Automata Theory: The Foundations of Computer Science

Anatoly Zavyalov

University of Toronto

August 14, 2023

About me

- I am entering my fourth year as an undergraduate at the University of Toronto, studying 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.

Photo Credit: Anastasia Zhurikhina

P vs. NP

Who wants to win $\$1,000,000$?

P vs. NP

Who wants to win $1,000,000?

- The **P** vs. **NP** problem is a major unsolved problem in computer science.

P vs. NP

Who wants to win $\$1,000,000$?

- The **P** vs. **NP** problem is a major unsolved problem in computer science.
- It (roughly) asks whether every problem whose solution can be efficiently checked can also be efficiently solved.

Who wants to win $1,000,000?

- The **P** vs. **NP** problem is a major unsolved problem in computer science.
- It (roughly) asks whether every problem whose solution can be efficiently checked can also be efficiently solved.
  - A solution to a Sudoku puzzle can be quickly *checked*, but does that mean you can quickly *solve* a Sudoku puzzle?

P vs. NP

Who wants to win $\$1,000,000$?

- The **P** vs. **NP** problem is a major unsolved problem in computer science.
- It (roughly) asks whether every problem whose solution can be efficiently checked can also be efficiently solved.
  - A solution to a Sudoku puzzle can be quickly *checked*, but does that mean you can quickly *solve* a Sudoku puzzle?
  - No one knows.

Who wants to win $1,000,000?

- The **P** vs. **NP** problem is a major unsolved problem in computer science.
- It (roughly) asks whether every problem whose solution can be efficiently checked can also be efficiently solved.
  - A solution to a Sudoku puzzle can be quickly *checked*, but does that mean you can quickly *solve* a Sudoku puzzle?
  - No one knows.
- It is one of the Millenium Prize Problems, which carry a $1,000,000 prize for the first solution.

P vs. NP

Who wants to win $\$1,000,000$?

- The **P** vs. **NP** problem is a major unsolved problem in computer science.
- It (roughly) asks whether every problem whose solution can be efficiently checked can also be efficiently solved.
  - A solution to a Sudoku puzzle can be quickly *checked*, but does that mean you can quickly *solve* a Sudoku puzzle?
  - No one knows.
- It is one of the Millenium Prize Problems, which carry a $\$1,000,000$ prize for the first solution.
- We'll talk about a key component of understanding the problem: automata theory.

What is automata theory?

- Automata theory is a foundational branch of theoretical computer science that allows to abstractly represent computational models.

What is automata theory?

- Automata theory is a foundational branch of theoretical computer science that allows to abstractly represent computational models.
- Automata theory appears everywhere in computer science and related fields, including but not limited to:
    - Analysis of algorithms

What is automata theory?

- Automata theory is a foundational branch of theoretical computer science that allows to abstractly represent computational models.
- Automata theory appears everywhere in computer science and related fields, including but not limited to:
  - Analysis of algorithms
  - Design of programming languages, compilers, interpreters

What is automata theory?

- Automata theory is a foundational branch of theoretical computer science that allows to abstractly represent computational models.
- Automata theory appears everywhere in computer science and related fields, including but not limited to:
  - Analysis of algorithms
  - Design of programming languages, compilers, interpreters
  - Artificial intelligence

What is automata theory?

- Automata theory is a foundational branch of theoretical computer science that allows to abstractly represent computational models.
- Automata theory appears everywhere in computer science and related fields, including but not limited to:
    - Analysis of algorithms
    - Design of programming languages, compilers, interpreters
    - Artificial intelligence
    - Computational linguistics

Roadmap

- **Finite automaton**: the simplest computational model

Roadmap

- **Finite automaton**: the simplest computational model
- **Applications of finite automata**: parsing, number theory

Roadmap

- **Finite automaton**: the simplest computational model
- **Applications of finite automata**: parsing, number theory
- **Turing machine**: how we abstractly represent computers

Roadmap

- **Finite automaton**: the simplest computational model
- **Applications of finite automata**: parsing, number theory
- **Turing machine**: how we abstractly represent computers
- **Turing completeness**: systems that are as powerful as computers

A dangerous gumball machine

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 without the gumball machine exploding?

A dangerous gumball machine

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 without the gumball machine exploding?

- 25¢

A dangerous gumball machine

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 without the gumball machine exploding?

- 25¢
- 5¢ 5¢ 10¢ 5¢

A dangerous gumball machine

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 without the gumball machine exploding?

- 25¢
- 5¢ 5¢ 10¢ 5¢
- 10¢ 5¢ 5¢ 5¢

A dangerous gumball machine

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 without the gumball machine exploding?

- 25¢
- 5¢ 5¢ 10¢ 5¢
- 10¢ 5¢ 5¢ 5¢

But not:

- 5¢ 5¢

A dangerous gumball machine

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 without the gumball machine exploding?

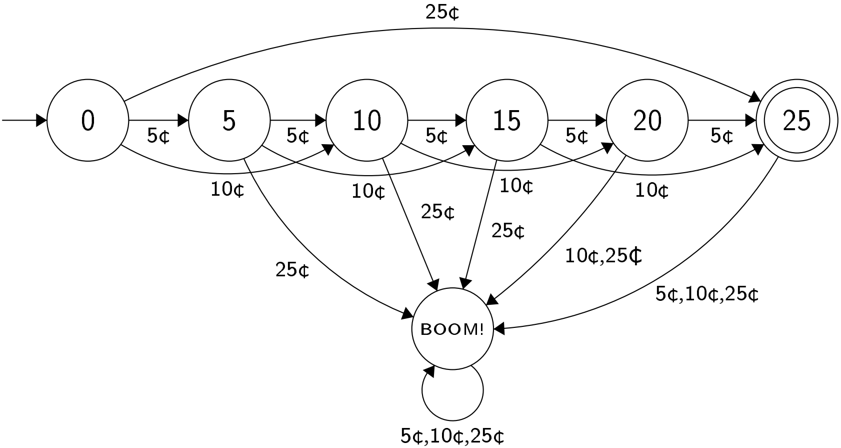
- 25¢
- 5¢ 5¢ 10¢ 5¢
- 10¢ 5¢ 5¢ 5¢

But not:

- 5¢ 5¢
- 10¢ 25¢ (BOOM!)

