Transduction of Automatic Sequences and Applications

Jeffrey Shallit[1] Anatoly Zavyalov[2]

[1]University of Waterloo

[2]University of Toronto

CIAA 2023

Problem

For which $n \in \mathbb{Z}^+$ is $n!$ the sum of three squares?

$$n! = a^2 + b^2 + c^2 \qquad \text{for some } a, b, c \in \mathbb{Z}$$

Problem

For which $n \in \mathbb{Z}^+$ is $n!$ the sum of three squares?

$$n! = a^2 + b^2 + c^2 \qquad \text{for some } a, b, c \in \mathbb{Z}$$

In 2010, Deshouillers and Luca showed that the density of such n is about 7/8:

How Often is $n!$ a Sum of Three Squares?

Jean-Marc Deshouillers and Florian Luca

Theorem 1. *The estimate*

$$\#\{n \leq x : n! \text{ is a sum of three squares}\} = 7x/8 + O(x^{2/3})$$

holds.

Problem

- Using automata theory, we can exactly characterize the n for which $n!$ is a sum of three squares, and get better bounds.

Problem

- Using automata theory, we can exactly characterize the n for which $n!$ is a sum of three squares, and get better bounds.
- Burns (2022) provided an ad hoc way to characterize such n using automata theory, and provides better bounds for this problem:

Factorials and Legendre's three-square theorem: II

Rob Burns

31st March 2022

Abstract

Let $\bar{S}$ denote the set of integers n such that $n!$ cannot be written as a sum of three squares. Let $\bar{S}(n)$ denote $\bar{S} \cap [1, n]$. We establish an exact formula for $\bar{S}(2^k)$ and show that $\bar{S}(n) = 1/8 * n + \mathcal{O}(\sqrt{n})$. We also list the lengths of gaps appearing in $\bar{S}$. We make use of the software package Walnut to establish these results.

Problem

- Using automata theory, we can exactly characterize the n for which $n!$ is a sum of three squares, and get better bounds.
- Burns (2022) provided an ad hoc way to characterize such n using automata theory, and provides better bounds for this problem:

Factorials and Legendre's three-square theorem: II

Rob Burns

31st March 2022

Abstract

Let $\bar{S}$ denote the set of integers n such that $n!$ cannot be written as a sum of three squares. Let $\bar{S}(n)$ denote $\bar{S} \cap [1, n]$. We establish an exact formula for $\bar{S}(2^k)$ and show that $\bar{S}(n) = 1/8 * n + \mathcal{O}(\sqrt{n})$. We also list the lengths of gaps appearing in $\bar{S}$. We make use of the software package Walnut to establish these results.

- We provide a more general and systematic procedure to characterize this set and solve similar problems using transducers.

What is `Walnut`?

We use `Walnut` to create and perform operations on automata algorithmically.

`Walnut`:

- Is an free and open-source software written in Java, originally designed by Hamoon Mousavi.
- Rigorously proves theorems about automatic sequences.
- Has additions and changes by Aseem Raj Baranwal, Laindon C. Burnett, Kai Hsiang Yang, and Anatoly Zavyalov.
- Is available for free download at
  `https://cs.uwaterloo.ca/~shallit/walnut.html` .

Automatic sequences

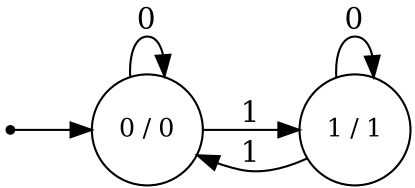
- A sequence $(a_n)_{n\geq 0}$ over a finite alphabet Σ is *k-automatic* if there exists a deterministic finite automaton with output (DFAO) that reaches a state with output a_n upon reading an input of $(n)_k$ (base-k representation of n).

