Mechanical Proofs of Properties of the Tribonacci Word

Hamoon Mousavi & Jeffrey Shallit
School of Computer Science, University of Waterloo
Waterloo, Ontario N2L 3G1, Canada
shallit@cs.uwaterloo.ca
http://www.cs.uwaterloo.ca/~shallit

Hamoon Mousavi

The Tribonacci numbers

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* $\mathbf{T}$ 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:

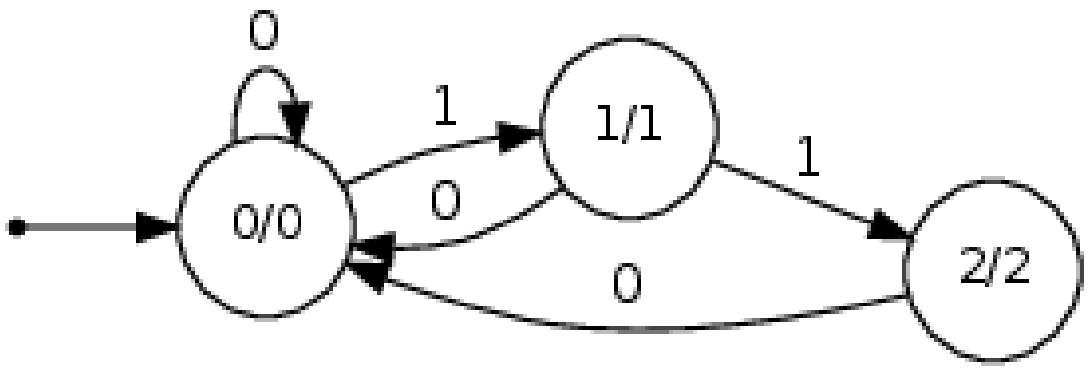
$$\mathbf{T} = 0102010010201010201001020102010010201010201 0 \cdots$$

Alternatively, $\mathbf{T}[n]$ 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.

Tribonacci-automatic sequences

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



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

Logic and automata

Given a Tribonacci-automatic sequence $\mathbf{x}$, the first-order logical theory

$$\mathrm{Th}(\mathbb{N}, +, n \to \mathbf{x}[n])$$

is decidable in much the same way that Presburger arithmetic is decidable, working with the Tribonacci representations of integers.

There is an adder for Tribonacci representations: an automaton that takes (in parallel) the Tribonacci representations of integers x, y, z and accepts if $z = x + y$ and rejects otherwise.

$\exists i\ P(i)$ can be implemented using nondeterminism, "guessing" the value of the variable i.

$\forall i\ P(i)$ can be implemented by $\neg\exists\neg P(i)$, which requires determinizing a nondeterministic machine (and hence potential exponential blowup).

Aperiodicity

Theorem

The word $\mathbf{T}$ *is not ultimately periodic.*

Proof.

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

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

The resulting automaton accepts nothing, so $\mathbf{T}$ is not ultimately periodic. $\square$

Aperiodicity

Here is the log of our program:

```
p >= 1 with 5 states, in 426ms
i >= n with 13 states, in 3ms
i + p with 150 states, in 31ms
TR[i] = TR[i + p] with 102 states, in 225ms
i >= n => TR[i] = TR[i + p] with 518 states, in 121ms
Ai i >= n => TR[i] = TR[i + p] with 4 states, in 1098ms
En Ai i >= n => TR[i] = TR[i + p] with 2 states, in 0ms
p >= 1 & En Ai i >= n => TR[i] = TR[i + p] with 2 states, in 1ms
overall time: 1905ms
```

The largest intermediate automaton during the computation had 5999 states.

Fourth powers

Theorem

$\mathbf{T}$ *contains no fourth powers.*

Proof.

A predicate for the orders of all fourth powers occurring in $\mathbf{T}$:

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

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 predicate

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

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 $\mathbf{T}$ *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* $\mathbf{T}$.

Proof.

A natural predicate for the orders of squares is

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

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

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

Orders of squares

Our program produces the following log:

```
i <= j with 13 states, in 10ms
i + n with 150 states, in 88ms
j < i + n with 229 states, in 652ms
i <= j & j < i + n with 241 states, in 42ms
j + n with 150 states, in 19ms
TR[j] = TR[j + n] with 102 states, in 61ms
i <= j & j < i + n => TR[j] = TR[j + n] with 1751 states, in 341ms
Aj i <= j & j < i + n => TR[j] = TR[j + n] with 11 states, in 4963ms
Ei Aj i <= j & j < i + n => TR[j] = TR[j + n] with 4 states, in 4ms
n > 0 & Ei Aj i <= j & j < i + n => TR[j] = TR[j + n] with 4 states, in 0ms
overall time: 6232ms
```

The resulting automaton accepts exactly the language $10^* + 110^*$.
The largest intermediate automaton had 26949 states. □

More about orders of squares

By modifying our previous predicate, we get

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

which encodes those (i, n) pairs such that there is a square of order n beginning at position i of $\mathbf{T}$.