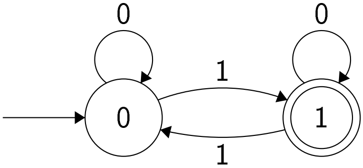
Gum

Here is a finite automaton for the gumball machine:



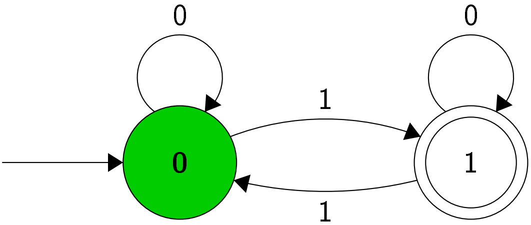
The states tracks how much money has been paid so far. Once the 25 state is reached, the fare is accepted.

Finite Automaton



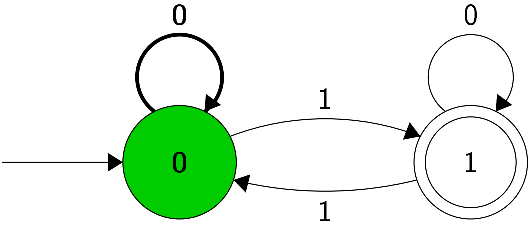
- The automaton starts at the initial state (arrow going in).
- We feed the input into the automaton character by character by following the transitions.
- A string x is accepted by an automaton if it ends on a final (double-circled) state after feeding it through the automaton.

