Automatic Sequences

Anatoly Zavyalov

University of Toronto

June 21, 2023

About me

- I am entering my fourth year as an undergraduate at the University of Toronto (St. George).
- I study math, computer science, and physics.
- My research interests are theoretical computer science (especially automata theory), and discrete math in general. Previously, I have also done research in astronomy.
- I also play piano and make video games for fun.

Photo Credit:
Anastasia Zhurikhina

Table of Contents

Gum

A gumball machine charges 25¢ for a gumball, and exact change is needed. The only types of coins you can choose from are 5¢, 10¢, and 25¢. If you put in more than 25¢, the gumball machine explodes.
In what ways can you get a gumball?

Gum

A gumball machine charges 25¢ for a gumball, and exact change is needed. The only types of coins you can choose from are 5¢, 10¢, and 25¢. If you put in more than 25¢, the gumball machine explodes.
In what ways can you get a gumball?

- 25¢

Gum

A gumball machine charges 25¢ for a gumball, and exact change is needed. The only types of coins you can choose from are 5¢, 10¢, and 25¢. If you put in more than 25¢, the gumball machine explodes.
In what ways can you get a gumball?

- 25¢
- 5¢ 5¢ 10¢ 5¢

Gum

A gumball machine charges 25¢ for a gumball, and exact change is needed. The only types of coins you can choose from are 5¢, 10¢, and 25¢. If you put in more than 25¢, the gumball machine explodes.
In what ways can you get a gumball?

- 25¢
- 5¢ 5¢ 10¢ 5¢
- 10¢ 5¢ 5¢ 5¢

Gum

A gumball machine charges 25¢ for a gumball, and exact change is needed. The only types of coins you can choose from are 5¢, 10¢, and 25¢. If you put in more than 25¢, the gumball machine explodes.
In what ways can you get a gumball?

- 25¢
- 5¢ 5¢ 10¢ 5¢
- 10¢ 5¢ 5¢ 5¢

But not:

- 5¢ 5¢

Gum

A gumball machine charges 25¢ for a gumball, and exact change is needed. The only types of coins you can choose from are 5¢, 10¢, and 25¢. If you put in more than 25¢, the gumball machine explodes.
In what ways can you get a gumball?

- 25¢
- 5¢ 5¢ 10¢ 5¢
- 10¢ 5¢ 5¢ 5¢

But not:

- 5¢ 5¢
- ε (empty string)

Gum

A gumball machine charges 25¢ for a gumball, and exact change is needed. The only types of coins you can choose from are 5¢, 10¢, and 25¢. If you put in more than 25¢, the gumball machine explodes.
In what ways can you get a gumball?

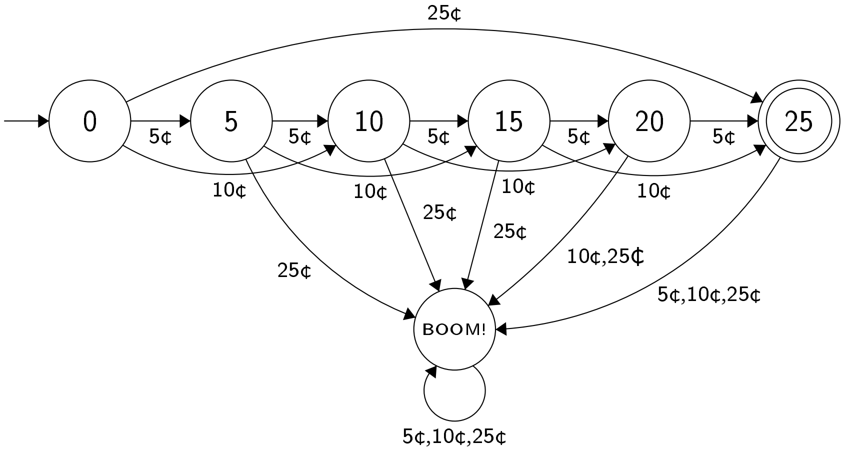
- 25¢
- 5¢ 5¢ 10¢ 5¢
- 10¢ 5¢ 5¢ 5¢

But not:

- 5¢ 5¢
- ε (empty string)
- 10¢ 25¢ (BOOM!)