Automatic sequences

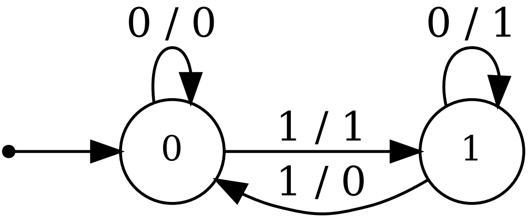
- A sequence $(a_n)_{n\geq 0}$ over a finite alphabet Σ is k-automatic if there exists a deterministic finite automaton with output (DFAO) that reaches a state with output a_n upon reading an input of $(n)_k$ (base-k representation of n).
- For example, the Thue-Morse sequence

$$\mathbf{t} = 0110100110010110\cdots$$

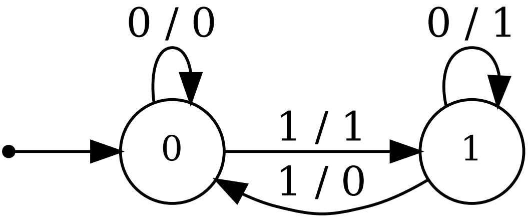
is 2-automatic, generated by the following DFAO:



Transducers

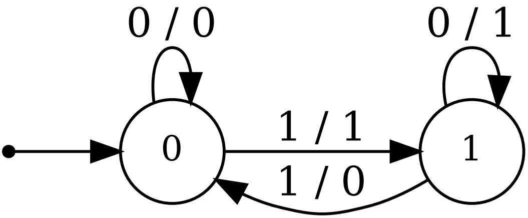


- A (1-uniform deterministic finite-state) transducer (*transducer* for short) can be viewed as a DFA with an output function on each transition.



- A (1-uniform deterministic finite-state) transducer (*transducer* for short) can be viewed as a DFA with an output function on each transition.
- Every transition has an output of a *single symbol*.

Transducers



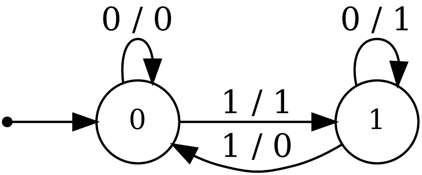
- A (1-uniform deterministic finite-state) transducer (*transducer* for short) can be viewed as a DFA with an output function on each transition.
- Every transition has an output of a *single symbol*.
- Unlike general models, the transducers we use are **finite-state**, **deterministic**, and output **one symbol per transition**.

Transducer

The following transducer calculates the running sum (mod 2) of a sequence:

$$\mathbf{t} = 0110100110010110\cdots$$

$$\downarrow$$



$$\downarrow$$

$$T(\mathbf{t}) = 0100111011100100\cdots$$

Automatic sequences are closed under transduction

- **Dekking (1994)**: Automatic sequences are closed under transduction.

Iteration of maps by an automaton

F.M. Dekking

Department of Mathematics and Informatics, Delft University of Technology, Mekelweg 4, 2628 CD Delft, Netherlands

Received 2 August 1991

Theorem A. *If* x *is q-automatic, then* $z=(\varphi_{x_1 \cdots x_n})$ *is q-automatic.*

Automatic sequences are closed under transduction

- **Dekking (1994)**: Automatic sequences are closed under transduction.

Iteration of maps by an automaton

F.M. Dekking

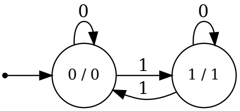
Department of Mathematics and Informatics, Delft University of Technology, Mekelweg 4, 2628 CD Delft, Netherlands

Received 2 August 1991

Theorem A. *If x is q-automatic, then $z = (\varphi_{x_1 \cdots x_n})$ is q-automatic.*

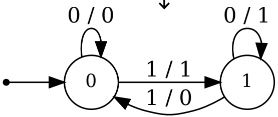
- Dekking provides a constructive proof, which is implemented algorithmically into `Walnut`.

"Transducing" an automaton



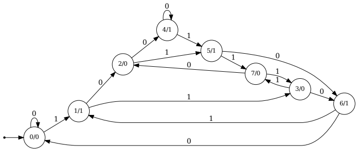
$\mathbf{t} = 0110100110010110\cdots$

$\downarrow$



$\downarrow$

$T(\mathbf{t}) = 0100111011100100\cdots$



(): transduce TSUM1 RUNSUM2 T;

Back to the problem

We want to create an automaton that accepts the binary representation of $n \in \mathbb{N}$ if and only if

$$n! = x^2 + y^2 + z^2$$

for some $x, y, z \in \mathbb{Z}$.

Legendre's three-square theorem

A natural number $n \in \mathbb{N}$ is a sum of three squares of integers

$$n = x^2 + y^2 + z^2, \text{ for some } x, y, z \in \mathbb{Z}$$

if and only if n is **not** of the form $n = 4^a(8b+7)$ for nonnegative integers a and b.