Example



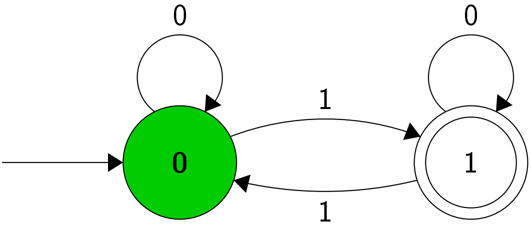
0 1 0 1 1 0

Example



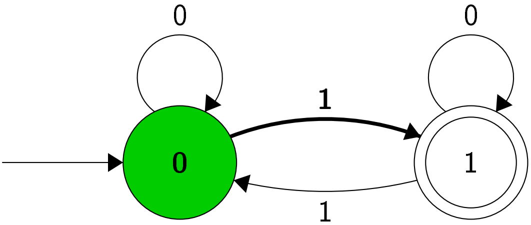
$\mathbf{0}\,1\,0\,1\,1\,0$

Example



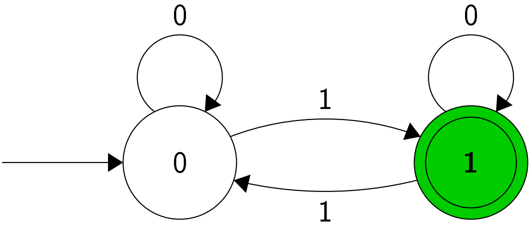
0 1 0 1 1 0

Example



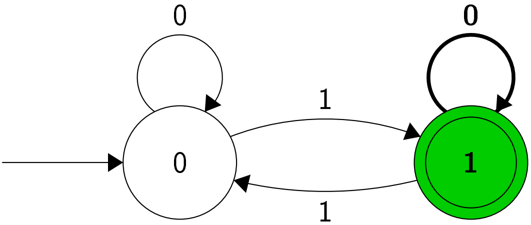
0 **1** 0 1 1 0

Example



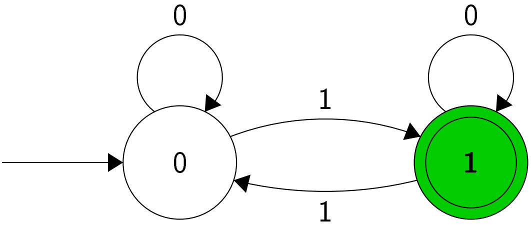
0 1 0 1 1 0

Example



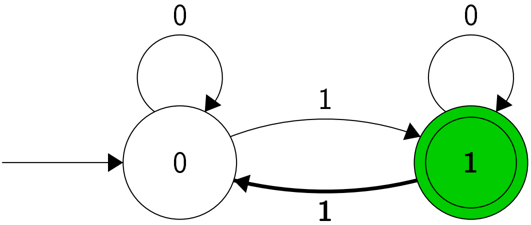
0 1 **0** 1 1 0

Example



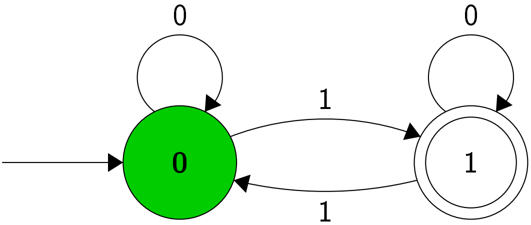
0 1 0 1 1 0

Example



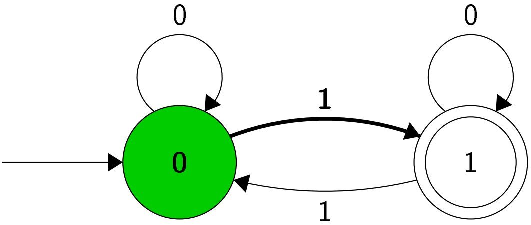
0 1 0 **1** 1 0

Example



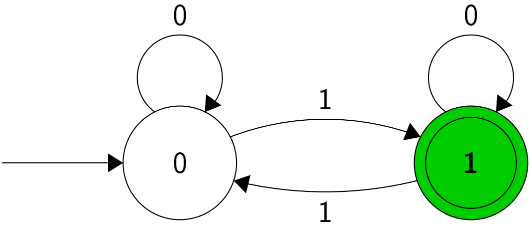
0 1 0 1 1 0

Example



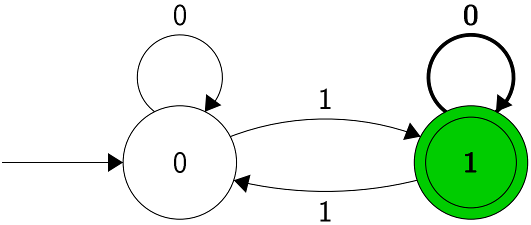
0 1 0 1 **1** 0

Example



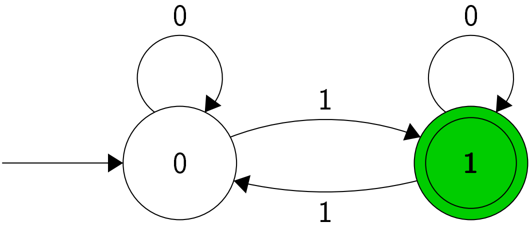
0 1 0 1 1 0

Example



0 1 0 1 1 **0**

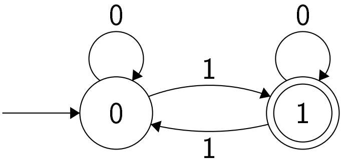
Example



0 1 0 1 1 0

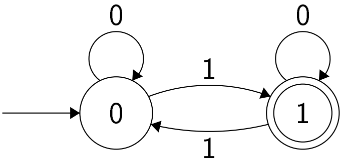
We end in a final (double-circled) state, so **010110** is accepted!

DFA Example



What kinds of strings does this automaton accept?

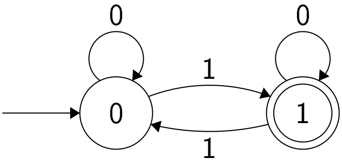
DFA Example



What kinds of strings does this automaton accept?

- 010110 ✓

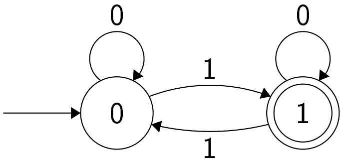
DFA Example



What kinds of strings does this automaton accept?

- 010110 ✓
- 010001111

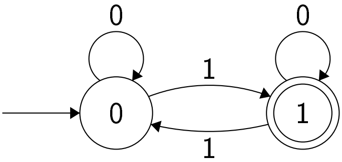
DFA Example



What kinds of strings does this automaton accept?

- 010110 ✓
- 010001111 ✓

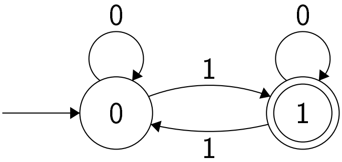
DFA Example



What kinds of strings does this automaton accept?

- 010110 ✓
- 010001111 ✓
- 1111111

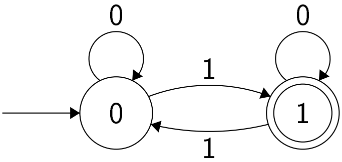
DFA Example



What kinds of strings does this automaton accept?

- 010110 ✓
- 010001111 ✓
- 1111111 ✓

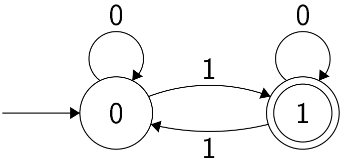
DFA Example



What kinds of strings does this automaton accept?

- 010110 ✓
- 010001111 ✓
- 1111111 ✓
- 0001000

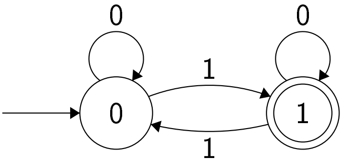
DFA Example



What kinds of strings does this automaton accept?

- 010110 ✓
- 010001111 ✓
- 1111111 ✓
- 0001000 ✓

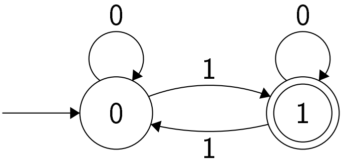
DFA Example



What kinds of strings does this automaton accept?

- 010110 ✓
- 010001111 ✓
- 1111111 ✓
- 0001000 ✓
- 1010

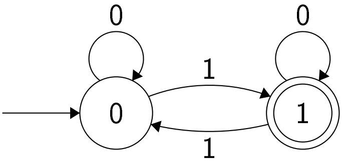
DFA Example



What kinds of strings does this automaton accept?

- 010110 ✓
- 010001111 ✓
- 1111111 ✓
- 0001000 ✓
- 1010 ✗

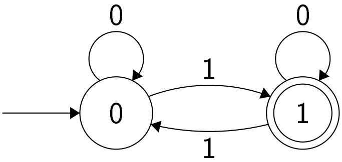
DFA Example



What kinds of strings does this automaton accept?