Deterministic Finite Automaton

Here is a deterministic finite automaton (DFA) for the gumball machine:



The states tracks how much money has been paid so far. Once the 25 state is reached, the fare is accepted.

Deterministic Finite Automaton

Definition

A deterministic finite automaton (DFA) is a tuple $M = \langle Q, \Sigma, \delta, q_0, F \rangle$ where

- Q is a finite set of *states*
- Σ is the (finite) *input alphabet*
- $\delta : Q \times \Sigma \to Q$ is the *transition function*
- $q_0 \in Q$ is the *initial state*
- $F \subseteq Q$ are the *accepting/final states*

Deterministic Finite Automaton

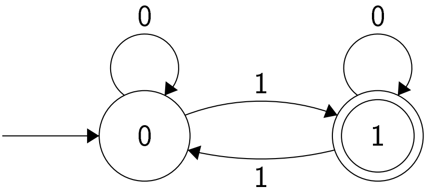
Definition

A deterministic finite automaton (DFA) is a tuple $M = \langle Q, \Sigma, \delta, q_0, F \rangle$ where

- Q is a finite set of *states*
- Σ is the (finite) *input alphabet*
- $\delta : Q \times \Sigma \to Q$ is the *transition function*
- $q_0 \in Q$ is the *initial state*
- $F \subseteq Q$ are the *accepting/final states*

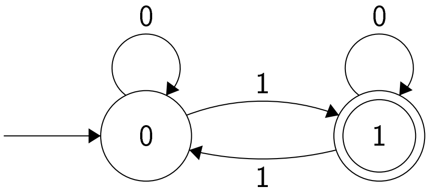
A DFA M accepts $x \in \Sigma^*$ if x ends at a state in F when passed through M.

DFA Example



What kinds of strings does this automaton accept?

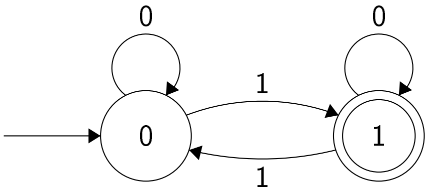
DFA Example



What kinds of strings does this automaton accept?

- 001

DFA Example

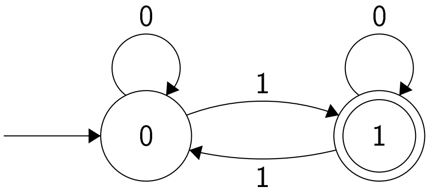


What kinds of strings does this automaton accept?

- 001
- 0100011

What strings will it reject?

DFA Example



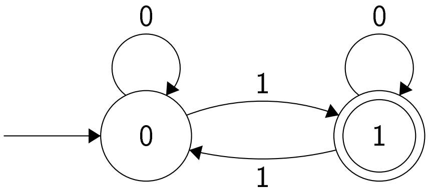
What kinds of strings does this automaton accept?

- 001
- 0100011

What strings will it reject?

- 1010

DFA Example



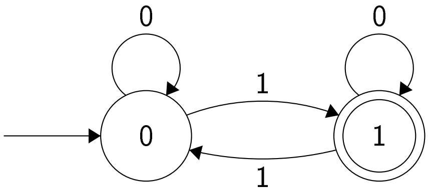
What kinds of strings does this automaton accept?

- 001
- 0100011

What strings will it reject?

- 1010
- 0000000

DFA Example



What kinds of strings does this automaton accept?

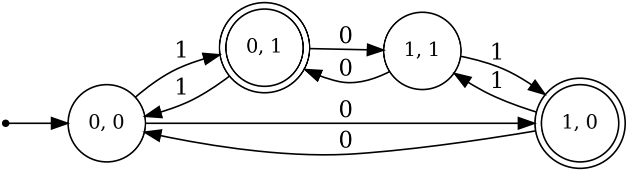
- 001
- 0100011

What strings will it reject?