This automaton has only 10 states and efficiently encodes both the orders and starting positions of each square in $\mathbf{T}$.

More about orders of squares

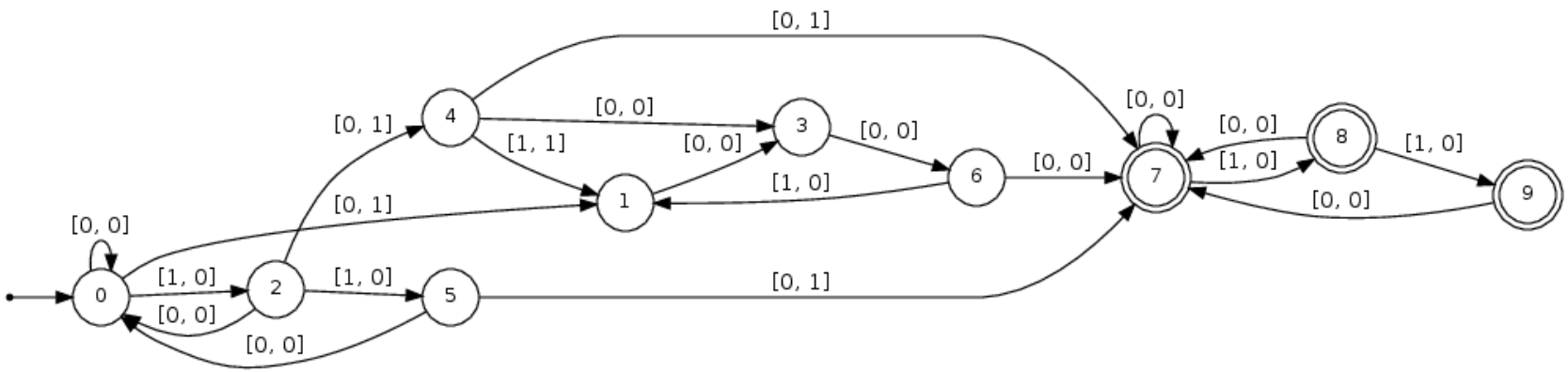
Thus we have proved the following new result:

Theorem

The language

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

is accepted by the following automaton:



Cubes

Theorem (Glen, 2006)

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

Proof.

We use the predicate

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

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. □

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 $\mathbf{T}$. An obvious choice is to use the language

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

based on a predicate 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 $\rho_{\mathbf{T}}(n)$.
However, this does not run to completion in our implementation.

Enumeration

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

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

This predicate 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) rank 22.

When we minimize this representation...

Enumeration

We get the rank-12 linear representation

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

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

$$M_1 = \begin{bmatrix}
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}$$

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

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 $\mathbf{T}$ *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 $\mathbf{T}$.

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

$$L_{\mathrm{ds}} := \{(n,i,j)_T \ : \ (j \geq 1) \text{ and } (i+2j \leq n) \\ \text{and } \mathbf{T}[i..i+j-1] = \mathbf{T}[i+j..i+2j-1] \\ \text{and } \forall i' < i \ \mathbf{T}[i'..i'+2j-1] \neq \mathbf{T}[i..i+2j-1]\}.$$

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

Counting the square occurrences in the finite Tribonacci words

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

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

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

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

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

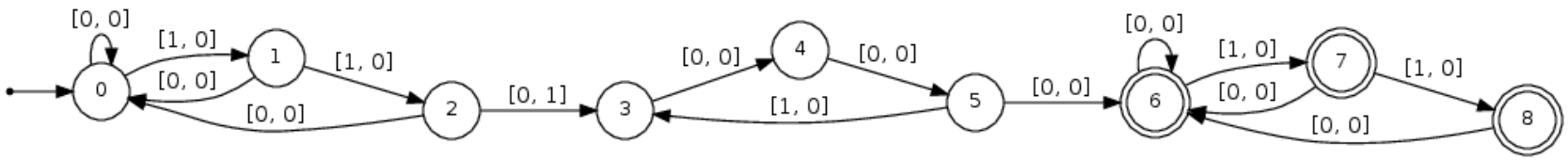
$$\{(i,n)_T \ : \ (n>0) \wedge \forall j \, ((i \leq j) \wedge (j < i+2n)) \implies \mathbf{T}[j] = \mathbf{T}[j+n]\}.$$