- 010110 ✓
- 010001111 ✓
- 1111111 ✓
- 0001000 ✓
- 1010 ✗
- 000

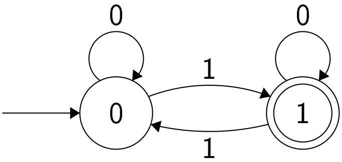
DFA Example



What kinds of strings does this automaton accept?

- 010110 ✓
- 010001111 ✓
- 1111111 ✓
- 0001000 ✓
- 1010 ✗
- 000 ✗

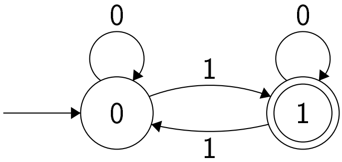
DFA Example



What kinds of strings does this automaton accept?

- 010110 ✓
- 010001111 ✓
- 1111111 ✓
- 0001000 ✓
- 1010 ✗
- 000 ✗
- 11111111

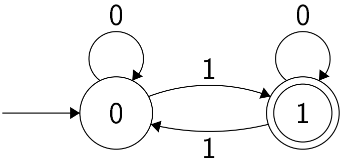
DFA Example



What kinds of strings does this automaton accept?

- 010110 ✓
- 010001111 ✓
- 1111111 ✓
- 0001000 ✓
- 1010 ✗
- 000 ✗
- 11111111 ✗

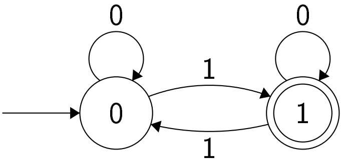
DFA Example



What kinds of strings does this automaton accept?

- 010110 ✓
- 010001111 ✓
- 1111111 ✓
- 0001000 ✓
- 1010 ✗
- 000 ✗
- 11111111 ✗
- 00000000000000000000000000000000

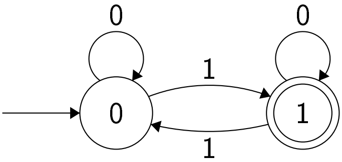
DFA Example



What kinds of strings does this automaton accept?

- 010110 ✓
- 010001111 ✓
- 1111111 ✓
- 0001000 ✓
- 1010 ✗
- 000 ✗
- 11111111 ✗
- 00000000000000000000000000000000 ✗

DFA Example

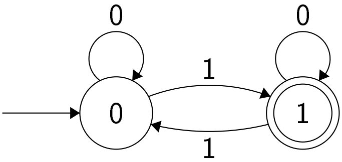


What kinds of strings does this automaton accept?

- 010110 ✓
- 010001111 ✓
- 1111111 ✓
- 0001000 ✓
- 1010 ✗
- 000 ✗
- 11111111 ✗
- 00000000000000000000000000000000 ✗

This automaton accepts a binary string x if and only if

DFA Example



What kinds of strings does this automaton accept?

- 010110 ✓
- 010001111 ✓
- 1111111 ✓
- 0001000 ✓
- 1010 ✗
- 000 ✗
- 11111111 ✗
- 0000000000000000000000000000000 ✗

This automaton accepts a binary string x if and only if the number of 1s in x is odd, or equivalently if the sum of the digits of x is odd.

Applications to Parsing

In Python, variable declaration is done with the following syntax (ignoring whitespace):

```
<variable name>=<value>
```

where the variable name does not start with a digit 0-9.

Applications to Parsing

In Python, variable declaration is done with the following syntax (ignoring whitespace):

```
<variable name>=<value>
```

where the variable name does not start with a digit 0-9.

For example:

- `year=2023`
- `name="Anatoly"`
- `location="SigmaCamp"`

Applications to Parsing

Let's make an automaton that accepts valid Python declarations of integers, i.e. strings of the form

`<variable name>=<digits in 0-9>`

where the variable name does not start with a digit. The allowed characters are a-z, `A-Z`, `0-9`, and =.

Applications to Parsing

Let's make an automaton that accepts valid Python declarations of integers, i.e. strings of the form

`<variable name>=<digits in 0-9>`

where the variable name does not start with a digit. The allowed characters are a-z, `A-Z`, 0-9, and =.

For example, our automaton should accept:

- age=88

Applications to Parsing

Let's make an automaton that accepts valid Python declarations of integers, i.e. strings of the form

`<variable name>=<digits in 0-9>`

where the variable name does not start with a digit. The allowed characters are a-z, `A-Z`, 0-9, and =.

For example, our automaton should accept:

- `age=88`
- `LEGS=2`

Applications to Parsing

Let's make an automaton that accepts valid Python declarations of integers, i.e. strings of the form

`<variable name>=<digits in 0-9>`

where the variable name does not start with a digit. The allowed characters are a-z, `A-Z`, `0-9`, and =.

For example, our automaton should accept:

- `age=88`
- `LEGS=2`
- `hello123Hi=5678901375621365613252348756234786`

Applications to Parsing

Let's make an automaton that accepts valid Python declarations of integers, i.e. strings of the form

`<variable name>=<digits in 0-9>`

where the variable name does not start with a digit. The allowed characters are `a-z`, `A-Z`, `0-9`, and `=`.

For example, our automaton should accept:

- `age=88`
- `LEGS=2`
- `hello123Hi=5678901375621365613252348756234786`

But it should not accept:

- `hello=hi`

Applications to Parsing

Let's make an automaton that accepts valid Python declarations of integers, i.e. strings of the form

`<variable name>=<digits in 0-9>`

where the variable name does not start with a digit. The allowed characters are `a-z`, `A-Z`, `0-9`, and `=`.

