings
- An automaton to test equality of two such representations (easiest thing: have a notion of *canonical* expansion)
- An "adder": an automaton to test the proposition $x + y = z$

Fibonacci (Zeckendorf) representation

- Fibonacci numbers: $F_0 = 0,\ F_1 = 1,\ F_n = F_{n-1} + F_{n-2}$

- In analogy with base-2 representation, we can represent every non-negative integer in the form

$$\sum_{0 \leq i \leq t} \epsilon_i F_{i+2} \quad \text{with} \quad \epsilon_i \in \{0, 1\}.$$

Fibonacci (Zeckendorf) representation

- But then some integers have multiple representations, e.g., $14 = 13 + 1 = 8 + 5 + 1 = 8 + 3 + 2 + 1$
- So we impose the additional condition that $\epsilon_i \epsilon_{i+1} = 0$ for all i: never use two adjacent Fibonacci numbers.
- Usually we write the representation in the form

$$\epsilon_t \epsilon_{t-1} \cdots \epsilon_0,$$

with most significant digit first. So, for example, 19 is represented by 101001. This is called *Fibonacci representation* or *Zeckendorf representation*.

Edouard Zeckendorf (1901–1983)

Fibonacci-automatic infinite words

- Consider a finite automaton that takes Fibonacci representation of n as input
- Outputs are associated with the last state reached
- Invalid inputs (those with two consecutive 1's) are rejected or not considered
- An infinite word results from feeding the canonical representation of each $n \geq 0$ into the automaton
- Example: the Fibonacci infinite word

$$\mathbf{f} = 0100101001001 \cdots$$

The Fibonacci decision procedure

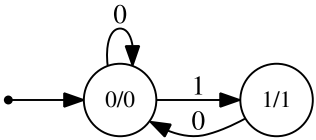
- Exactly like before, except now all integers are represented in Fibonacci representation
- Comparison is easy
- Addition is harder; need an adder
- There is a 17-state automaton that on input (x, y, z) in Fibonacci representation will determine whether $x + y = z$
- Based on ideas originally due to Jean Berstel and since elaborated by others: Frougny, Sakarovitch, etc.

The infinite Fibonacci word

The most famous Fibonacci-automatic word is the Fibonacci word

$$\mathbf{f} = 0100101001001010010100100101001001 \cdots ,$$

which can be defined in various ways.
One way is the fixed point of the morphism $\varphi(0) = 01$, $\varphi(1) = 0$.
Another way is the automaton



The infinite Fibonacci word

Yet another way is through the recursion

$$\begin{aligned} X_1 &= 1 \\ X_2 &= 0 \\ X_n &= X_{n-1}X_{n-2}, \quad (n \geq 2) \end{aligned}$$

So $X_3 = 01$, $X_4 = 010$, $X_5 = 01001$, etc.

Note that $|X_n| = F_n$.

The $(X_n)_{n\geq 1}$ are called the *finite Fibonacci words*, and for $n \geq 2$ they are all prefixes of $\mathbf{f}$.

The Fibonacci word

Properties of the infinite Fibonacci word **f** have been widely studied, e.g.:

- **f** is not ultimately periodic
- **f** contains no 4th powers (Karhumäki, 1983)
- All squares in **f** are of order F_n for $n \geq 2$, and squares of all these lengths exist (Séébold, 1985)
- There exist palindromes of all lengths in **f** (Chuan, 1993)

All of these claims can easily be verified using our method.

An extended example: avoiding the pattern xxx^R

- Recall that by x^R we mean the reversal of the string x. For example, $(\texttt{stressed})^R = \texttt{desserts}$ in English; $(\texttt{relativ})^R = \texttt{vitaler}$ in German.
- We are interested in avoiding the pattern xxx^R in binary words.
- An example of the pattern xxx^R in English is contained in the word

  `bepepper`.
- Examples in German: `Wiedererreichen` (reattainment) and `besessen` (obsessed)
- Are there infinite binary words avoiding this pattern?