Theorem

The language

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

is accepted by the automaton below:



Palindromes

We now turn to a characterization of the palindromes in $\mathbf{T}$. Once again, it turns out that the obvious predicate

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

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

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

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

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

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

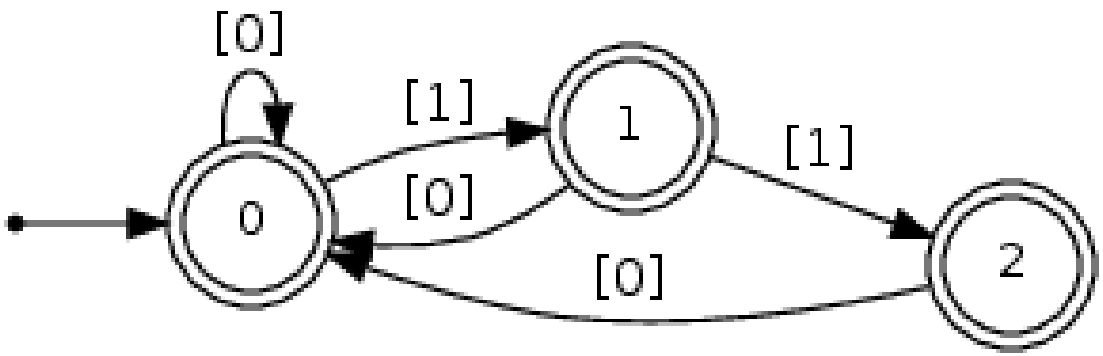
Palindromes

Theorem

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

Proof.

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



Palindromes

The log of our program follows.

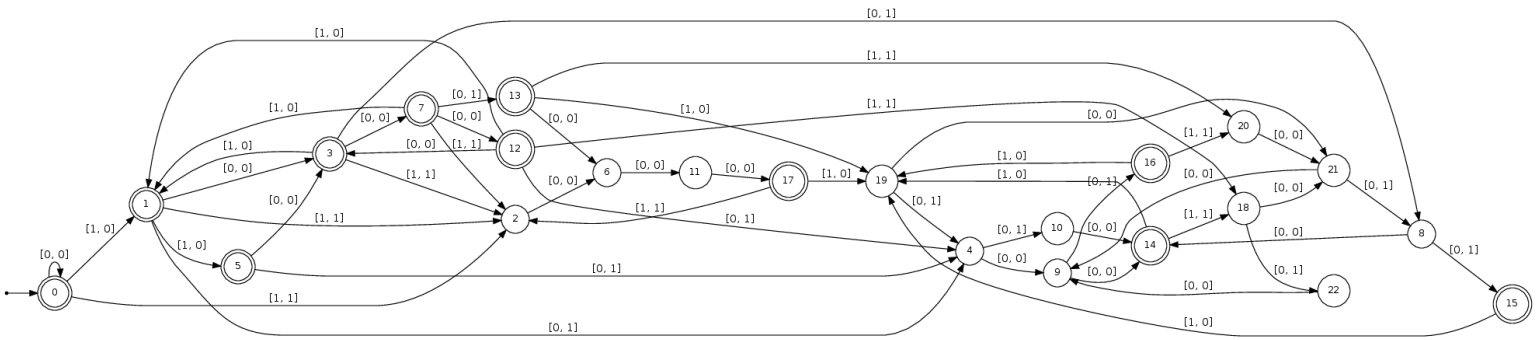
```
i >= n with 13 states, in 34ms
j < n with 13 states, in 8ms
i + j with 150 states, in 53ms
i - 1 with 7 states, in 155ms
i - 1 - j with 150 states, in 166ms
TR[i + j] = TR[i - 1 - j] with 664 states, in 723ms
j < n => TR[i + j] = TR[i - 1 - j] with 3312 states, in 669ms
Aj j < n => TR[i + j] = TR[i - 1 - j] with 24 states, in 5782274ms
i >= n & Aj j < n => TR[i + j] = TR[i - 1 - j] with 24 states, in 0ms
Ei i >= n & Aj j < n => TR[i + j] = TR[i - 1 - j] with 4 states, in 6ms
overall time: 5784088ms
```

The largest intermediate automaton had 918871 states. This was a fairly significant computation, taking about two hours' CPU time on a laptop.

The computation for the odd-length palindromes is quite similar.