Adrien-Marie Legendre
Credit: Wikipedia

Binary representation of sums of three squares

- A natural number $n \in \mathbb{N}$ is **not** a sum of three squares if and only if $n = 4^a(8b+7)$ for nonnegative integers a, b.

Binary representation of sums of three squares

- A natural number $n \in \mathbb{N}$ is **not** a sum of three squares if and only if $n = 4^a(8b+7)$ for nonnegative integers a, b.
- So, $n \in \mathbb{N}$ is **not** a sum of three squares if and only if its binary representation is of the form

$$(n)_2 = \underbrace{\cdots}_{\in\{0,1\}^*} 111 \underbrace{00\cdots00}_{\substack{\text{even } \# \text{ of 0's,} \\ \text{may be } \varepsilon}} .$$

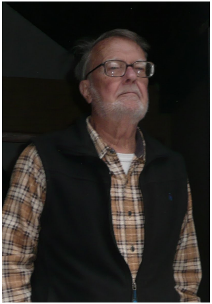
Cobham's observation

Cobham (1972) showed that the sequence of n that are sums of three squares are 2-automatic!

Example 8. Sums of three squares. A natural number can be represented as the sum of three perfect squares if and only if it is not representable in the from $4^n(8k+7)$, k, $n \in N$ [28, Chapter XIII]. A number is not representable in the form $4^n(8k+7)$ if and only if its binary representation does not terminate with three successive 1's followed by an even number of 0's. Using this observation, we can construct an automaton which recognizes the set of sums of three squares. This automaton has six states, a transition function δ defined by the table

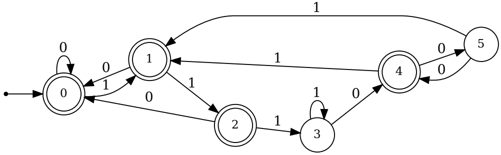
δ	s_1	s_2	s_3	s_4	s_5	s_6
0	s_1	s_1	s_1	s_5	s_6	s_5
1	s_2	s_3	s_4	s_4	s_2	s_2

and the set of designated states $F_1 = \{s_1, s_2, s_3, s_5\}$.



Alan Cobham (1927-2011)
Credit: Jeffrey Shallit

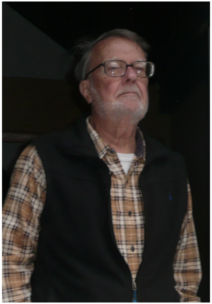
Cobham's observation



The above DFA rejects $(n)_2$ iff it is of the form

$$(n)_2 = \underbrace{\cdots}_{\in\{0,1\}^*} \; 111 \; \underbrace{00\cdots00}_{\substack{\text{even \# of 0's,} \\ \text{may be } \varepsilon}} \, ,$$

so $(n)_2$ is accepted iff n is a sum of three squares.



Alan Cobham (1927-2011)
Credit: Jeffrey Shallit

Characterizing sums of three squares

- Any $n \geq 1$ can be written as

$$n = g(n) \cdot 2^{\nu_2(n)}$$

where $g(n)$ is odd.

Characterizing sums of three squares

- Any $n \geq 1$ can be written as

$$n = g(n) \cdot 2^{\nu_2(n)}$$

  where $g(n)$ is odd.
- For example, $40 = 5 \cdot 2^3$, so $g(40) = 5$ and $\nu_2(40) = 3$.

Characterizing sums of three squares

- Any $n \geq 1$ can be written as

$$n = g(n) \cdot 2^{\nu_2(n)}$$

  where $g(n)$ is odd.
- For example, $40 = 5 \cdot 2^3$, so $g(40) = 5$ and $\nu_2(40) = 3$.
- Easy to show: $\nu_2(mn) = \nu_2(m) + \nu_2(n)$ for $m, n \geq 1$.

