Automatic Sequences

Anatoly Zavyalov

University of Toronto

January 19, 2023

About me

- I am a third-year undergraduate student at the University of Toronto (St. George).
- I study math, computer science, and physics.
- I have been doing research in automata theory since summer of 2022, and have previously done research in astronomy.
- I also play piano and make video games for fun.

Photo Credit:
Anastasia Zhurikhina

Table of Contents

Getting on the Bus

Bus fare costs 25¢, and exact change is needed. The only types of coins you can choose from are 5¢, 10¢, and 25¢. In what ways can you pay the fare?

Getting on the Bus

Bus fare costs 25¢, and exact change is needed. The only types of coins you can choose from are 5¢, 10¢, and 25¢. In what ways can you pay the fare?

- 25¢

Getting on the Bus

Bus fare costs 25¢, and exact change is needed. The only types of coins you can choose from are 5¢, 10¢, and 25¢. In what ways can you pay the fare?

- 25¢
- 5¢ 5¢ 10¢ 5¢

Bus fare costs 25¢, and exact change is needed. The only types of coins you can choose from are 5¢, 10¢, and 25¢. In what ways can you pay the fare?

- 25¢
- 5¢ 5¢ 10¢ 5¢
- 10¢ 5¢ 5¢ 5¢

Getting on the Bus

Bus fare costs 25¢, and exact change is needed. The only types of coins you can choose from are 5¢, 10¢, and 25¢. In what ways can you pay the fare?

- 25¢
- 5¢ 5¢ 10¢ 5¢
- 10¢ 5¢ 5¢ 5¢

But not:

- 5¢ 5¢

Getting on the Bus

Bus fare costs 25¢, and exact change is needed. The only types of coins you can choose from are 5¢, 10¢, and 25¢. In what ways can you pay the fare?

- 25¢
- 5¢ 5¢ 10¢ 5¢
- 10¢ 5¢ 5¢ 5¢

But not:

- 5¢ 5¢
- ε

Getting on the Bus

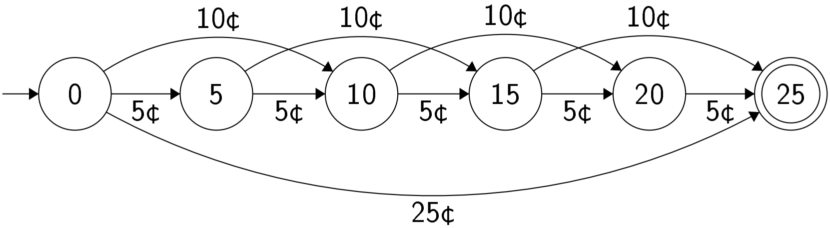
Bus fare costs 25¢, and exact change is needed. The only types of coins you can choose from are 5¢, 10¢, and 25¢. In what ways can you pay the fare?

- 25¢
- 5¢ 5¢ 10¢ 5¢
- 10¢ 5¢ 5¢ 5¢

But not:

- 5¢ 5¢
- ε
- 10¢ 25¢

State machine



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

Formal Languages

Let Σ be a finite nonempty set called an alphabet.

Σ^* denotes the set of all finite words over Σ.

For example, if $\Sigma = \{0,1\}$, then

$$\Sigma^* = \{\varepsilon, 0, 1, 00, 01, 10, 11, 000, 001, \ldots\},$$

where ε is the empty word.

If $x \in \Sigma^*$ is a word, $|x|$ denotes the length of x.

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 states*

Deterministic Finite Automaton

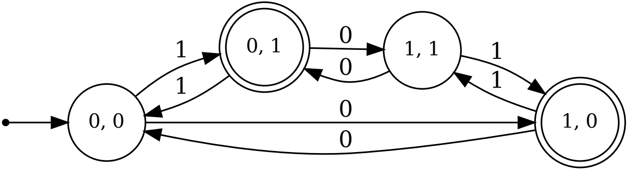
Definition

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 states*

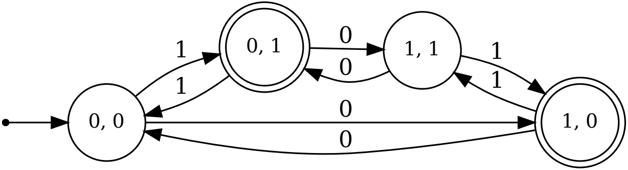
A DFA M accepts $x \in \Sigma^*$ if x ends at a state in F when passed through M.