An extended example: avoiding the pattern xxx^R

- We start by trying depth-first search of the space of binary words
- If there is a word avoiding the pattern, this procedure will give the lexicographically least such sequence.
- When we do, we get the word

$$(001)^3(10)^\omega = 001001001101010\cdots .$$

- So in particular the word $(10)^\omega = 101010\cdots$ avoids the pattern. (Easy proof!)
- This suggests: are there any *other* periodic infinite words avoiding xxx^R?
- Also: are there any *aperiodic* infinite words avoiding xxx^R?

An extended example: avoiding the pattern xxx^R

When we search for other primitive words z such that z^ω avoids the pattern, we find there are some of length 10:

0010011011 0011011001 0100110110 0110010011 0110110010
1001001101 1001101100 1011001001 1100100110 1101100100

- We notice that each of these words is of the form $w\overline{w}$.
- This suggests looking at words of this form.
- The next ones are $w = 0010010011011001001100$, and its shifts and complements.

An extended example: avoiding the pattern xxx^R

- To summarize, here are the solutions we've found so far:

w	$\lvert w \rvert$
01	2
00100	5
001001001101100100100	21

- The presence of the numbers 2,5,21 suggests some connection with the Fibonacci numbers: these are F_2, F_5, F_8.

An aperiodic word avoiding xxx^R

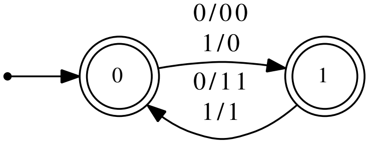
- Suppose we take the run-length encodings of the strings of length 21. One of them looks familiar: 2122121221221. This is a prefix of the infinite Fibonacci word generated by $2 \to 21$, $1 \to 2$.
- This suggests the construction of an *infinite* aperiodic word avoiding xxx^R: take the infinite Fibonacci word, and use it as "repetition factors" for 0 and 1 alternating. This gives the infinite word

  $$\mathbf{R} = 001001101101100100110 \cdots$$

  which we conjecture avoids xxx^R.
- Can we find an automaton generating this sequence? Yes, but now it is not based on base-2 representations, but rather Fibonacci (or "Zeckendorf") representations.

Another way to describe the word **R** is as follows:

Take the infinite Fibonacci word **f** and run it through the following transducer:



obtaining the infinite word

$$\mathbf{R} = 001001101101100100110110110010010011011001001001101100\cdots$$

Claim: it avoids the patterns xxx^R and also xx^Rx^R.

An aperiodic word avoiding xxx^R

- We can try to find an automaton for **R** using a "guess and test" procedure.
- When we do, we get the following automaton of 8 states.

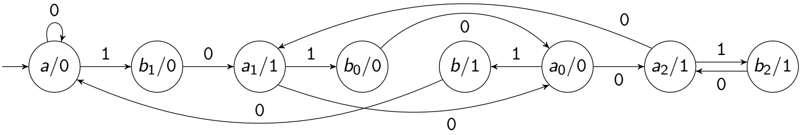


Figure : Fibonacci automaton generating the sequence **R**

An aperiodic word avoiding xxx^R

- We now have the conjecture that the word generated by this automaton (a) is aperiodic and (b) avoids xxx^R and (c) avoids xx^Rx^R.
- All three conjectures can be proved using our decision procedure.
- We just need to write predicates for them:
  - Ultimate periodicity:

$$\exists p \geq 1 \ \exists N \geq 0 \ \forall i \geq N \ \mathbf{R}[i] = \mathbf{R}[i+p].$$

  - Has xxx^R:

$$\exists i \geq 0 \ \exists n \geq 1 \ \forall t < n$$

$$(\mathbf{R}[i+t] = \mathbf{R}[i+t+n]) \ \wedge \ (\mathbf{R}[i+t] = \mathbf{R}[i+3n-1-t]).$$

The new result

Has xx^Rx^R:

$$\exists i \geq 0 \; \exists n \geq 1 \; \forall t < n$$

$$(\mathbf{R}[i+t] = \mathbf{R}[i+2n-1-t]) \;\wedge\; (\mathbf{R}[i+n+t] = \mathbf{R}[i+2n+t]).$$

Using `Walnut`, we can prove

Theorem. The Fibonacci-automatic word $\mathbf{R}$ generated by the automaton above is

(a) aperiodic and

(b) has no instances of the pattern xxx^R for x nonempty and

(c) also has no instances of the pattern xx^Rx^R for x nonempty.