Palindrome positions

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



Palindromic prefixes

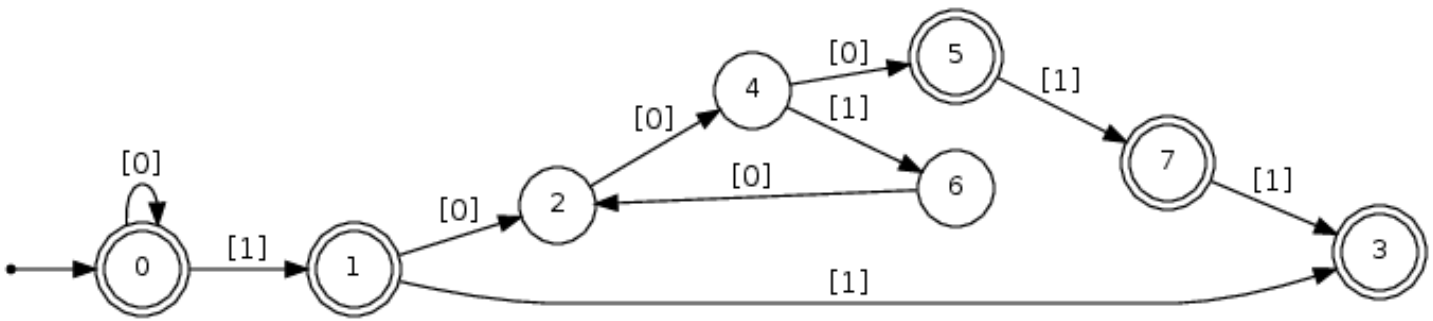
The prefixes are factors of particular interest. Let us determine which prefixes are palindromes:

Theorem

The prefix $\mathbf{T}[0..n-1]$ *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 predicate $\forall i < n\ \mathbf{T}[i] = \mathbf{T}[n-1-i]$. The automaton generated is given below.



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.

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 sometimes called *aligned covers*.

More precisely, for us $\mathbf{a} = a_0 a_1 a_2 \cdots$ is quasiperiodic if there exists x such that for all $i \geq 0$ there exists $j \geq 0$ with $i - n < j \leq i$ such that $a_j a_{j+1} \cdots a_{j+n-1} = x$, where $n = |x|$.

Such an x is called a *quasiperiod*.

Note that the condition $j \geq 0$ implies that, in this interpretation, any quasiperiod must actually be a prefix of **a**.

Quasiperiods

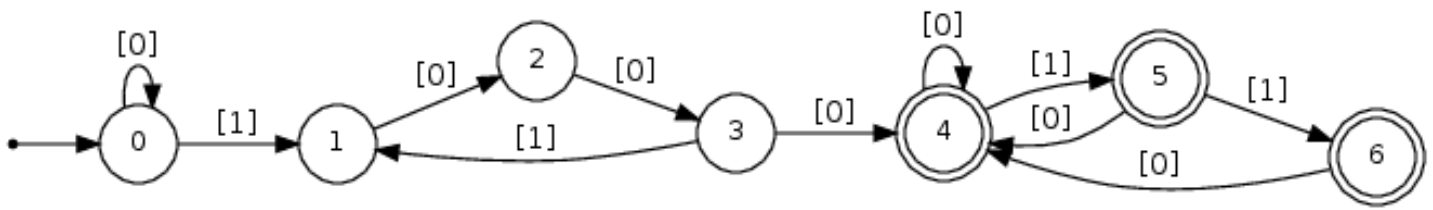
Glen, Levé, and Richomme (2008) characterized the quasiperiods of a large class of words, including the Tribonacci word.

However, their characterization did not explicitly give the *lengths* of the quasiperiods.

We do that in the following new result.

Theorem

A nonempty length-n prefix of **T** *is a quasiperiod of* **T** *if and only if n is accepted by the following automaton:*



Quasiperiods

Proof.

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

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

When we do this, we get the automaton on the previous slide.