Characterizing sums of three squares

- Any $n \geq 1$ can be written as
$$n = g(n) \cdot 2^{\nu_2(n)}$$
where $g(n)$ is odd.
- For example, $40 = 5 \cdot 2^3$, so $g(40) = 5$ and $\nu_2(40) = 3$.
- Easy to show: $\nu_2(mn) = \nu_2(m) + \nu_2(n)$ for $m, n \geq 1$.

$$\Longrightarrow \boxed{\nu_2(n!) = \sum_{i=1}^{n} \nu_2(i)}$$

Characterizing sums of three squares

- Any $n \geq 1$ can be written as

$$n = g(n) \cdot 2^{\nu_2(n)}$$

  where $g(n)$ is odd.
- For example, $40 = 5 \cdot 2^3$, so $g(40) = 5$ and $\nu_2(40) = 3$.
- Easy to show: $\nu_2(mn) = \nu_2(m) + \nu_2(n)$ for $m, n \geq 1$.

$$\Longrightarrow \boxed{\nu_2(n!) = \sum_{i=1}^{n} \nu_2(i)}$$

- $g(n) \bmod 8 \in \{1, 3, 5, 7\}$ for $n \geq 1$ and $g(mn) \equiv g(m)g(n) \pmod 8$.

Characterizing sums of three squares

- Any $n \geq 1$ can be written as
$$n = g(n) \cdot 2^{\nu_2(n)}$$
where $g(n)$ is odd.
- For example, $40 = 5 \cdot 2^3$, so $g(40) = 5$ and $\nu_2(40) = 3$.
- Easy to show: $\nu_2(mn) = \nu_2(m) + \nu_2(n)$ for $m, n \geq 1$.

$$\Longrightarrow \boxed{\nu_2(n!) = \sum_{i=1}^{n} \nu_2(i)}$$

- $g(n) \bmod 8 \in \{1, 3, 5, 7\}$ for $n \geq 1$ and $g(mn) \equiv g(m)g(n) \pmod 8$.

$$\Longrightarrow \boxed{g(n!) \equiv \prod_{i=1}^{n} g(i) \pmod 8}$$

Characterizing sums of three squares

- Let S_3 be the set of natural numbers that are sums of three squares.

Characterizing sums of three squares

- Let S_3 be the set of natural numbers that are sums of three squares.
- Let $S = \{n : n! \in S_3\}$ be the set of n such that $n!$ is a sum of three squares.

Characterizing sums of three squares

- Let S_3 be the set of natural numbers that are sums of three squares.
- Let $S = \{n : n! \in S_3\}$ be the set of n such that $n!$ is a sum of three squares.
- We have $n \notin S_3$ iff $(n)_2 = \underbrace{\cdots}_{\in\{0,1\}^*} 1\,1\,1 \underbrace{0\,0 \cdots 0\,0}_{\substack{\text{even \# of 0's,} \\ \text{may be } \varepsilon}}$.

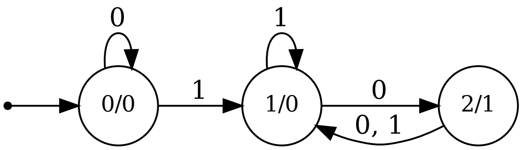
Characterizing sums of three squares

- Let S_3 be the set of natural numbers that are sums of three squares.
- Let $S = \{n : n! \in S_3\}$ be the set of n such that $n!$ is a sum of three squares.
- We have $n \notin S_3$ iff $(n)_2 = \underbrace{\cdots}_{\in\{0,1\}^*} 111 \underbrace{00\cdots00}_{\substack{\text{even \# of 0's,} \\ \text{may be } \varepsilon}}$.
- This is equivalent to $\nu_2(n) \equiv 0 \pmod 2$ and $g(n) \equiv 7 \pmod 8$.