Theorems about the finite Fibonacci words

- Since every finite Fibonacci word is a prefix of length F_n of the infinite Fibonacci word, we can rephrase many claims about the finite Fibonacci words in terms of our logical language
- There are two possible approaches: we can state these claims for length-n prefixes and ask for which n they are satisfied
- Or we can additionally restrict n in our logical language to have Fibonacci representation of the form 10^*

Example claim about the finite Fibonacci words

To illustrate this idea, consider one of the most famous properties of the Fibonacci words, the *almost-commutative* property: letting $\eta(a_1 a_2 \cdots a_n) = a_1 a_2 \cdots a_{n-2} a_n a_{n-1}$ be the map that interchanges the last two letters of a string of length at least 2, we have

Theorem

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

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

Theorem

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 of the almost-commutative property

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 formula:

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

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

Fibonacci-regular words and enumeration

- In many cases we can count the number $T(n)$ of length-n factors of a Fibonacci-automatic sequence having a particular property P.
- Here by "count" we mean, give an algorithm A to compute $T(n)$ efficiently, that is, in time bounded by a polynomial in $\log n$.
- Although *finding* the algorithm A may not be particularly efficient, once we have it, we can compute $T(n)$ quickly.

Reproving (and fixing) a result of Fraenkel and Simpson

We turn to a result of Fraenkel and Simpson (1999). They computed the exact number of occurrences of all squares appearing in the finite Fibonacci words X_n.
To solve this using our approach, we generalize the problem to consider *any* length-n prefix of $\mathbf{f}$.
The total number of square occurrences in $\mathbf{f}[0..n-1]$:

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

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

Next, we compute matrices M_0 and M_1, indexed by states of M, such that $(M_a)_{k,l}$ counts the number of edges (corresponding to the variables i and j) from state k to state l on the digit a of n. We also compute a vector u corresponding to the initial state of M and a vector v corresponding to the final states of M. This gives us the following linear representation of the sequence $b(n)$: if $x = a_1 a_2 \cdots a_t$ is the Fibonacci representation of n, then

$$b(n) = uM_{a_1} \cdots M_{a_t} v, \tag{1}$$

which, incidentally, gives a fast algorithm for computing $b(n)$ for any n.

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

$$\begin{bmatrix}
1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}$$

Reproving (and fixing) a result of Fraenkel and Simpson

- M_0 has minimal polynomial

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

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

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

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

When we do so, we find

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

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

Theorem

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

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

for $n \geq 3$.

This statement corrects a small error in their paper.

Counting cube occurrences in finite Fibonacci words

In a similar way, we can count the cube occurrences in X_n. Using analysis exactly like the square case, we easily find

Theorem

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

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

where

$$d_1 = \frac{3-\sqrt{5}}{10} \qquad d_2 = \frac{17}{50}\sqrt{5} - \frac{3}{2}$$

$$d_3 = \frac{3+\sqrt{5}}{10} \qquad d_4 = -\frac{17}{50}\sqrt{5} - \frac{3}{2}$$

$$d_5 = 1 \qquad d_6 = -1.$$

Beyond Fibonacci... Tribonacci!

Define the Tribonacci numbers $(T_n)_{n\geq 0}$ by

$$T_n = \begin{cases} 0, & \text{if } n = 0; \\ 1, & \text{if } n = 1 \text{ or } n = 2; \\ T_{n-1} + T_{n-2} + T_{n-3}, & \text{if } n \geq 3. \end{cases}$$

Here are the first few terms:

n	0	1	2	3	4	5	6	7	8	9	10	11	12
T_n	0	1	1	2	4	7	13	24	44	81	149	274	504

Tribonacci representation

Theorem (Carlitz-Scoville-Hoggatt, 1972)

Every integer $n \geq 0$ has a unique representation as a sum of Tribonacci numbers of index ≥ 2, provided no three consecutive indices are used.