DFA Example



What kinds of strings does this automaton accept?

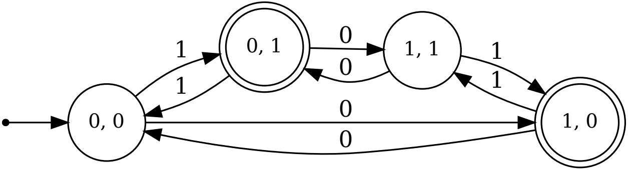
DFA Example



What kinds of strings does this automaton accept?

- 000

DFA Example

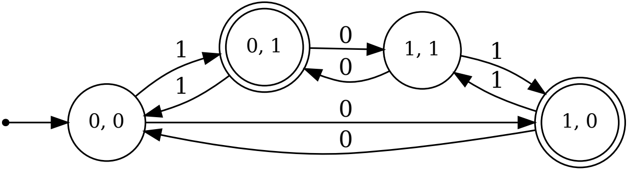


What kinds of strings does this automaton accept?

- 000
- 0100011

What strings will it reject?

DFA Example



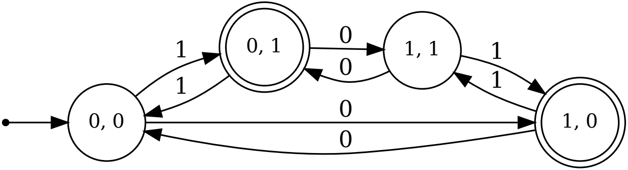
What kinds of strings does this automaton accept?

- 000
- 0100011

What strings will it reject?

- 1010

DFA Example



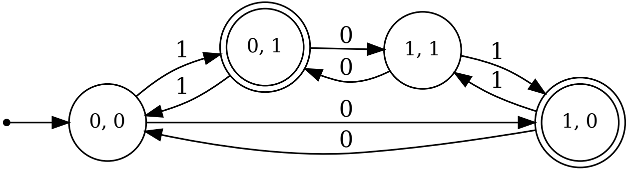
What kinds of strings does this automaton accept?

- 000
- 0100011

What strings will it reject?

- 1010
- 000001

DFA Example



What kinds of strings does this automaton accept?

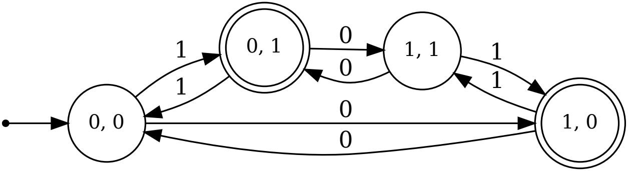
- 000
- 0100011

What strings will it reject?

- 1010
- 000001

Accepts $x \in \{0,1\}^*$ if and only if x the parity of the number of 0s in x is different from the parity of the number of 1s in x.

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

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

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

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

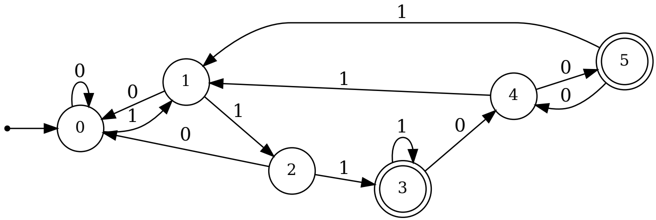
Lastly, $(4^a(8b+7))_2$ looks like

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

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:

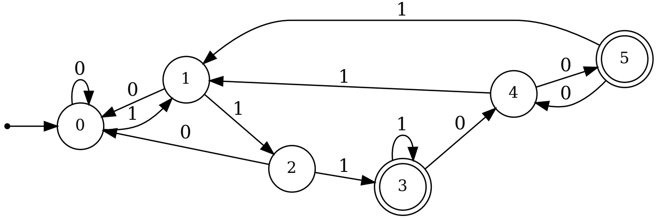


Example: Sum of three squares

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

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

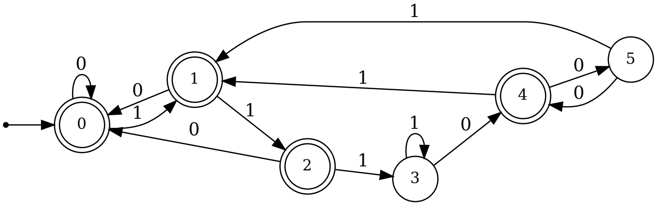
is:



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:



DFAO

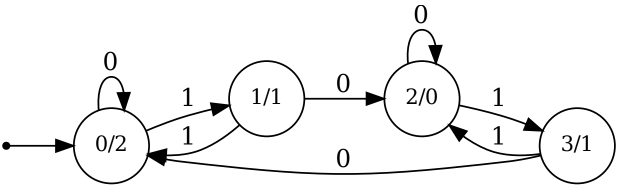
Definition

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*)

Example of a DFAO

Instead of final states, DFAOs have an output for every state:



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

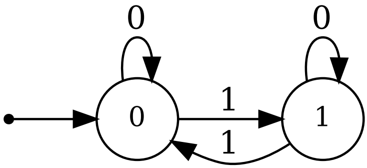
$$x_n = \lambda(\delta(q_0, (n)_k)),$$

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 $i = 0, \ldots, t$.

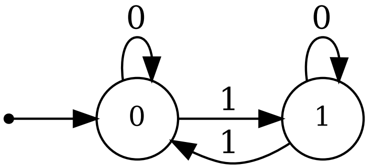
A sequence $\mathbf{x} = (x_n)_{n \geq 0}$ is called k-automatic if there exists a DFAO M with input alphabet $\Sigma = \{0, \ldots, k-1\}$ that computes $\mathbf{x}$.

Example: Thue-Morse sequence

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

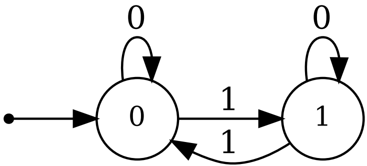


Example: Thue-Morse sequence



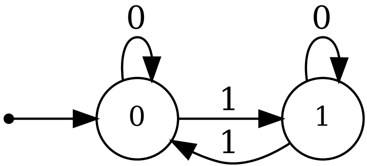
n	$(n)_2$	$\mathbf{t}[n]$
0	0	0
1	1	

Example: Thue-Morse sequence



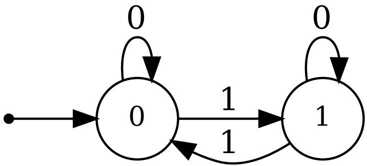
n	$(n)_2$	$\mathbf{t}[n]$
0	0	0
1	1	1
2	10	

Example: Thue-Morse sequence



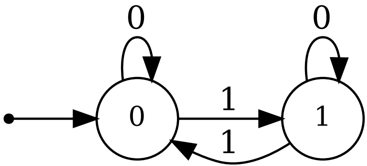
n	$(n)_2$	$\mathbf{t}[n]$
0	0	0
1	1	1
2	10	1
3	11	

Example: Thue-Morse sequence



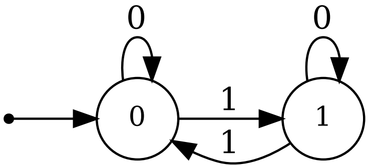
n	$(n)_2$	$\mathbf{t}[n]$
0	0	0
1	1	1
2	10	1
3	11	0
4	100	

Example: Thue-Morse sequence



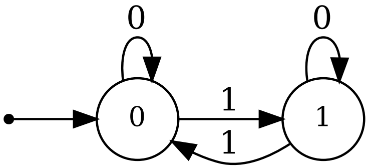
n	$(n)_2$	$\mathbf{t}[n]$
0	0	0
1	1	1
2	10	1
3	11	0
4	100	1
5	101	

Example: Thue-Morse sequence



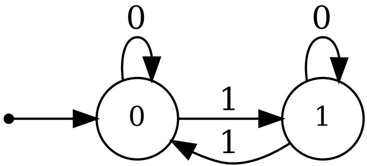
n	$(n)_2$	$\mathbf{t}[n]$
0	0	0
1	1	1
2	10	1
3	11	0
4	100	1
5	101	0
6	110	

Example: Thue-Morse sequence

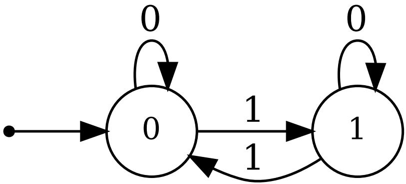


n	$(n)_2$	$\mathbf{t}[n]$
0	0	0
1	1	1
2	10	1
3	11	0
4	100	1
5	101	0
6	110	0
7	111	

Example: Thue-Morse sequence



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
⋮	⋮	⋮



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 $(n)_2$.