- 1010
- 0000000

Accepts $x \in \{0,1\}^*$ if and only if x the parity of the number of 1 in x is odd, or equivalently if the sum of the digits of x is odd.

DFA as a computational model



- DFAs are a memoryless computational model: they only remember what state it is on!
- They are very simple, but can be used to solve surprisingly difficult problems.

Example: Sum of three squares

Legendre's three square theorem says that a number $n \in \mathbb{N}$ is a sum of three squares of integers

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

if and only if n is *not* of the form $n = 4^a(8b+7)$ for $a, b \in \mathbb{Z}_{\geq 0}$.

Example: Sum of three squares

Legendre's three square theorem says that a number $n \in \mathbb{N}$ is a sum of three squares of integers

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

if and only if n is *not* of the form $n = 4^a(8b+7)$ for $a, b \in \mathbb{Z}_{\geq 0}$.

We will make a DFA that reads in a binary representation of n and accepts if and only if n is a sum of three squares of integers.

Example: Sum of three squares

Suppose $n = 4^a(8b+7)$ for some $a, b \in \mathbb{Z}_{\geq 0}$. What can we say about the binary representation of n?

Example: Sum of three squares

Suppose $n = 4^a(8b+7)$ for some $a, b \in \mathbb{Z}_{\geq 0}$. What can we say about the binary representation of n?

If $b \in \mathbb{Z}_{\geq 0}$, then $(8b)_2$ looks like

$$\underbrace{\cdots}_{\in\{0,1\}^*} 000$$

Example: Sum of three squares

Suppose $n = 4^a(8b+7)$ for some $a, b \in \mathbb{Z}_{\geq 0}$. What can we say about the binary representation of n?

If $b \in \mathbb{Z}_{\geq 0}$, then $(8b)_2$ looks like

$$\underbrace{\cdots}_{\in\{0,1\}^*} 000$$

So $(8b+7)_2$ looks like

$$\underbrace{\cdots}_{\in\{0,1\}^*} 111$$

Example: Sum of three squares

Suppose $n = 4^a(8b+7)$ for some $a, b \in \mathbb{Z}_{\geq 0}$. What can we say about the binary representation of n?

If $b \in \mathbb{Z}_{\geq 0}$, then $(8b)_2$ looks like

$$\underbrace{\cdots}_{\in\{0,1\}^*} 000$$

So $(8b+7)_2$ looks like

$$\underbrace{\cdots}_{\in\{0,1\}^*} 111$$

Lastly, $(4^a(8b+7))_2$ looks like

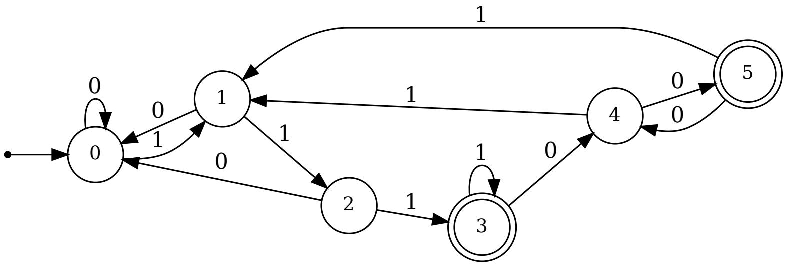
$$\underbrace{\cdots}_{\in\{0,1\}^*} 111 \underbrace{00\cdots00}_{\substack{\text{even \# of 0's,} \\ \text{may be } \varepsilon}}$$

Example: Sum of three squares

The automaton that accepts $(n)_2$ if and only if it is in the form

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

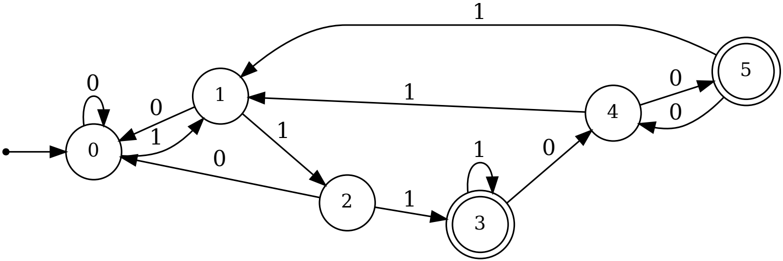
is:



The automaton that accepts $(n)_2$ if and only if it is in the form

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

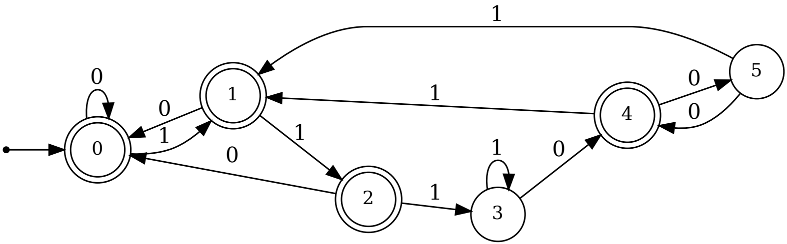
is:



So this automaton accepts $(n)_2$ if and only if n *is not* a sum of three squares.

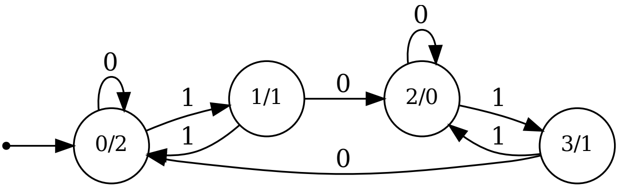
Example: Sum of three squares

To accept all $(n)_2$ if and only if n *is* a sum of three squares, just flip the final states:



Deterministic Finite Automaton with Output (DFAO)

Instead of final states, let's give our automaton an output on every state:



This is called a deterministic finite automaton *with output* (DFAO).

DFAO

Definition

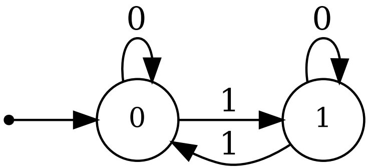
A deterministic finite automaton with output (DFAO) is a tuple $M = \langle Q, \Sigma, \delta, q_0, \Delta, \lambda \rangle$, where

- Q is a finite set of *states*
- Σ is the (finite) *input alphabet*
- $\delta \colon Q \times \Sigma \to Q$ is the *transition function*
- $q_0 \in Q$ is the *initial state*
- Δ is the (finite) *output alphabet*
- $\lambda \colon Q \to \Delta$ is the *coding* (*output function*)

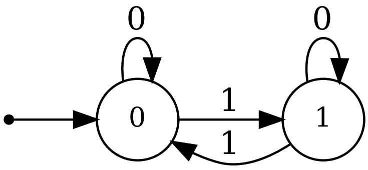
Automatic Sequences

Let's take a DFAO with transitions labelled by 0 and 1, and put numbers in base-2 into it.

n	$(n)_2$	$\mathbf{t}[n]$
0	0	

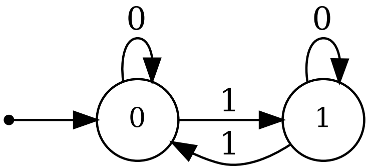


Let's take a DFAO with transitions labelled by 0 and 1, and put numbers in base-2 into it.



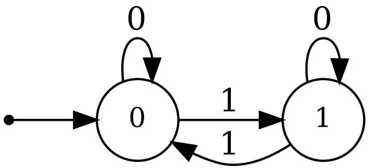
n	$(n)_2$	$\mathbf{t}[n]$
0	0	**0**
1	1	

Let's take a DFAO with transitions labelled by 0 and 1, and put numbers in base-2 into it.



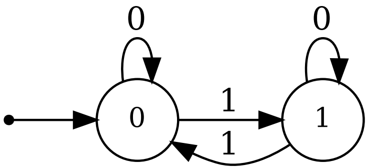
n	$(n)_2$	$\mathbf{t}[n]$
0	0	**0**
1	1	**1**
2	10	

Let's take a DFAO with transitions labelled by 0 and 1, and put numbers in base-2 into it.