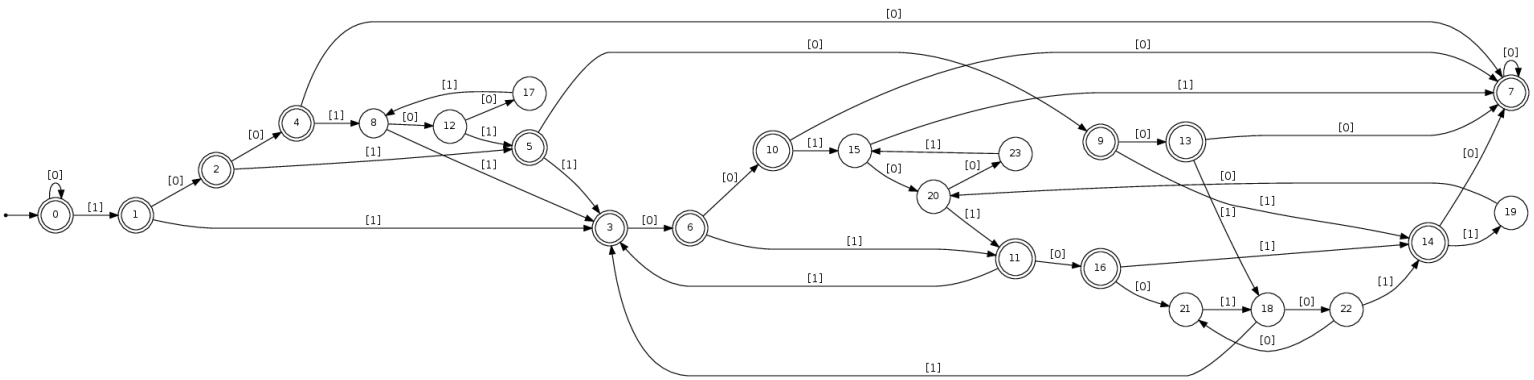
These numbers are those i for which $T_n \leq i \leq U_n$ for $n \geq 5$, where $U_2 = 0$, $U_3 = 1$, $U_4 = 3$, and $U_n = U_{n-1} + U_{n-2} + U_{n-3} + 3$ for $n \geq 5$. □

Unbordered factors

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

Theorem

There is an unbordered factor of length n of $\mathbf{T}$ if and only if $(n)_T$ is accepted by the automaton given below.



Unbordered factors

Proof.

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

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

However, this does not run to completion within the available space on our prover. Instead, make the substitutions $t' = n - j$ and $u = i + t$. This gives the predicate

$$\exists i\ \forall t',\ n/2 \leq t' < n,\ \exists u,\ (i \leq u < i + n - t')\ \mathbf{T}[u] \neq \mathbf{T}[u + t'].$$

□

Unbordered factors

Here is the log:

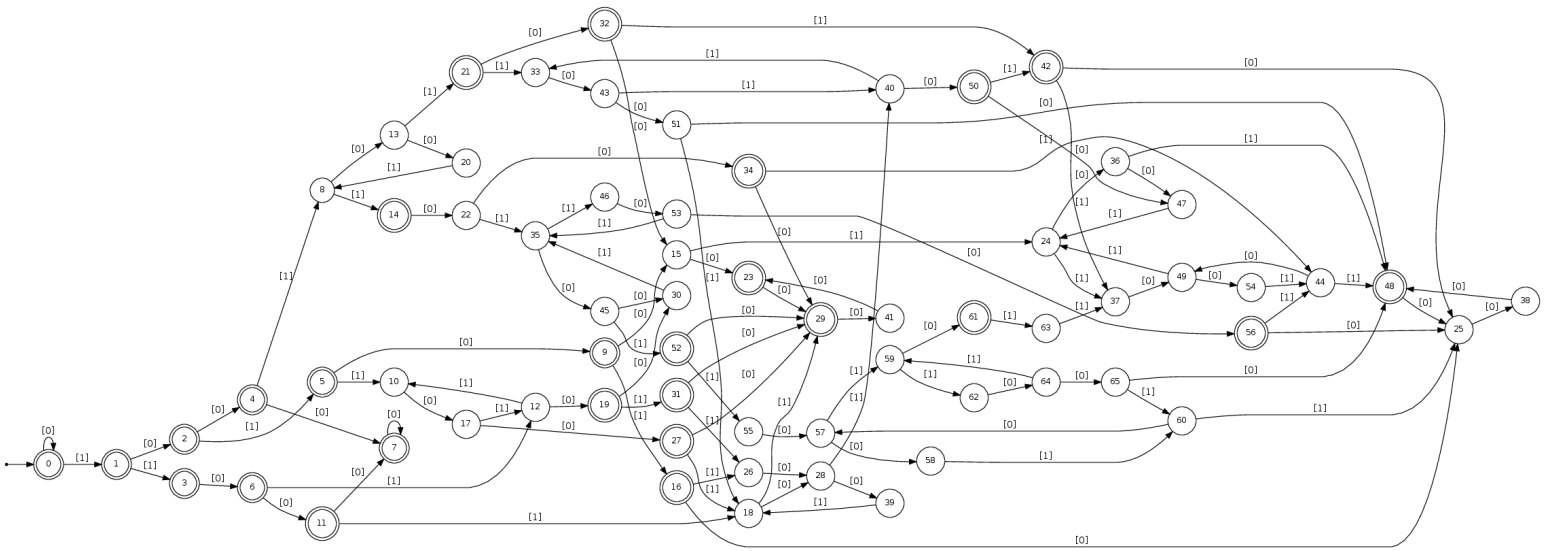
```
2 * t with 61 states, in 276ms
n <= 2 * t with 79 states, in 216ms
t < n with 13 states, in 3ms
n <= 2 * t & t < n with 83 states, in 9ms
u >= i with 13 states, in 7ms
i + n with 150 states, in 27ms
i + n - t with 1088 states, in 7365ms
u < i + n - t with 1486 states, in 6041ms
u >= i & u < i + n - t with 1540 states, in 275ms
u + t with 150 states, in 5ms
TR[u] != TR[u + t] with 102 states, in 22ms
u >= i & u < i + n - t & TR[u] != TR[u + t] with 7489 states, in 3364ms
Eu u >= i & u < i + n - t & TR[u] != TR[u + t] with 552 states, in 5246873ms
n <= 2 * t & t < n => Eu u >= i & u < i + n - t & TR[u] != TR[u + t] with 944 states, in 38ms
At n <= 2 * t & t < n => Eu u >= i & u < i + n - t & TR[u] != TR[u + t] with 47 states, in 1184ms
Ei At n <= 2 * t & t < n => Eu u >= i & u < i + n - t & TR[u] != TR[u + t] with 25 states, in 2ms
overall time: 5265707ms
```