For example, our automaton should accept:

- `age=88`
- `LEGS=2`
- `hello123Hi=567890137562136561325234875623478 6`

But it should not accept:

- `hello=hi`
- `SIGMA=2023CAMP`

Applications to Parsing

Let's make an automaton that accepts valid Python declarations of integers, i.e. strings of the form

`<variable name>=<digits in 0-9>`

where the variable name does not start with a digit. The allowed characters are `a-z`, `A-Z`, `0-9`, and `=`.

For example, our automaton should accept:

- `age=88`
- `LEGS=2`
- `hello123Hi=5678901375621365613252348756234786`

But it should not accept:

- `hello=hi`
- `SIGMA=2023CAMP`
- `2023month=8` (variable name can't start with a number)

Applications to Parsing

Let's make an automaton that accepts valid Python declarations of integers, i.e. strings of the form

`<variable name>=<digits in 0-9>`

where the variable name does not start with a digit. The allowed characters are a-z, `A-Z`, 0-9, and =.

For example, our automaton should accept:

- `age=88`
- `LEGS=2`
- `hello123Hi=5678901375621365613252348756234786`

But it should not accept:

- `hello=hi`
- `SIGMA=2023CAMP`
- `2023month=8` (variable name can't start with a number)

Let's do it on the board!

Application: Sum of three squares

We'll say that an integer N is "unfriendly" if it is the sum of three squares of integers:

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

Application: Sum of three squares

We'll say that an integer N is "unfriendly" if it is the sum of three squares of integers:

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

For example, 38 is unfriendly, as

$$38 = 2^2 + 3^2 + 5^2.$$

Application: Sum of three squares

We'll say that an integer N is "unfriendly" if it is the sum of three squares of integers:

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

For example, 38 is unfriendly, as

$$38 = 2^2 + 3^2 + 5^2.$$

$28, 15, 7, 240, 92, 348$ are examples of friendly integers.

Application: Sum of three squares

Legendre's three square theorem says that an integer 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

$$N = 4^a(8b + 7)$$

where a and b are non-negative integers.

Application: Sum of three squares

Legendre's three square theorem says that an integer 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

$$N = 4^a(8b+7)$$

where a and b are non-negative integers.

So, an integer N is friendly if and only if it is of the form $N = 4^a(8b+7)$.

Application: Sum of three squares

Legendre's three square theorem says that an integer 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

$$N = 4^a(8b + 7)$$

where a and b are non-negative integers.

So, an integer N is friendly if and only if it is of the form $N = 4^a(8b + 7)$.

We will make an automaton that decides whether or not N is friendly by reading its binary representation.

Application: Sum of three squares

An integer N is friendly if and only if it is of the form $N = 4^a(8b+7)$.

We want to look at the binary representations of friendly and unfriendly numbers.

Application: Sum of three squares

An integer N is friendly if and only if it is of the form $N = 4^a(8b+7)$.

We want to look at the binary representations of friendly and unfriendly numbers.

- 240 is friendly, and $240 = 4^2(8 \cdot 1 + 7)$. 240 in binary is 11110000.

Application: Sum of three squares

An integer N is friendly if and only if it is of the form $N = 4^a(8b + 7)$.

We want to look at the binary representations of friendly and unfriendly numbers.

- 240 is friendly, and $240 = 4^2(8 \cdot 1 + 7)$. 240 in binary is 11110000.
- 15 is friendly, and $15 = 4^0(8 \cdot 1 + 7)$. 15 in binary is 1111.

Application: Sum of three squares

An integer N is friendly if and only if it is of the form $N = 4^a(8b+7)$.

We want to look at the binary representations of friendly and unfriendly numbers.

- 240 is friendly, and $240 = 4^2(8 \cdot 1 + 7)$. 240 in binary is 11110000.
- 15 is friendly, and $15 = 4^0(8 \cdot 1 + 7)$. 15 in binary is 1111.
- 92 is friendly, and $92 = 4^1(8 \cdot 2 + 7)$. 92 in binary is 1011100.

Application: Sum of three squares

An integer N is friendly if and only if it is of the form $N = 4^a(8b + 7)$.

We want to look at the binary representations of friendly and unfriendly numbers.

- 240 is friendly, and $240 = 4^2(8 \cdot 1 + 7)$. 240 in binary is 11110000.
- 15 is friendly, and $15 = 4^0(8 \cdot 1 + 7)$. 15 in binary is 1111.
- 92 is friendly, and $92 = 4^1(8 \cdot 2 + 7)$. 92 in binary is 1011100.
- 348 is friendly, and $348 = 4^1(8 \cdot 10 + 7)$. 348 in binary is 101011100.

Application: Sum of three squares

An integer N is friendly if and only if it is of the form $N = 4^a(8b+7)$.

We want to look at the binary representations of friendly and unfriendly numbers.

- 240 is friendly, and $240 = 4^2(8 \cdot 1 + 7)$. 240 in binary is 11110000.
- 15 is friendly, and $15 = 4^0(8 \cdot 1 + 7)$. 15 in binary is 1111.
- 92 is friendly, and $92 = 4^1(8 \cdot 2 + 7)$. 92 in binary is 1011100.
- 348 is friendly, and $348 = 4^1(8 \cdot 10 + 7)$. 348 in binary is 101011100.
- 38 is unfriendly, as $38 = 2^2 + 3^2 + 5^2$. 38 in binary is 100110.