Characterizing sums of three squares

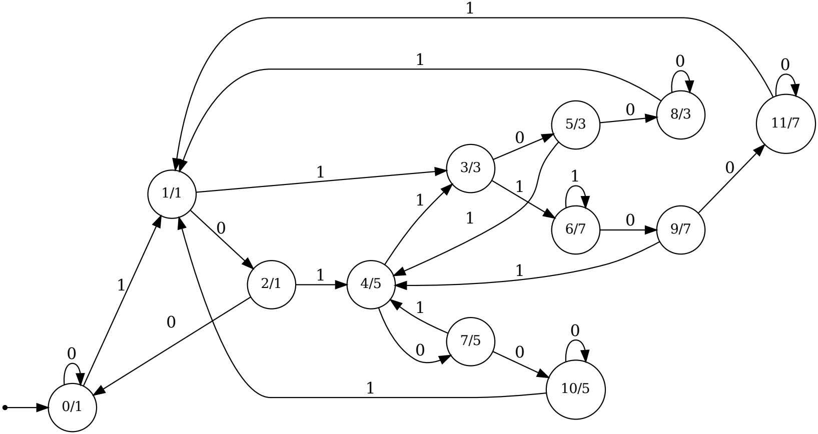
- Let S_3 be the set of natural numbers that are sums of three squares.
- Let $S = \{n : n! \in S_3\}$ be the set of n such that $n!$ is a sum of three squares.
- We have $n \notin S_3$ iff $(n)_2 = \underbrace{\cdots}_{\in\{0,1\}^*} 111 \underbrace{00\cdots00}_{\substack{\text{even \# of 0's,} \\ \text{may be } \varepsilon}}$.
- This is equivalent to $\nu_2(n) \equiv 0 \pmod 2$ and $g(n) \equiv 7 \pmod 8$.
- Thus, $n \notin S$ if and only if $\nu_2(n!) = \sum_{i=1}^{n} \nu_2(i) \equiv 0 \pmod 2$ and $g(n!) = \prod_{i=1}^{n} g(i) \equiv 7 \pmod 8$.

Define the DFAO `NU_MOD2`, which generates the sequence $(\nu_2(n) \bmod 2)_{n \geq 1}$:



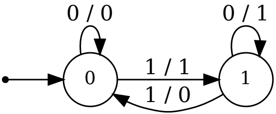
In Walnut

Define the DFAO G_MOD8, which generates the sequence $(g(n) \bmod 8)_{n \geq 1}$:



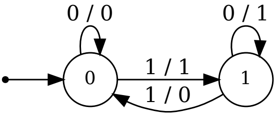
In `Walnut`

- Define the transducer `RUNSUM2`, which generates the running sum mod 2:

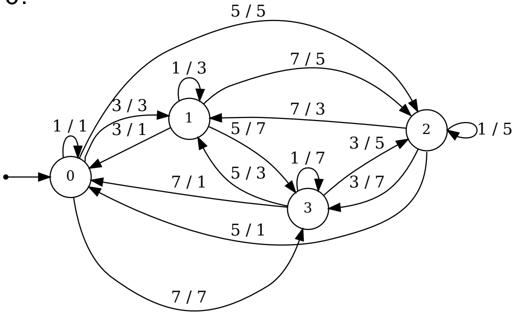


In Walnut

- Define the transducer RUNSUM2, which generates the running sum mod 2:



- Define the transducer RUNPROD1357, which generates the running product mod 8:



In Walnut

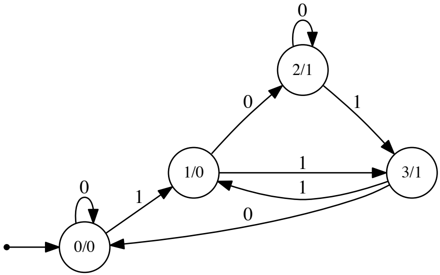
- Now, transduce NU_MOD2 with RUNSUM2 to get the DFAO NU_RUNSUM using the Walnut command

```
transduce NU_RUNSUM RUNSUM2 NU_MOD2;
```

In Walnut

- Now, transduce NU_MOD2 with RUNSUM2 to get the DFAO NU_RUNSUM using the Walnut command

```
transduce NU_RUNSUM RUNSUM2 NU_MOD2;
```



- NU_RUNSUM generates the sequence $\left(\left(\sum_{i=1}^{n} \nu_2(i)\right) \bmod 2\right)_{n\geq 1} = \left(\nu_2(n!) \bmod 2\right)_{n\geq 1}$.