Thus, for example,

$$\begin{aligned} 43 &= T_7 + T_6 + T_4 + T_2 \\ &= 24 + 13 + 4 + 2. \end{aligned}$$

We can associate each such representation of n with a binary word $(n)_T$ indicating whether a term is included in the representation. Thus, $(43)_T = 110110$.

The infinite Tribonacci word

The *infinite Tribonacci word* **TR** is the fixed point, starting with 0, of the morphism

$$0 \to 01, \qquad 1 \to 02, \qquad 2 \to 0.$$

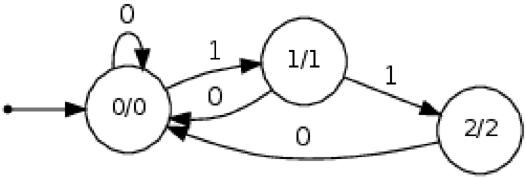
Here are the first few terms:

$$\mathbf{TR} = 01020100102010102010010201020100102010102010 \cdots$$

Alternatively, $\mathbf{TR}[n]$ can be computed by looking at the Tribonacci representation of n. It is

- 0, if the Tribonacci representation of n ends in a 0;
- 1, if the Tribonacci representation of n ends in a single 1;
- 2, if the Tribonacci representation of n ends in two 1's.

From the previous slide, it follows that **TR** can be computed by an automaton that takes, as input, the Tribonacci representation of n and outputs $\mathbf{TR}[n]$:



Any sequence that can be computed similarly is called *Tribonacci-automatic*.

Aperiodicity

Theorem

The word **TR** *is not ultimately periodic.*

Proof.

We construct a formula asserting that the integer $p \geq 1$ is a period of some suffix of **TR**:

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

The resulting automaton accepts nothing, so **TR** is not ultimately periodic. $\square$

Fourth powers

Theorem

TR *contains no fourth powers.*

Proof.

A formula for the orders of all fourth powers occurring in **TR**:

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

However, this did not run to completion on our prover. (It ran out of space while trying to determinize an NFA with 24904 states.)

Instead, substitute $j = i + t$, obtaining the new formula

$$(n > 0) \ \wedge \ \exists i \ \forall j \ ((j \geq i) \wedge (j < i + 3n)) \implies \mathbf{TR}[j] = \mathbf{TR}[j + n].$$

The resulting automaton accepts nothing, so there are no fourth powers. The largest intermediate automaton in the computation had 86711 states.

Orders of squares

The *order* of a square xx is $|x|$, the length of x.

Theorem (Glen, 2006)

All squares in $\mathbf{TR}$ *are of order* T_n *or* $T_n + T_{n-1}$ *for some* $n \geq 2$. *Furthermore, for all* $n \geq 2$, *there exists a square of order* T_n *and* $T_n + T_{n-1}$ *in* $\mathbf{TR}$.

Proof.

A natural formula for the orders of squares is

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

but this did not run to completion on our prover.
Instead, introduce a new variable $j = i + t$. This gives

$$(n > 0) \ \wedge \ \exists i \ \forall j \ ((i \leq j) \wedge (j < i + n)) \implies \mathbf{TR}[j] = \mathbf{TR}[j + n].$$

More about orders of squares

By modifying our previous formula, we get

$$(n > 0) \ \wedge \ \forall j \ ((i \leq j) \wedge (j < i + n)) \implies \mathbf{TR}[j] = \mathbf{TR}[j + n]$$

which encodes those (i, n) pairs such that there is a square of order n beginning at position i of **TR**.

This automaton has only 10 states and efficiently encodes both the orders and starting positions of each square in **TR**.