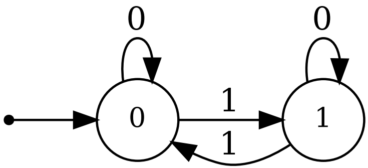
n	$(n)_2$	$\mathbf{t}[n]$
0	0	**0**
1	1	**1**
2	10	**1**
3	11	

Let's take a DFAO with transitions labelled by 0 and 1, and put numbers in base-2 into it.



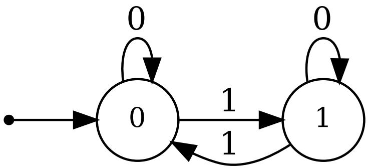
n	$(n)_2$	$\mathbf{t}[n]$
0	0	**0**
1	1	**1**
2	10	**1**
3	11	**0**
4	100	

Let's take a DFAO with transitions labelled by 0 and 1, and put numbers in base-2 into it.



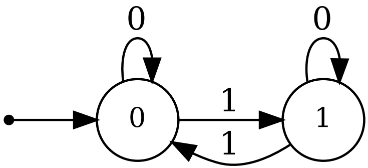
n	$(n)_2$	$\mathbf{t}[n]$
0	0	**0**
1	1	**1**
2	10	**1**
3	11	**0**
4	100	**1**
5	101	

Let's take a DFAO with transitions labelled by 0 and 1, and put numbers in base-2 into it.



n	$(n)_2$	$\mathbf{t}[n]$
0	0	**0**
1	1	**1**
2	10	**1**
3	11	**0**
4	100	**1**
5	101	**0**
6	110	

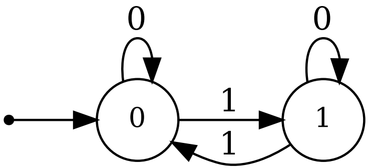
Let's take a DFAO with transitions labelled by 0 and 1, and put numbers in base-2 into it.



n	$(n)_2$	$\mathbf{t}[n]$
0	0	**0**
1	1	**1**
2	10	**1**
3	11	**0**
4	100	**1**
5	101	**0**
6	110	**0**
7	111	

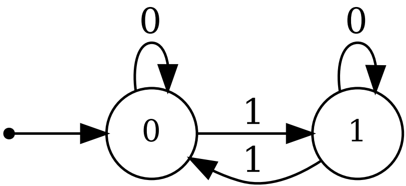
Automatic Sequences

Let's take a DFAO with transitions labelled by 0 and 1, and put numbers in base-2 into it.



n	$(n)_2$	$\mathbf{t}[n]$
0	0	**0**
1	1	**1**
2	10	**1**
3	11	**0**
4	100	**1**
5	101	**0**
6	110	**0**
7	111	**1**
⋮	⋮	⋮

Example: Thue-Morse sequence

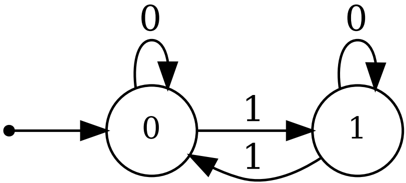


This automaton computes the Thue-Morse sequence

$$\mathbf{t} = 0110\,1001\,1001\,0110 \cdots,$$

where $\mathbf{t}[n]$ is the parity of the number of 1s in the binary representation of n, or equivalently the sum (mod 2) of the bits in $(n)_2$.

Example: Thue-Morse sequence



This automaton computes the Thue-Morse sequence

$$\mathbf{t} = 0110\,1001\,1001\,0110 \cdots,$$

where $\mathbf{t}[n]$ is the parity of the number of 1s in the binary representation of n, or equivalently the sum (mod 2) of the bits in $(n)_2$.

A sequence that can be computed by an automaton in this way is called automatic.