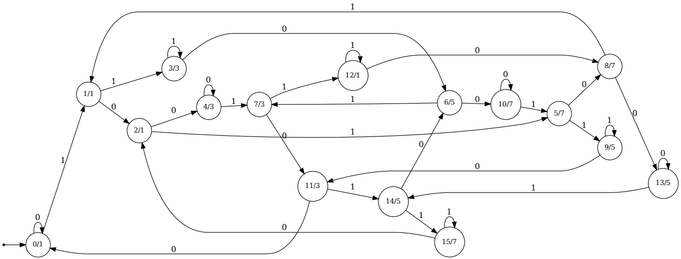
In Walnut

- Next, transduce `G_MOD8` with `RUNPROD1357` to get the DFAO `G_RUNPROD` with the Walnut command

```
transduce G_RUNPROD RUNPROD1357 G_MOD8;
```

- Next, transduce G_MOD8 with RUNPROD1357 to get the DFAO G_RUNPROD with the Walnut command

```
transduce G_RUNPROD RUNPROD1357 G_MOD8;
```



- G_RUNPROD is an 18-state DFAO that generates the sequence $\left(\left(\prod_{i=1}^{n} g(i)\right) \bmod 8\right)_{n \geq 1} = (g(n!) \bmod 8)_{n \geq 1}$.

In Walnut

- Lastly, we generate the final automaton that accepts S using the characterization that

$$n \in S \text{ iff } \nu_2(n!) \equiv 1 \pmod 2 \text{ or } g(n!) \not\equiv 7 \pmod 8,$$

which is directly translated into the Walnut command

```
def nfac_in_s "(NU_RUNSUM[i] = @1) | ~(G_RUNPROD[i] = @7)";
```

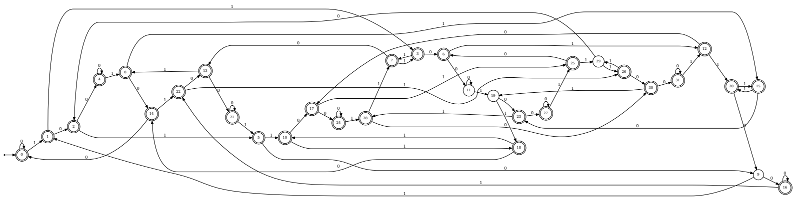
In Walnut

- Lastly, we generate the final automaton that accepts S using the characterization that

$$n \in S \text{ iff } \nu_2(n!) \equiv 1 \pmod{2} \text{ or } g(n!) \not\equiv 7 \pmod{8},$$

which is directly translated into the Walnut command

```
def nfac_in_s "(NU_RUNSUM[i] = @1) | ~(G_RUNPROD[i] = @7)";
```



- This 32-state DFA accepts $(n)_2$ if and only if $n!$ is a sum of three squares.

Complexity of Dekking's algorithm

- The resulting automaton after applying Dekking's transduction algorithm has an astronomical worst-case state complexity (before minimization) of $\leq |Q| \cdot |V|^{2 \cdot |V|^{|Q| \cdot |V|+1}}$, where Q and V are the number of states in the initial automaton and transducer, respectively.
- However, this complexity very rarely occurs in practice, with all computations taking at most a few seconds to run.

Transducing Fibonacci-automatic sequences

Transducing Fibonacci-automatic sequences

- So far, we have only been transducing k-automatic sequences, which are generated by DFAOs that read $(n)_k$.

Transducing Fibonacci-automatic sequences

- So far, we have only been transducing k-automatic sequences, which are generated by DFAOs that read $(n)_k$.
- We can define more general automatic sequences such as Fibonacci-automatic sequences, which read in Fibonacci representations $(n)_{\text{fib}}$.

Transducing Fibonacci-automatic sequences

- So far, we have only been transducing k-automatic sequences, which are generated by DFAOs that read $(n)_k$.
- We can define more general automatic sequences such as Fibonacci-automatic sequences, which read in Fibonacci representations $(n)_{\text{fib}}$.
- We showed that Fibonacci-automatic sequences can also be transduced using Dekking's algorithm.

Fibonacci representations

- Define $F_0 = 0$, $F_1 = 1$, and $F_n = F_{n-1} + F_{n-2}$ for $n \geq 2$.

Fibonacci representations

- Define $F_0 = 0$, $F_1 = 1$, and $F_n = F_{n-1} + F_{n-2}$ for $n \geq 2$.
- The *Fibonacci representation* of an integer $n \geq 0$ is $(n)_{\text{fib}} = d_t d_{t-1} \cdots d_1 d_0$, where $n = \sum_{i=0}^{t} d_i F_{i+2}$ and $d_i \in \{0, 1\}$ *with no two consecutive* $1s$.