Fair sharing

Alice and Bob are dividing things of non-increasing value amongst themselves. What is the fairest order for them to pick?

Fair sharing

Alice and Bob are dividing things of non-increasing value amongst themselves. What is the fairest order for them to pick?

Suppose Alice picks first. Then Bob should pick second, then Alice picks third, Bob picks fourth, etc:

$$ABABABABAB\cdots$$

Fair sharing

Alice and Bob are dividing things of non-increasing value amongst themselves. What is the fairest order for them to pick?

Suppose Alice picks first. Then Bob should pick second, then Alice picks third, Bob picks fourth, etc:

$$ABABABABAB \cdots$$

Alice gets an advantage: For every pair of items, Alice will get to pick the better one!

Fair sharing

Maybe after AB, what if they swapped order after?

$$AB\,BA$$

Fair sharing

Maybe after AB, what if they swapped order after?

$$AB\,BA$$

Now it's more fair if there are 4 items, but if we repeat this:

$$ABBA\,ABBA\,ABBA\,ABBA\cdots$$

Fair sharing

Maybe after AB, what if they swapped order after?

$$AB\,BA$$

Now it's more fair if there are 4 items, but if we repeat this:

$$ABBA\,ABBA\,ABBA\,ABBA\cdots$$

Alice gets an advantage again: Alice will get to pick the best item out of every 4 items!

Fair sharing

Let's flip the order again:

$$ABBA\,BAAB$$

Fair sharing

Let's flip the order again:

$$ABBA\,BAAB$$

Again, if we repeat this, Alice will get to pick the best item out of every 8 items!

Fair sharing

Let's flip the order again:

$$ABBA\,BAAB$$

Again, if we repeat this, Alice will get to pick the best item out of every 8 items!

If we keep flipping the order,

$$ABBA\,BAAB\,BAAB\,ABBA\,BAAB\,ABBA\,ABBA\,BAAB\cdots$$

Fair sharing

Let's flip the order again:

$$ABBA\,BAAB$$

Again, if we repeat this, Alice will get to pick the best item out of every 8 items!

If we keep flipping the order,

$$ABBA\,BAAB\,BAAB\,ABBA\,BAAB\,ABBA\,ABBA\,BAAB\cdots$$

If we replace $A \to 0$ and $B \to 1$, this is the Thue-Morse sequence!

Fair sharing

A

$\downarrow$

$A\,\mathbf{B}$

Fair sharing

$$A$$

$$\downarrow$$

$$A\,\mathbf{B}$$

$$\downarrow$$

$$AB\,\mathbf{BA}$$

Fair sharing

A

$\downarrow$

A **B**

$\downarrow$

AB **BA**

$\downarrow$

$ABBA$ **BAAB**

Fair sharing

A

$\downarrow$

$A\,\mathbf{B}$

$\downarrow$

$AB\,\mathbf{BA}$

$\downarrow$

$ABBA\,\mathbf{BAAB}$

$\downarrow$

$ABBA\,BAAB\,\mathbf{BAAB\,ABBA}$

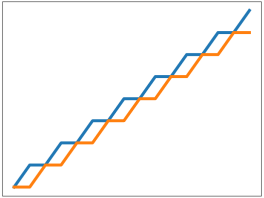
$\vdots$

$ABBA\,BAAB\,BAAB\,ABBA\,BAAB\,ABBA\,ABBA\,BAAB\cdots$

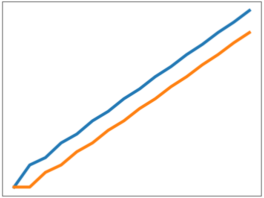
This is an equivalent definition of the Thue-Morse sequence.

Is it really more fair?

If the value of the items is constant,



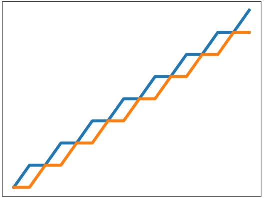
(a) $ABABABABAB\cdots$



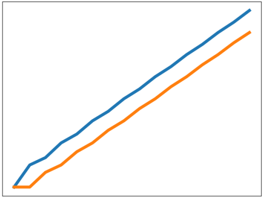
(b) Running average

Is it really more fair?