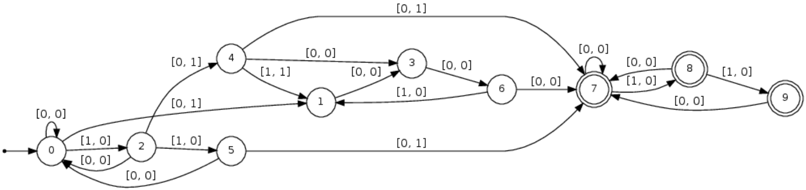
Thus we have proved the following new result:

Theorem

The language

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

is accepted by the following automaton:



Cubes

Theorem (Glen, 2006)

The cubes in **TR** *are of order* T_n *for* $n \geq 5$, *and a cube of each such order occurs.*

Proof.

We use the formula

$$(n > 0) \ \wedge \ \exists i \ \forall j \ ((i \leq j) \wedge (j < i + 2n)) \implies \mathbf{TR}[j] = \mathbf{TR}[j+n].$$

When we run our program, we obtain an automaton accepting exactly the language $(1000)0^*$, which corresponds to T_n for $n \geq 5$. The largest intermediate automaton had 60743 states. $\square$

Enumeration

We can also mechanically *enumerate* many properties of Tribonacci-automatic sequences.

For example, we can encode the factors having a given property property in terms of paths of an automaton. This gives the concept of *Tribonacci-regular sequence*.

Every Tribonacci-regular sequence $(a(n))_{n \geq 0}$ has a *linear representation* (u, μ, v) where u and v are row and column vectors, respectively, and $\mu : \Sigma_2 \to \mathbb{N}^{d \times d}$ is a matrix-valued morphism, where $\mu(0) = M_0$ and $\mu(1) = M_1$ are $d \times d$ matrices for some $d \geq 1$, such that

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

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

Enumeration

If $\mathbf{x}$ is an infinite word, the subword complexity function $\rho_{\mathbf{x}}(n)$ counts the number of distinct factors of length n.

Theorem

If $\mathbf{x}$ is Tribonacci-automatic, then the subword complexity function of $\mathbf{x}$ is Tribonacci-regular.

Using our implementation, we can obtain a linear representation of the subword complexity function for $\mathbf{TR}$. An obvious choice is to use the language

$$\{(n,i)_T \;:\; \forall j < i \; \mathbf{TR}[i..i+n-1] \neq \mathbf{TR}[j..j+n-1]\},$$

based on a formula that expresses the assertion that the factor of length n beginning at position i has never appeared before. Then, for each n, the number of corresponding i gives $\rho_{\mathbf{TR}}(n)$. However, this does not run to completion in our implementation.

Enumeration

Instead, substitute $u = j + t$ and and $k = i - j$ to get the formula

$$\forall k\ ((k > 0) \wedge (k \leq i)) \implies (\exists u\ ((u \geq j) \wedge (u < n + j) \wedge (\mathbf{TR}[u] \neq \mathbf{TR}[u + k]))).$$

This formula is close to the upper limit of what we can compute using our program.

The largest intermediate automaton had 1230379 states and the program took 12323.82 seconds, giving us a linear representation (u, μ, v) of rank 22.

When we minimize this representation...

We get the rank-12 linear representation

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

$$M_0 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ -2 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ -3 & 0 & 2 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ -4 & 0 & 2 & 0 & 2 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ -5 & 0 & 2 & 0 & 2 & 0 & 2 & 0 & 0 & 0 & 0 & 0 \\ -6 & 0 & 2 & 0 & 3 & 0 & 2 & 0 & 0 & 0 & 0 & 0 \\ -10 & 0 & 3 & 0 & 4 & 0 & 4 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad M_1 = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$v = [1\ 3\ 5\ 7\ 9\ 11\ 15\ 17\ 21\ 29\ 33\ 55]^R.$$

Comparing this to an independently-derived linear representation of the function $n \to 2n+1$, we see they are the same. Thus we get

Theorem (Droubay-Justin-Pirillo, 2001)

The subword complexity function of **TR** *is* $2n+1$.