Fibonacci representations

- Define $F_0 = 0$, $F_1 = 1$, and $F_n = F_{n-1} + F_{n-2}$ for $n \geq 2$.
- The *Fibonacci representation* of an integer $n \geq 0$ is $(n)_{\text{fib}} = d_t d_{t-1} \cdots d_1 d_0$, where $n = \sum_{i=0}^{t} d_i F_{i+2}$ and $d_i \in \{0, 1\}$ *with no two consecutive* 1*s*.
- Every integer can be uniquely written in this way.

Fibonacci representations

- Define $F_0 = 0$, $F_1 = 1$, and $F_n = F_{n-1} + F_{n-2}$ for $n \geq 2$.
- The *Fibonacci representation* of an integer $n \geq 0$ is $(n)_{\text{fib}} = d_t d_{t-1} \cdots d_1 d_0$, where $n = \sum_{i=0}^{t} d_i F_{i+2}$ and $d_i \in \{0, 1\}$ *with no two consecutive* 1*s*.
- Every integer can be uniquely written in this way.
- For example, $(14)_{\text{fib}} = 100001$, as $14 = 1 \cdot 13 + 0 \cdot 8 + 0 \cdot 5 + 0 \cdot 3 + 0 \cdot 2 + 1 \cdot 1$.

Fibonacci-Thue-Morse sequence

- Consider the *Fibonacci-Thue-Morse* sequence $(\mathbf{ftm}[n])_{n\geq 0}$, where $\mathbf{ftm}[n]$ is the sum (mod 2) of the digits of $(n)_{\mathrm{fib}}$:

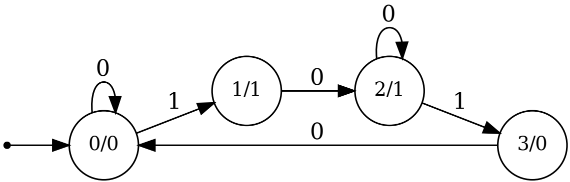
$$\mathbf{ftm} = 0111010010001100010 1\cdots$$

Fibonacci-Thue-Morse sequence

- Consider the *Fibonacci-Thue-Morse* sequence $(\mathbf{ftm}[n])_{n\geq 0}$, where $\mathbf{ftm}[n]$ is the sum (mod 2) of the digits of $(n)_{\mathrm{fib}}$:

$$\mathbf{ftm} = 0111010010001100010 1\cdots$$

- The sequence **ftm** is *Fibonacci-automatic*, i.e. there is a finite automaton M with output that computes $\mathbf{ftm}[n]$ on input the Fibonacci representation of n:

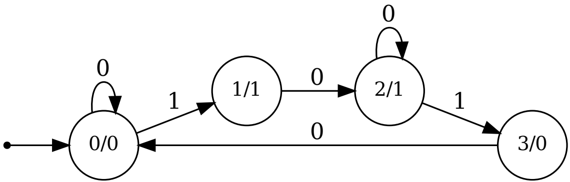


Fibonacci-Thue-Morse sequence

- Consider the *Fibonacci-Thue-Morse* sequence $(\mathbf{ftm}[n])_{n\geq 0}$, where $\mathbf{ftm}[n]$ is the sum (mod 2) of the digits of $(n)_{\text{fib}}$:

$$\mathbf{ftm} = 0111010010001100010 1\cdots$$

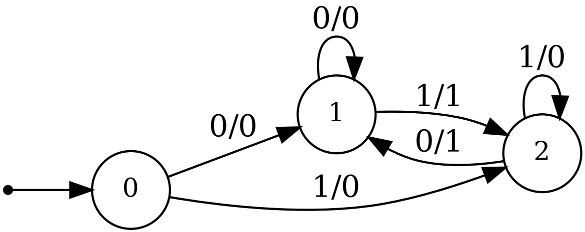
- The sequence **ftm** is *Fibonacci-automatic*, i.e. there is a finite automaton M with output that computes $\mathbf{ftm}[n]$ on input the Fibonacci representation of n:



- M is only defined on valid Fibonacci representations, i.e., binary strings with no consecutive 1s.