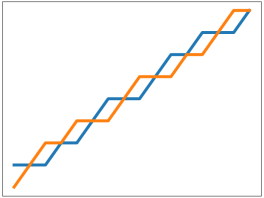
If the value of the items is constant,



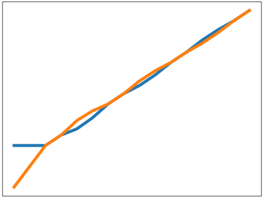
(a) $ABABABABAB\cdots$



(b) Running average



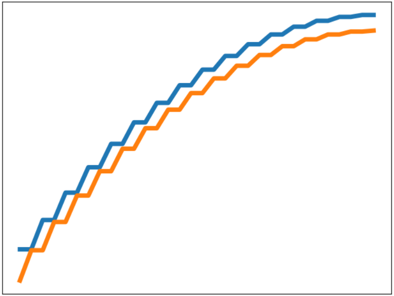
(c) Thue-Morse



(d) Thue-Morse average

Is it really more fair?

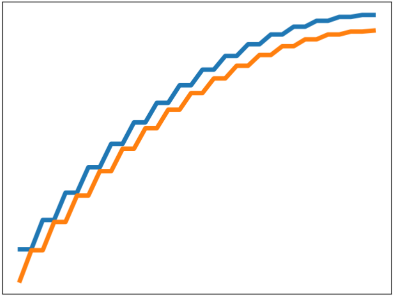
If the value of the items is decreasing,



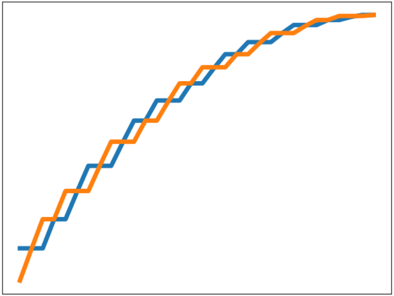
(a) $ABABABABAB\cdots$

Is it really more fair?

If the value of the items is decreasing,



(a) $ABABABABAB\cdots$



(b) Thue-Morse

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!

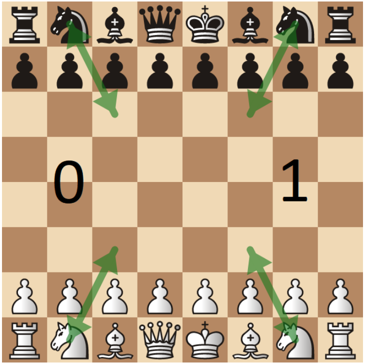
The Thue-Morse sequence is cubefree: it contains no blocks of the form XXX.

For example,

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

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

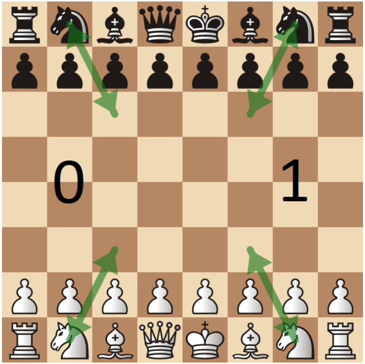
We use this property of the Thue-Morse sequence to construct our infinite game.



$0 \mapsto$ 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

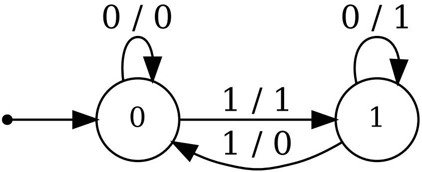
Apply these moves in the order of the Thue-Morse sequence:

$$0110100110010110\cdots$$

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

Transducers

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



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

Transducers

Definition

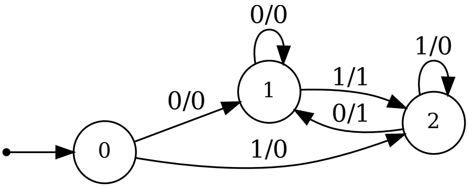
A transducer is a tuple

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

where

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

Example: XOR of Thue-Morse

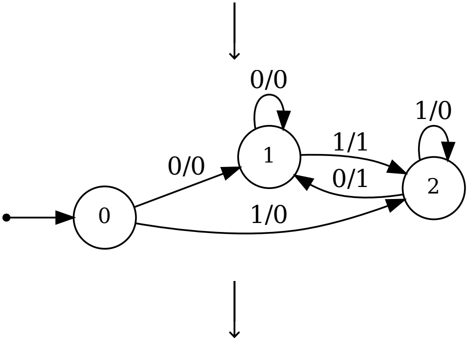


This transducer computes the XOR of consecutive bits (with the first bit outputted always being 0).