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.

Theorem

There is a factor of length n of $\mathbf{T}$ *that is Lyndon if and only if n is accepted by the automaton given below.*



Proof.

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

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

Unfortunately this predicate did not run to completion, so we substituted $u' := i + u$ to get

$$\exists i\ \forall j, 1 \leq j < n,\ \exists t < n - j$$
$$(\forall u', i \leq u' < i + t\ \mathbf{T}[u'] = \mathbf{T}[u' + j])\ \wedge\ \mathbf{T}[i + t] < \mathbf{T}[i + j + t].$$

□

Critical exponent

Define $\exp(w) = |w|/P$, 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 $\exp(w)$.

Theorem (Tan-Wen, 2007)

The critical exponent of $\mathbf{T}$ *is* $\rho \doteq 3.191487883953118747 06$, *the real zero of the polynomial* $2x^3 - 12x^2 + 22x - 13$.

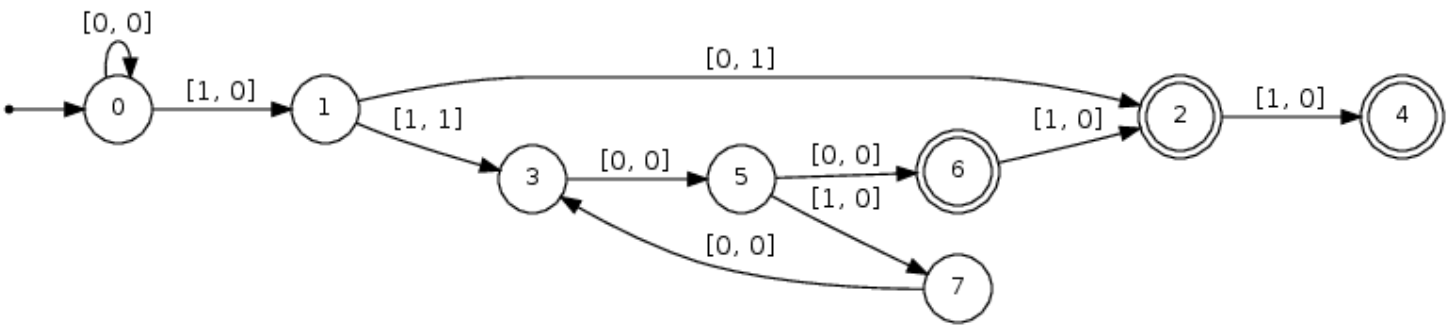
Proof. Let x be any factor of exponent ≥ 3 in $\mathbf{T}$. Let $n = |x|$ and p be the period, so that $n/p \geq 3$. Then by considering the first $3p$ symbols of x, which form a cube, we know that $p = T_n$.