Automatic sequence

Definition

Let $M = \langle Q, \Sigma, \delta, q_0, \Delta, \lambda \rangle$ is a DFAO and suppose $\Sigma = \{0, \ldots, k-1\}$ for some $k \in \mathbb{N}$. The sequence $(x_n)_{n \geq 0}$ computed by M is defined by

$$x_n = \lambda(\delta(q_0, (n)_k)),$$

where $(n)_k$ denotes the most-significant-digit-first base-k representation of $n \in \mathbb{N}$, i.e. $(n)_k = d_t d_{t-1} \cdots d_1 d_0$ where $n = \sum_{i=0}^{t} d_i k^i$ and $d_i \in \{0, \ldots, k-1\}$ for all $i = 0, \ldots, t$.

A sequence $\mathbf{x} = (x_n)_{n \geq 0}$ is called k-automatic if there exists a DFAO M with input alphabet $\Sigma = \{0, \ldots, k-1\}$ that computes $\mathbf{x}$.

Infinite chess games?

The three-fold repetition rule in chess states that if the same position is reached three times, then the game is declared a draw.

With this rule, games cannot go on forever, as there are only a finite number of positions.

Infinite chess games?

The three-fold repetition rule in chess states that if the same position is reached three times, then the game is declared a draw.

With this rule, games cannot go on forever, as there are only a finite number of positions.

A former (German) official rule (up until 1929) was as follows: if the same *sequence of moves* is made *three times in a row*, then the game is declared a draw.

Can infinite games exist with this weakened rule?

The three-fold repetition rule in chess states that if the same position is reached three times, then the game is declared a draw.

With this rule, games cannot go on forever, as there are only a finite number of positions.

A former (German) official rule (up until 1929) was as follows: if the same *sequence of moves* is made *three times in a row*, then the game is declared a draw.

Can infinite games exist with this weakened rule?

Yes!

Infinite chess games!

Max Euwe, a Dutch mathematician and former chess world champion, showed that infinite chess games are possible under this rule using the Thue-Morse sequence!

Max Euwe (1901 - 1981)
Credit: Wikipedia

Infinite chess games!

The Thue-Morse sequence is cubefree: it contains no blocks of the form XXX.

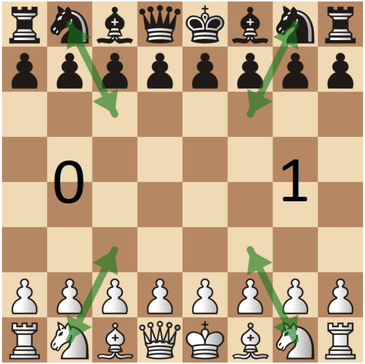
For example,

$$011010011\,\mathbf{001}\,0110\cdots$$

"001001001" will never appear in the Thue-Morse sequence.

We use this property of the Thue-Morse sequence to construct our infinite game.

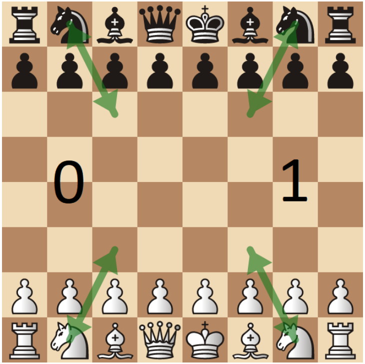
Infinite chess games!



$0 \mapsto$ Nc3 Nc6, Nb1 Nb8

$1 \mapsto$ Nf3 Nf6, Ng1 Ng8

Infinite chess games!



$0 \mapsto$ Nc3 Nc6, Nb1 Nb8

$1 \mapsto$ Nf3 Nf6, Ng1 Ng8

Apply these moves in the order of the Thue-Morse sequence:

$$0110100110010110\cdots$$

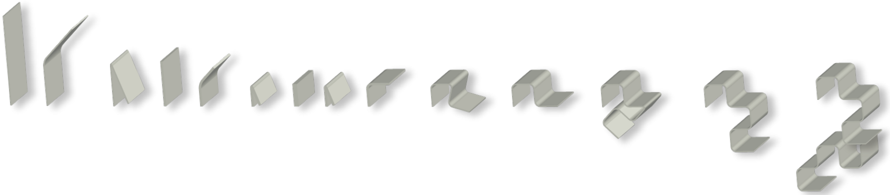
Because the Thue-Morse sequence is cubefree, the same sequence of moves will never be made three times in a row!