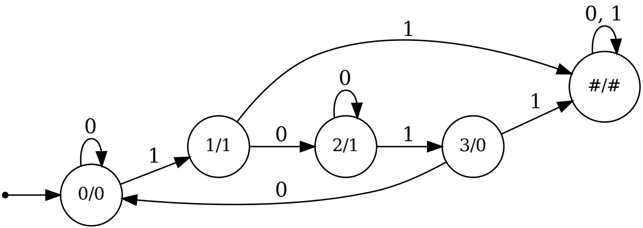
Transducing Fibonacci-Thue-Morse

- We will create an automaton that computes the running XOR of **ftm** using Dekking's transduction algorithm.
- Here is our XOR transducer, which computes the XOR of consecutive bits:



Transducing Fibonacci-Thue-Morse

- We add a "dead state" to M along with a special output $\#$, giving a new DFAO M':

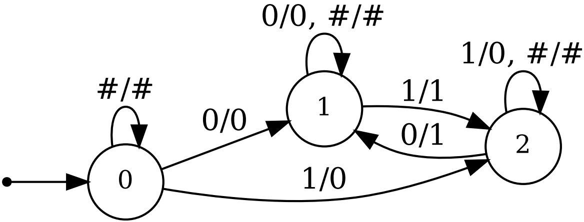


- The sequence computed by M' is

$$\mathbf{ftm}' = 011\#10\#\#100\#\#\#\#\#1\cdots,$$

which is now a 2-automatic sequence.

We similarly create an extension T' of T such that upon reading $\#$, it outputs $\#$ and not change state:

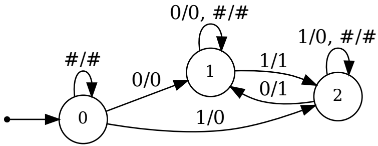


Transducing Fibonacci-Thue-Morse

We can now transduce $\mathbf{ftm}'$ with T':

$$\mathbf{ftm}' = 011\#10\#\#100\#\#\#\#\#1\cdots$$

$$\downarrow$$



$$\downarrow$$

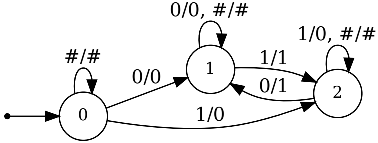
$$T'(\mathbf{ftm}') = 010\#01\#\#110\#\#\#\#\#1\cdots$$

Transducing Fibonacci-Thue-Morse

We can now transduce $\mathbf{ftm}'$ with T':

$$\mathbf{ftm}' = 011\#10\#\#100\#\#\#\#\#1\cdots$$

$$\downarrow$$



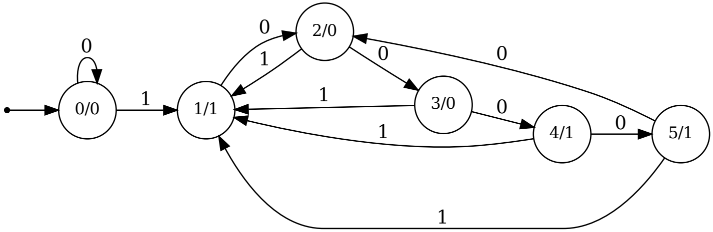
$$\downarrow$$

$$T'(\mathbf{ftm}') = 010\#01\#\#110\#\#\#\#\#1\cdots$$

Removing the $\#$s gives us our desired sequence $T(\mathbf{ftm}) = 010011101\cdots$.

`Walnut` automatically does all of this under the hood, so only one command is needed:

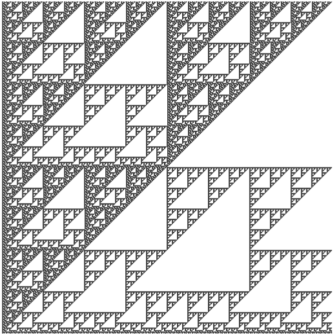
`transduce FTMXOR XOR FTM:`



This automaton computes $T(\mathbf{ftm})$, the running XOR of **ftm**.

Future Work

- More generalized transducers can be implemented into `Walnut`.
  - Schaeffer (2013) provides a transducer model that allows transduction of arbitrary factors of automatic sequences.
- Explicit characterization of automata computing iterated running sums of the Thue-Morse sequence. So far, we explicitly characterize the 2^n-fold running sums of Thue-Morse in our full paper.



The first 512 iterated running sums of the Thue-Morse sequence.

Thank you!