Critical exponent

So it suffices to determine the largest n corresponding to every p of the form T_n. We did this using the predicate

$$(n > p) \wedge (p \geq 1) \wedge (\exists i\, \forall j\, ((j \geq i) \wedge (j + p < i + n)) \implies (\mathbf{T}[j] = \mathbf{T}[j + p])) \wedge (\forall i'\, (\exists j'\, (j' \geq i') \wedge (j' + p \leq i' + n) \wedge (\mathbf{T}[j'] \neq \mathbf{T}[j' + p]))).$$

This gives the automaton



Critial exponent

From inspection of the automaton, we see that the maximum length of a factor $n = U_j$ having period $p = T_j$, $j \geq 2$, is given by

$$U_j = \begin{cases} 2, & \text{if } j = 2; \\ 5, & \text{if } j = 3; \\ [110(100)^{i-1}0]_T, & \text{if } j = 3i+1 \geq 4; \\ [110(100)^{i-1}01]_T, & \text{if } j = 3i+2 \geq 5; \\ [110(100)^{i-1}011]_T, & \text{if } j = 3i+3 \geq 6. \end{cases}$$

A tedious induction shows that U_j satisfies the linear recurrence $U_j = U_{j-1} + U_{j-2} + U_{j-3} + 3$ for $j \geq 5$. Hence we can write U_j as a linear combination of Tribonacci sequences and the constant sequence 1, and solving for the constants we get

$$U_j = \frac{5}{2} T_j + T_{j-1} + \frac{1}{2} T_{j-2} - \frac{3}{2}$$

for $j \geq 2$.

Critical exponent

The critical exponent of T is then $\sup_{j\geq 1} U_j/T_j$.

Now

$$U_j/T_j = \frac{5}{2} + \frac{T_{j-1}}{T_j} + \frac{T_{j-2}}{2T_j} - \frac{3}{2T_j} = \frac{5}{2} + \alpha^{-1} + \frac{1}{2}\alpha^{-2} + O(1.8^{-j}).$$

Hence U_j/T_j tends to $5/2 + \alpha^{-1} + \frac{1}{2}\alpha^{-2} = \rho$. □

Initial critical exponent

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

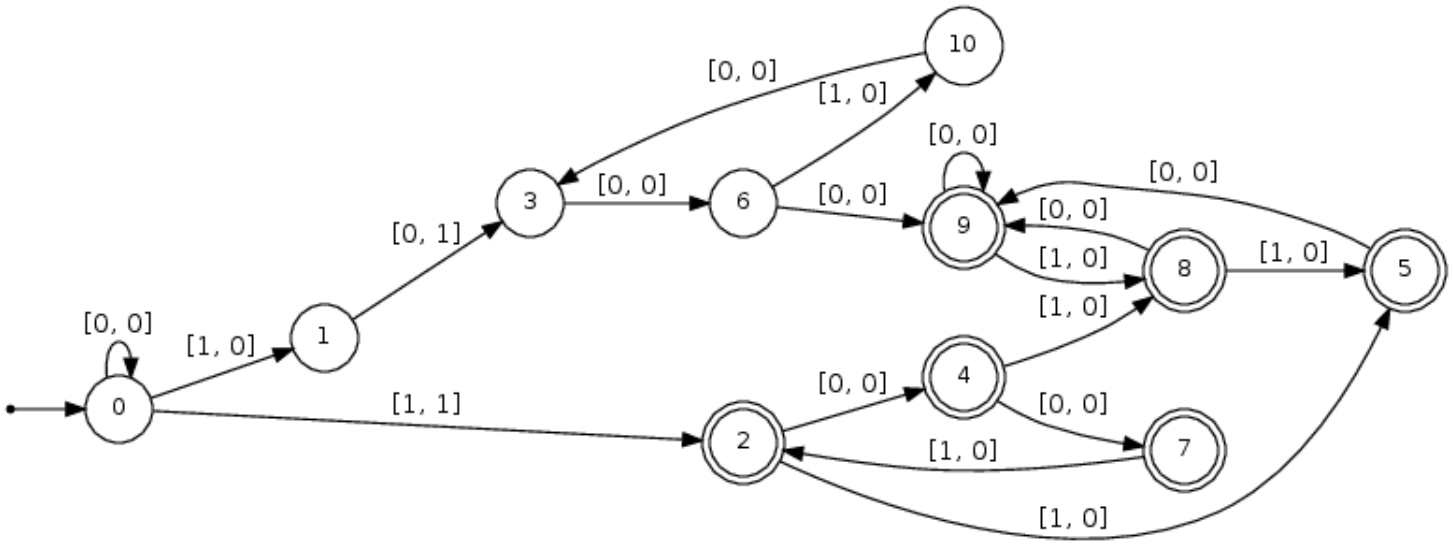
The initial critical exponent of $\mathbf{T}$ *is* $\rho - 1$.

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

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

An analysis similar to that we gave above for the critical exponent gives the result.

Initial critical exponent



Theorem

The only prefixes of the Tribonacci word that are powers are those of length $2T_n$ for $n \geq 5$.