Paperfolding Sequence

Take a piece of paper and keep folding it in the same direction, then unfold it.

Paperfolding Sequence

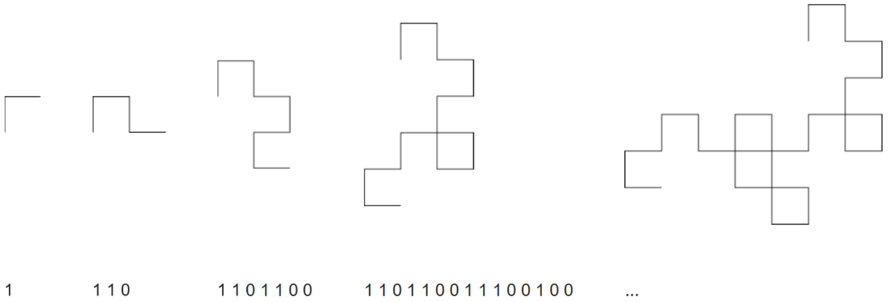
Take a piece of paper and keep folding it in the same direction, then unfold it.



Credit: (French) Wikipedia

Paperfolding Sequence

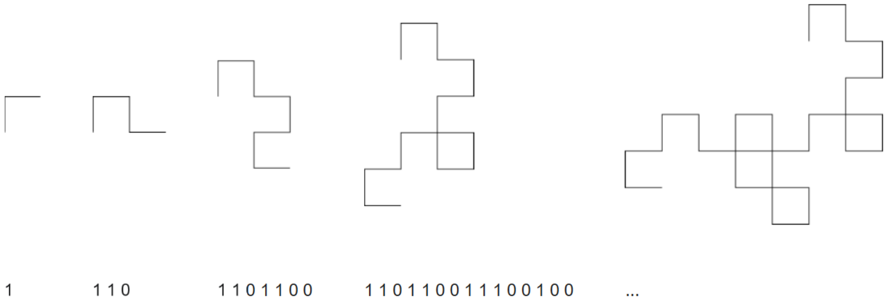
Call every left turn a 0, and every right turn a 1.



Credit: Wikipedia

Paperfolding Sequence

Call every left turn a 0, and every right turn a 1.

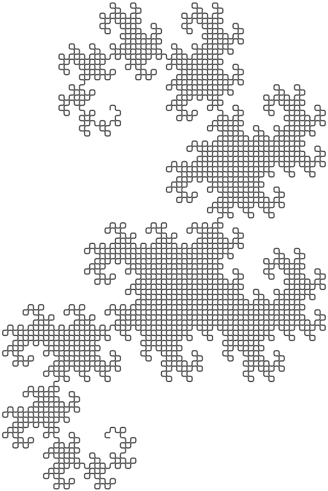


Credit: Wikipedia

Extending this to infinity, we get the paperfolding sequence (also called the dragon curve sequence):

$$1101100111001001110110001100100 11\cdots$$

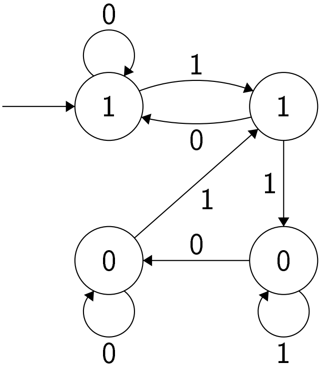
Paperfolding Sequence



After 12 folds. Credit: Allouche & Shallit

Paperfolding Sequence

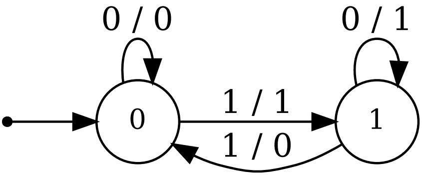
The paperfolding sequence is automatic, computed by this automaton:



To determine whether the k'th fold is a left or right turn, just feed $(k)_2$ into this automaton and look at the output!