The finite Tribonacci words

The *finite Tribonacci words* $(Y_n)_{n \geq 0}$ are defined as follows:

$$
\begin{aligned}
Y_0 &= \epsilon \\
Y_1 &= 2 \\
Y_2 &= 0 \\
Y_3 &= 01 \\
Y_n &= Y_{n-1} Y_{n-2} Y_{n-3} \text{ for } n \geq 4.
\end{aligned}
$$

Note that Y_n, for $n \geq 2$, is the prefix of length T_n of **TR**.

Our method can also prove interesting things about the finite Tribonacci words.

Counting the square occurrences in the finite Tribonacci words

What is the exact number of square occurrences in the finite Tribonacci words Y_n?

To solve this using our approach, we first *generalize* the problem to consider *any* length-n prefix of Y_n, and not simply the prefixes of length T_n.

The formula represents the number of distinct squares in $\mathbf{TR}[0..n-1]$:

$$\begin{aligned} L_{\mathrm{ds}} := \{(n,i,j)_T \ :\ & (j \geq 1) \text{ and } (i+2j \leq n) \\ & \text{and } \mathbf{TR}[i..i+j-1] = \mathbf{TR}[i+j..i+2j-1] \\ & \text{and } \forall i' < i \ \mathbf{TR}[i'..i'+2j-1] \neq \mathbf{TR}[i..i+2j-1]\}. \end{aligned}$$

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

Counting the square occurrences in the finite Tribonacci words

This represents the total number of occurrences of squares in $\mathbf{TR}[0..n-1]$:

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

This formula asserts that $\mathbf{TR}[i..i+2j-1]$ is a square occurring in $\mathbf{TR}[0..n-1]$.
Unfortunately, applying our enumeration method to this suffers from the same problem as before, so we rewrite it as

$$(j \geq 1) \wedge (i{+}2j \leq n) \wedge \forall u \, ((u \geq i) \wedge (u < i{+}j)) \implies \mathbf{TR}[u] = \mathbf{TR}[u{+}j]$$

When we compute the linear representation of the function counting the number of such i and j, we get a linear representation of rank 63.

Counting the square occurrences in the finite Tribonacci words

Now we compute the minimal polynomial of M_0, which is $(x-1)^2(x^2+x+1)^2(x^3-x^2-x-1)^2$. Solving a linear system in terms of the roots (or, more accurately, in terms of the sequences $1,\ n,\ T_n,\ T_{n-1},\ T_{n-2},\ nT_n,\ nT_{n-1},\ nT_{n-2}$) gives

Theorem

The total number of occurrences of squares in the Tribonacci word Y_n is

$$c(n) = \frac{n}{22}(9T_n - T_{n-1} - 5T_{n-2}) + \frac{1}{44}(-117T_n + 30T_{n-1} + 33T_{n-2}) + n - \frac{7}{4}$$

for $n \geq 5$.

Cube occurrences

In a similar way, we can count the occurrences of cubes in the finite Tribonacci word Y_n. Here we get a linear representation of rank 46. The minimal polynomial for M_0 is $x^4(x^3 - x^2 - x - 1)^2(x^2 + x + 1)^2(x - 1)^2$. Using analysis exactly like the square case, we find

Theorem

Let $C(n)$ denote the number of cube occurrences in the Tribonacci word Y_n. Then for $n \geq 3$ we have

$$C(n) = \frac{1}{44}(T_n + 2T_{n-1} - 33T_{n-2}) + \frac{n}{22}(-6T_n + 8T_{n-1} + 7T_{n-2}) + \frac{n}{6}$$
$$-\frac{1}{4}[n \equiv 0 \pmod 3] + \frac{1}{12}[n \equiv 1 \pmod 3] - \frac{7}{12}[n \equiv 2 \pmod 3].$$

Here $[P]$ is Iverson notation, and equals 1 if P holds and 0 otherwise.