Example: XOR of Thue-Morse

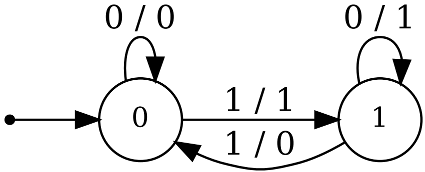
Thue-Morse sequence:

$$\mathbf{t} = 0110100110010110100101 10\cdots$$



$$T(\mathbf{t}) = 01011101010111011101 1101\cdots$$

Example: Running sum transducer



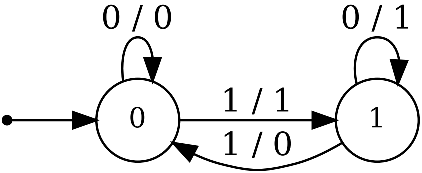
This transducer outputs the running sum mod 2 of the input.

Example: Running sum of Thue-Morse

Thue-Morse sequence:

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

$$\downarrow$$



$$\downarrow$$

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

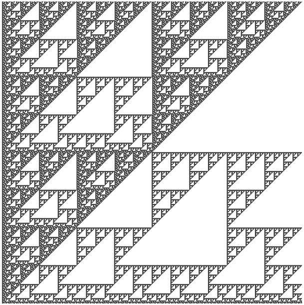
Example: Running sum of Thue-Morse

Continue taking running sums,

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

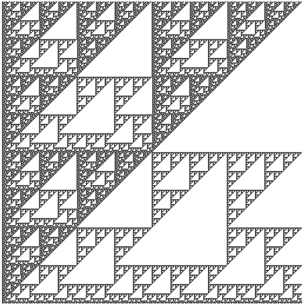
Example: Running sum of Thue-Morse

If we plot each running sum $T^k(\mathbf{t})$ on a separate row, we get a Sierpinski-like fractal:



Example: Running sum of Thue-Morse

If we plot each running sum $T^k(\mathbf{t})$ on a separate row, we get a Sierpinski-like fractal:



How can we characterize each row? Can we get a nice expression for $T^k(\mathbf{t})$ for arbitrary k? Right now, we only know expressions for $k = 2^n$.

Beyond base-k

Up until now, we've only considered automata that compute an automatic sequence when taking as input nmbers in base-k:

$$(n)_k = d_t d_{t-1} \cdots d_1 d_0 \text{ where } n = \sum_{i=0}^{t} d_i k^i,$$

and $d_i \in \{0, \ldots, k-1\}$ for all $i = 0, \ldots, t$.

Beyond base-k

Up until now, we've only considered automata that compute an automatic sequence when taking as input nmbers in base-k:

$$(n)_k = d_t d_{t-1} \cdots d_1 d_0 \text{ where } n = \sum_{i=0}^{t} d_i k^i,$$

and $d_i \in \{0, \ldots, k-1\}$ for all $i = 0, \ldots, t$.

Instead of writing numbers as sums of powers of k, we could write them in different numeration systems, e.g. Fibonacci!

Beyond base-k

The Fibonacci numbers are defined by the recurrence $F_n = F_{n-1} + F_{n-2}$, where $F_0 = 1,\ F_1 = 2$.

You can write any number $n \in \mathbb{N}$ as a sum of Fibonacci numbers:

$$(n)_{\text{fib}} = d_t d_{t-1} \cdots d_1 d_0 \text{ where } n = \sum_{i=0}^{t} d_i F_i$$

and $d_i \in \{0, 1\}$ for all $i = 0, \ldots, t$.

Beyond base-k

The Fibonacci numbers are defined by the recurrence $F_n = F_{n-1} + F_{n-2}$, where $F_0 = 1,\ F_1 = 2$.