Application: Sum of three squares

An integer N is friendly if and only if it is of the form $N = 4^a(8b+7)$.

We want to look at the binary representations of friendly and unfriendly numbers.

- 240 is friendly, and $240 = 4^2(8 \cdot 1 + 7)$. 240 in binary is 11110000.
- 15 is friendly, and $15 = 4^0(8 \cdot 1 + 7)$. 15 in binary is 1111.
- 92 is friendly, and $92 = 4^1(8 \cdot 2 + 7)$. 92 in binary is 1011100.
- 348 is friendly, and $348 = 4^1(8 \cdot 10 + 7)$. 348 in binary is 101011100.
- 38 is unfriendly, as $38 = 2^2 + 3^2 + 5^2$. 38 in binary is 100110.
- 33 is unfriendly, as $33 = 2^2 + 2^2 + 5^2$. 33 in binary is 100001.

Application: Sum of three squares

Suppose $N = 4^a(8b+7)$ where a and b are non-negative integers. What can we say about the binary representation of N?

Application: Sum of three squares

Suppose $N = 4^a(8b+7)$ where a and b are non-negative integers. What can we say about the binary representation of N?

- If b is a non-negative integer, then the binary representation $(8b)_2$ looks like

$$\underbrace{\ldots}_{\text{1s and 0s}}\ 000$$

Application: Sum of three squares

Suppose $N = 4^a(8b+7)$ where a and b are non-negative integers. What can we say about the binary representation of N?

- If b is a non-negative integer, then the binary representation $(8b)_2$ looks like

$$\underbrace{\ldots}_{\text{1s and 0s}}\ 000$$

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

$$\underbrace{\ldots}_{\text{1s and 0s}}\ 111$$

Application: Sum of three squares

Suppose $N = 4^a(8b+7)$ where a and b are non-negative integers. What can we say about the binary representation of N?

- If b is a non-negative integer, then the binary representation $(8b)_2$ looks like

$$\underbrace{\cdots}_{\text{1s and 0s}}\ 000$$

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

$$\underbrace{\cdots}_{\text{1s and 0s}}\ 111$$

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

$$\underbrace{\cdots}_{\text{1s and 0s}}\ 111\ \underbrace{00\cdots00}_{\substack{\text{even \# of 0's,}\\ \text{may be empty}}}$$

Application: Sum of three squares

So N is friendly if and only if its binary representation is of the form

$$\underbrace{\ldots}_{\text{1s and 0s}} \; 111 \; \underbrace{00\cdots00}_{\substack{\text{even \# of 0's,} \\ \text{may be empty}}}$$

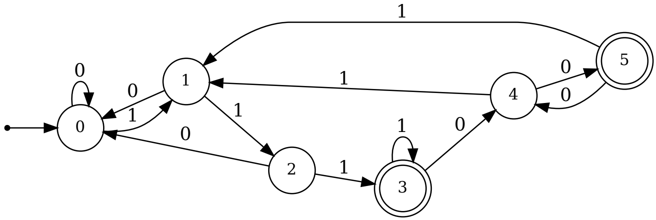
Let's make an automaton on the board that recognizes this!

Application: Sum of three squares

So N is friendly if and only if its binary representation is of the form

$$\underbrace{\ldots}_{\text{1s and 0s}} \; 111 \; \underbrace{00\cdots00}_{\substack{\text{even \# of 0's,}\\ \text{may be empty}}}$$

Let's make an automaton on the board that recognizes this!

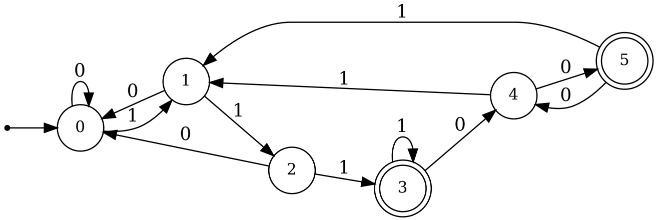


Application: Sum of three squares

So N is friendly if and only if its binary representation is of the form

$$\underbrace{\cdots}_{\text{1s and 0s}} \; 111 \; \underbrace{00\cdots00}_{\substack{\text{even \# of 0's,}\\ \text{may be empty}}}$$

Let's make an automaton on the board that recognizes this!



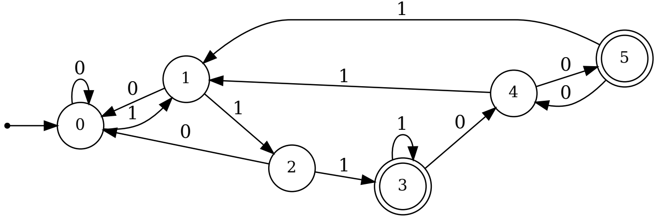
So this automaton accepts $(N)_2$ if and only if N is friendly.

Application: Sum of three squares

So N is friendly if and only if its binary representation is of the form

$$\underbrace{\ldots}_{\text{1s and 0s}} \; 111 \; \underbrace{00\cdots00}_{\substack{\text{even \# of 0's,} \\ \text{may be empty}}}$$

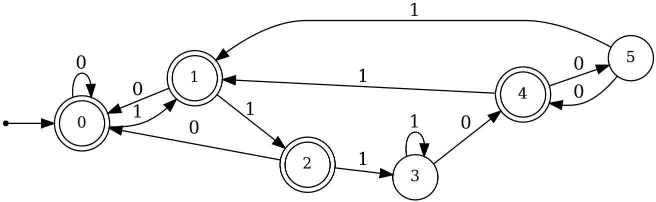
Let's make an automaton on the board that recognizes this!