Orders and positions of cubes

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

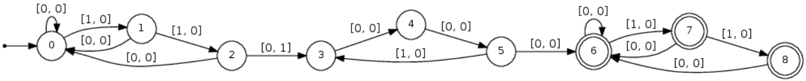
$$\{(i,n)_T \ : \ (n>0) \wedge \forall j\,((i \le j) \wedge (j < i+2n)) \implies \mathbf{TR}[j] = \mathbf{TR}[j+n]\}.$$

Theorem

The language

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

is accepted by the automaton below:



Palindromes

We now turn to a characterization of the palindromes in **TR**. Once again, it turns out that the obvious formula

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

resulted in an intermediate NFA of 5711 states that we could not successfully determinize.

Instead, we used two equivalent formulas. The first accepts n if there is an even-length palindrome, of length $2n$, centered at position i:

$$\exists i \geq n\ \forall j < n\ \mathbf{TR}[i+j] = \mathbf{TR}[i-j-1].$$

The second accepts n if there is an odd-length palindrome, of length $2n+1$, centered at position i:

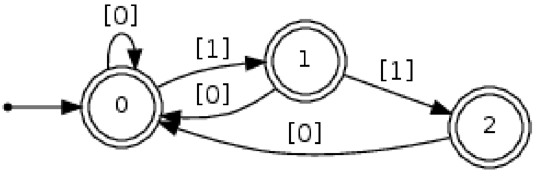
$$\exists i \geq n\ \forall j\ (1 \leq j \leq n) \implies \mathbf{TR}[i+j] = \mathbf{TR}[i-j].$$

Theorem

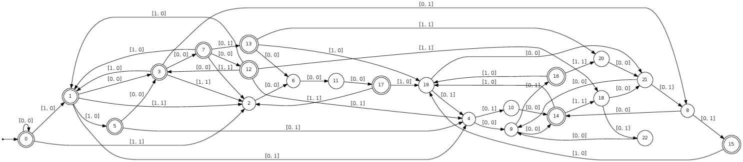
There exist palindromes of every length ≥ 0 *in* **TR**.

Proof.

For the first formula, our program outputs the automaton below. It clearly accepts the Tribonacci representations for all n.



We could also characterize the positions of all nonempty palindromes. To illustrate the idea, we generated an automaton accepting (i, n) such that $\mathbf{TR}[i-n..i+n-1]$ is an (even-length) palindrome.



Palindromic prefixes

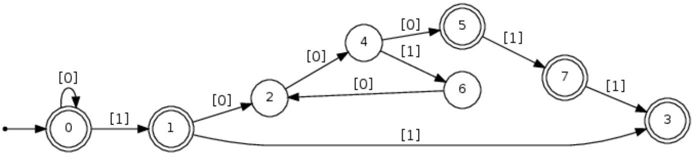
The prefixes are factors of particular interest. Let us determine which prefixes are palindromes:

Theorem

The prefix $\mathbf{TR}[0..n-1]$ *of length* n *is a palindrome if and only if* $n = 0$ *or* $(n)_T \in 1 + 11 + 10(010)^*(00 + 001 + 0011)$.

Proof.

We use the formula $\forall i < n \; \mathbf{TR}[i] = \mathbf{TR}[n-1-i]$. The automaton generated is given below.



Going even further

- Adders exist for numeration systems based on Pisot numbers: these are real numbers > 1 all of whose conjugates lie inside the unit circle. So we can create decision procedures for these numeration systems, too.
- The paperfolding words: this is an uncountable class of non-automatic sequences encoded by infinite words: we can prove theorems about uncountably many different sequences simultaneously!
- The Sturmian words: modulo a few details which still need to be proven, Luke Schaeffer could show that there is a decidable theory for these words, too.

In summary

- The logic-based approach gives a powerful way to state, decide, and enumerate properties of automatic sequences and their generalizations
- It allows proving, in generality, many particular cases that already appeared in the literature, using a unified framework
- Although the worst-case running time of the decision procedure is formidable, an implementation often succeeds in proving useful results