You can write any number $n \in \mathbb{N}$ as a sum of Fibonacci numbers:

$$(n)_{\text{fib}} = d_t d_{t-1} \cdots d_1 d_0 \text{ where } n = \sum_{i=0}^{t} d_i F_i$$

and $d_i \in \{0, 1\}$ for all $i = 0, \ldots, t$.
However, this decomposition is not unique! For instance,

$$14 = 13 + 1 = 8 + 5 + 1 = 8 + 3 + 2 + 1$$

To make representations unique, we require that no two consecutive Fibonacci numbers be used in the sum, i.e.

$$(14)_{\text{fib}} = 100001, \text{ but not } 11001.$$

For example,

$$101 \to 1 \cdot 3 + 0 \cdot 2 + 1 \cdot 1 = 4$$

and

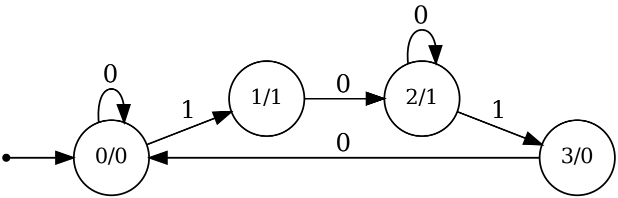
$$100101 \to 13 + 3 + 1 = 17$$

are valid Fibonacci representations, but 1101 and 1001100 are not.

So $x \in \{0,1\}^*$ is a valid Fibonacci representation if and only if x contains no consecutive 1s.

Fibonacci Thue-Morse

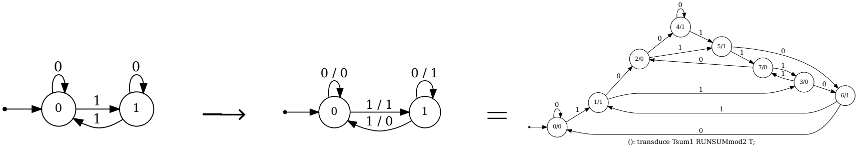
The Fibonacci Thue-Morse sequence **ftm** is the sum (mod 2) of the Fibonacci representation of n. The automaton that computes it is:



$$\mathbf{ftm} = 0111010010001100101\cdots$$

ftm is *Fibonacci*-automatic, but not k-automatic for any k. Notice that the above automaton is only defined on valid Fibonacci representations.

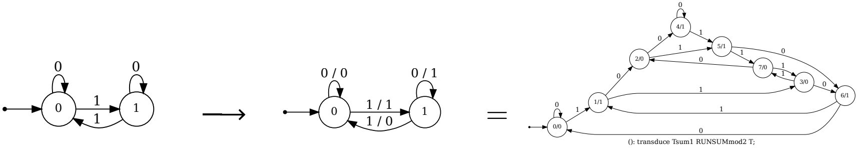
The transduction of a k-automatic sequence is still automatic:



Automaton $\longrightarrow$ Transducer $=$ Automaton

But only for k-automatic sequences! Can we apply transducers to Fibonacci-automatic sequences and get another Fibonacci automaton?

The transduction of a k-automatic sequence is still automatic:



Automaton ⟶ Transducer = Automaton

But only for k-automatic sequences! Can we apply transducers to Fibonacci-automatic sequences and get another Fibonacci automaton? **I proved that we can!** (Still unpublished)

Further Work

- `Walnut` is a software written by Hamoon Mousavi that answers questions posed in first-order logic about automatic sequences; it shortens long proofs by cases to writing and running a few commands.

Further Work

- `Walnut` is a software written by Hamoon Mousavi that answers questions posed in first-order logic about automatic sequences; it shortens long proofs by cases to writing and running a few commands.
- Transducers have only recently been added to `Walnut`, and new applications for them are constantly being found.

Further Work

- `Walnut` is a software written by Hamoon Mousavi that answers questions posed in first-order logic about automatic sequences; it shortens long proofs by cases to writing and running a few commands.
- Transducers have only recently been added to `Walnut`, and new applications for them are constantly being found.
- Applying transducers to sequences that are not over base-k has only recently been considered, and is still mostly unexplored.

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.
- Use the Thue-Morse sequence to share things fairly with your friends!

Acknowledgements

Professor Jeffrey O. Shallit
School of Computer Science
University of Waterloo