So this automaton accepts $(N)_2$ if and only if N is friendly.
What about an automaton for unfriendly N?

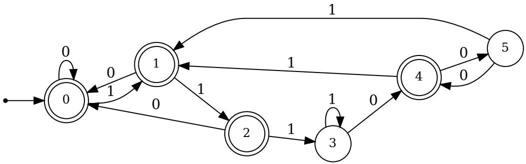
Example: Sum of three squares

To accept all $(N)_2$ if and only if N is unfriendly, just flip the final states:



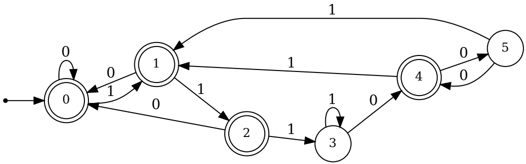
Then everything that wasn't accepted before is now accepted, and vice versa.

Limitations of finite automata



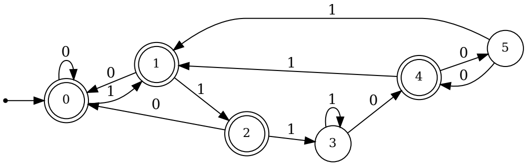
- Finite automata are quite limited, as their number of states is fixed!

Limitations of finite automata



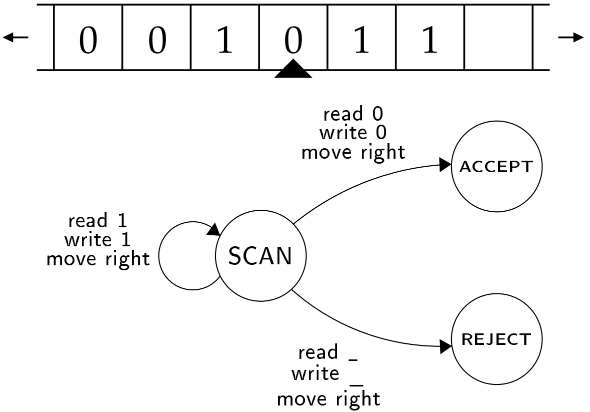
- Finite automata are quite limited, as their number of states is fixed!
- For example, no finite automaton can accept only strings of the form $0\ldots0\,1\ldots1$, where the number of 0s and 1s is the same.

Limitations of finite automata



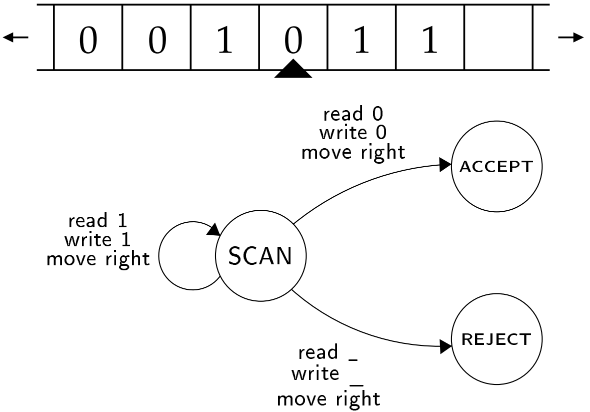
- Finite automata are quite limited, as their number of states is fixed!
- For example, no finite automaton can accept only strings of the form 0…01…1, where the number of 0s and 1s is the same.
  - Intuitively, it's because finite automata can't "count" arbitrarily high.

Turing machine



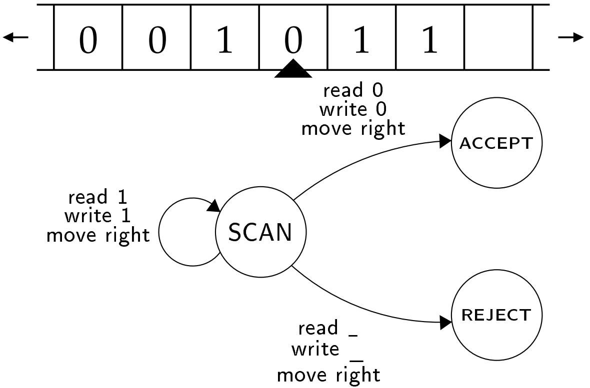
- To make our automaton more powerful, we're going to give it an infinitely long tape that it can read from and write to, which will act as its memory. The tape consists of infinitely many cells.

Turing machine



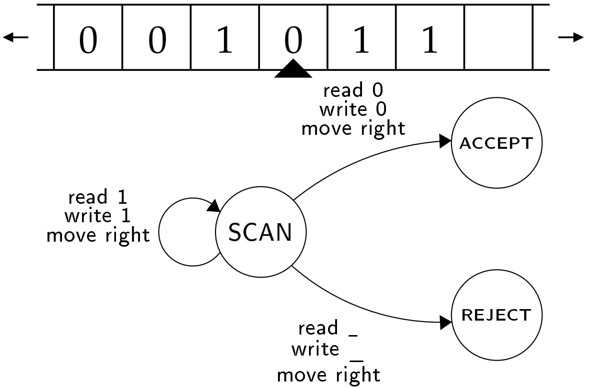
- To make our automaton more powerful, we're going to give it an infinitely long tape that it can read from and write to, which will act as its memory. The tape consists of infinitely many cells.
- This is called a Turing machine.