Initial critical exponent

Proof.

The predicate

$$\exists d < n\,(\forall j < n-d\ \mathbf{T}[j] = \mathbf{T}[d+j]) \wedge (\forall k < d\ \mathbf{T}[k] = \mathbf{T}[n-d+k])$$

asserts that the prefix $\mathbf{T}[0..n-1]$ is a power. When we run this through our program, the resulting automaton accepts 100010^*, which corresponds to $F_{n+1} + F_{n-3} = 2T_n$ for $n \geq 5$. □

Abelian properties

We can derive some results about the abelian properties of the Tribonacci word $\mathbf{T}$ by making use of the following theorem:

Theorem

Let n be a non-negative integer and let $e_1 e_2 \cdots e_j$ be a Tribonacci representation of n, possibly with leading zeros, with $j \geq 3$. Then

(a) $|\mathbf{T}[0..n-1]|_0 = [e_1 e_2 \cdots e_{j-1}]_T + e_j$.

(b) $|\mathbf{T}[0..n-1]|_1 = [e_1 e_2 \cdots e_{j-2}]_T + e_{j-1}$.

(c) $|\mathbf{T}[0..n-1]|_2 = [e_1 e_2 \cdots e_{j-3}]_T + e_{j-2}$.

Proof.

By induction. □

Abelian properties

The *Parikh vector* $\psi(x)$ of a word x over an ordered alphabet $\Sigma = \{a_1, a_2, \ldots, a_k\}$ is defined to be $(|x|_{a_1}, \ldots, |x|_{a_k})$, the number of occurrences of each letter in x.

The *abelian complexity function* $\rho^{\mathrm{ab}}_{\mathbf{w}}(n)$ counts the number of distinct Parikh vectors of the length-n factors of an infinite word $\mathbf{w}$.

Corollary (Turek, 2015)

The abelian complexity function of $\mathbf{T}$ *is Tribonacci-regular.*

Abelian properties

Proof.

First, from above there exists an automaton TAB such that $(n, i, j, k)_T$ is accepted iff $(i, j, k) = \psi(\mathbf{T}[0..n-1])$. In fact, such an automaton has 32 states.

Using this automaton, we can create a predicate $P(n, i)$ such that the number of i for which $P(n, i)$ is true equals $\rho^{\mathrm{ab}}_{\mathbf{T}}(n)$. For this we assert that i is the least index at which we find an occurrence of the Parikh vector of $\mathbf{T}[i..i+n-1]$:

$$\forall i' < i \; \exists a_0, a_1, a_2, b_0, b_1, b_2, c_0, c_1, c_2, d_0, d_1, d_2$$
$$\mathrm{TAB}(i+n, a_0, a_1, a_2) \wedge \mathrm{TAB}(i, b_0, b_1, b_2) \wedge \mathrm{TAB}(i'+n, c_0, c_1, c_2) \wedge$$
$$\mathrm{TAB}(i', d_0, d_1, d_2) \wedge ((a_0 - b_0 \neq c_0 - d_0) \vee (a_1 - b_1 \neq c_1 - d_1) \vee$$
$$(a_2 - b_2 \neq c_2 - d_2)).$$

Abelian properties

Remark

In principle we could mechanically compute the Tribonacci-regular representation of the abelian complexity function using this technique, but with our current implementation this is not computationally feasible.

Theorem

Any morphic image of the Tribonacci word is Tribonacci-automatic.

Things we could not do yet

There are a number of things we have not succeeded in computing with our prover because it ran out of space. These include

- mirror invariance of $\mathbf{T}$ (that is, if x is a finite factor then so is x^R);
- Counting the number of special factors of length n (although it can be deduced from the subword complexity function);
- statistics about, e.g, lengths of squares, cubes, etc., in the "flipped" Tribonacci sequence, the fixed point of $0 \to 01$, $1 \to 20$, $2 \to 0$;
- recurrence properties of the Tribonacci word;
- counting the number of distinct squares (not occurrences) in the finite Tribonacci word Y_n;
- abelian complexity of the Tribonacci word.

In the future, an improved implementation may succeed in resolving these in a mechanical fashion.

Conclusions

1. A mechanical decision procedure for k-automatic sequences, Fibonacci-automatic sequences, Tribonacci-automatic sequences, etc., despite its horrible worst-case running time, can often be used to prove results of genuine interest in combinatorics on words.
2. Many existing results in the literature, which had long and complicated case-based proofs, can be proven in a uniform way using the decision procedure.
3. Many new results, including about enumeration, can be proven.
4. We can use the decision procedure to prove things about the finite Tribonacci words, too.