Transducers

What if instead of putting the outputs on the states, we put them on the edges?



This is a transducer.

As we input a string into a transducer, we write down the outputs of the edges we pass through.

Transducers

Definition

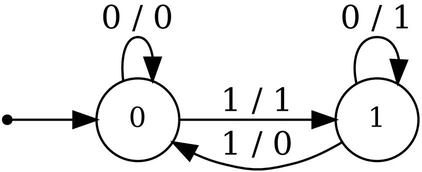
A transducer is a tuple

$$T = \langle V, \Delta, \varphi, v_0, \Gamma, \sigma \rangle,$$

where

- V is a finite set of *states*
- Δ is the finite *input alphabet*
- $\varphi \colon V \times \Delta \to V$ is the *transition function*
- $v_0 \in V$ is the *initial state*
- Γ is the finite *output alphabet*
- $\sigma \colon V \times \Delta \to \Gamma$ is the *output function*

Example: Running sum transducer



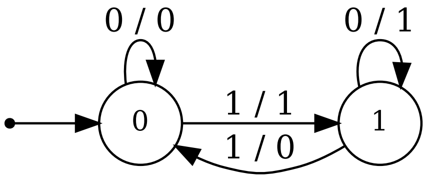
This transducer outputs the running sum mod 2 of the input.

Example: Running sum of Thue-Morse

Thue-Morse sequence:

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

$$\downarrow$$



$$\downarrow$$

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

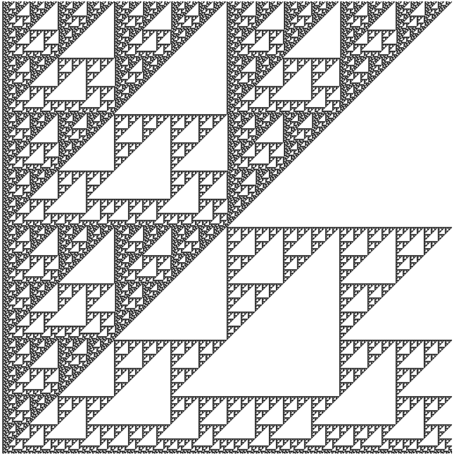
Example: Running sum of Thue-Morse

Continue taking running sums,

$$\begin{aligned} \mathbf{t} &= 0110\,1001\,1001\,0110\cdots \\ T(\mathbf{t}) &= 0100\,1110\,1110\,0100\cdots \\ T^2(\mathbf{t}) &= 0111\,0100\,1011\,1000\cdots \\ T^3(\mathbf{t}) &= 0101\,1000\,1101\,0000\cdots \\ T^4(\mathbf{t}) &= 0110\,1000\,1001\,0000 \\ &\vdots \end{aligned}$$

Example: Running sum of Thue-Morse

If we plot each running sum on a separate row, we get an awesome Sierpinski-like fractal:



Summary

- Automatic sequences are a class of sequences that are computed by finite automata.

Summary

- Automatic sequences are a class of sequences that are computed by finite automata.
- A lot of seemingly difficult problems become surprisingly simple after viewing them through the lens of automata theory.

Summary

- Automatic sequences are a class of sequences that are computed by finite automata.
- A lot of seemingly difficult problems become surprisingly simple after viewing them through the lens of automata theory.

Further reading:

- **For automatic sequences:** "Automatic Sequences: Theory, Applications, Generalizations" by Jean-Paul Allouche and Jeffrey Shallit

Summary

- Automatic sequences are a class of sequences that are computed by finite automata.
- A lot of seemingly difficult problems become surprisingly simple after viewing them through the lens of automata theory.

Further reading:

- **For automatic sequences:** "Automatic Sequences: Theory, Applications, Generalizations" by Jean-Paul Allouche and Jeffrey Shallit
- **For transducers:** Jeffrey Shallit, Anatoly Zavyalov. "Transduction of Automatic Sequences and Applications" (https://arxiv.org/abs/2303.15203)

Thank you!