How Turing machines work



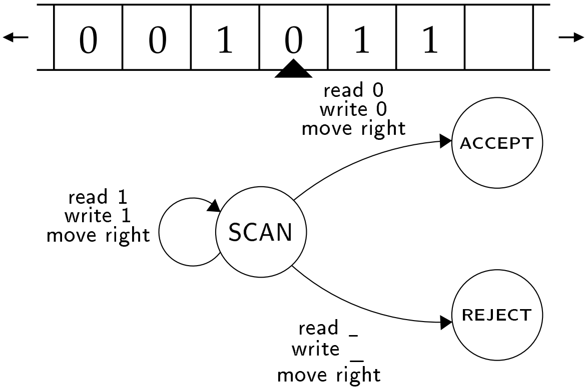
- To give a Turing machine a (finite) input, we write it on the tape.

How Turing machines work



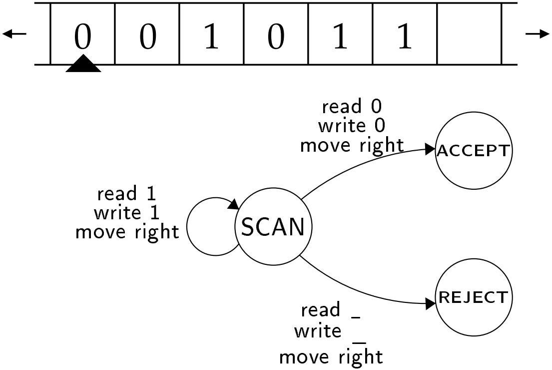
- To give a Turing machine a (finite) input, we write it on the tape.
- The Turing machine then moves a head across the tape according to its state control (underlying automaton). It can only read the cell that its head is pointing to.

How Turing machines work



- To give a Turing machine a (finite) input, we write it on the tape.
- The Turing machine then moves a head across the tape according to its state control (underlying automaton). It can only read the cell that its head is pointing to.
- Turing machines can accept or reject an input.

Turing machine example

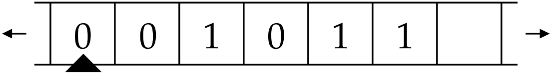


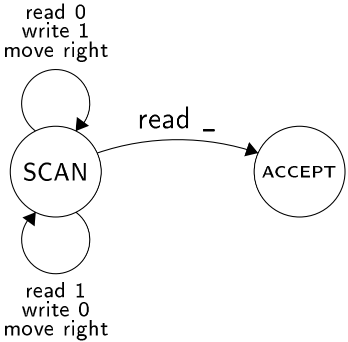
This Turing machine will read an input comprised of 1s and 0s and:

- accept if the input has any 1s, and
- reject if the input has no 1s.

Another example

This Turing machine will read an input comprised of 1s and 0s and flip every 0 to a 1 and vice versa. Once it is done, it will accept. The output is the bitwise complement of the input.





Turing machines are POWERFUL

Church-Turing Thesis

Every computation that can be done in the real world can be performed by a Turing machine.

Turing machines are POWERFUL

Church-Turing Thesis

Every computation that can be done in the real world can be performed by a Turing machine.

- In other words, anything that can be done on a computer can be done by a Turing machine.

Turing machines are POWERFUL

Church-Turing Thesis

Every computation that can be done in the real world can be performed by a Turing machine.

- In other words, anything that can be done on a computer can be done by a Turing machine.
- Conversely, Turing machines can be simulated by a computer.

Turing machines are POWERFUL

Church-Turing Thesis

Every computation that can be done in the real world can be performed by a Turing machine.

- In other words, anything that can be done on a computer can be done by a Turing machine.
- Conversely, Turing machines can be simulated by a computer.
- So, Turing machines can do everything that computers can do.

Turing machines are POWERFUL

Church-Turing Thesis

Every computation that can be done in the real world can be performed by a Turing machine.

- In other words, anything that can be done on a computer can be done by a Turing machine.
- Conversely, Turing machines can be simulated by a computer.
- So, Turing machines can do everything that computers can do.
- To study how efficient an algorithm is on a computer, we can study how efficient it is on a Turing machine.

Turing completeness

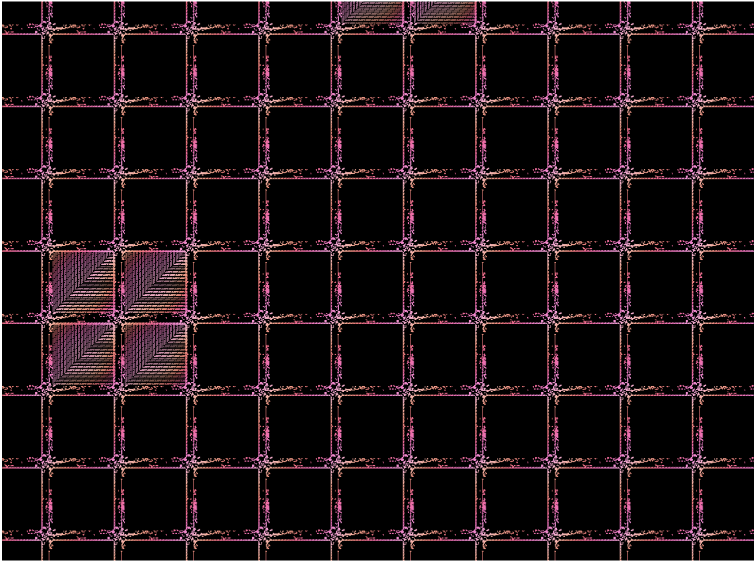
- Any computational model that can simulate a Turing machine is called Turing complete.

Turing completeness

- Any computational model that can simulate a Turing machine is called Turing complete.
- Anything that's Turing complete can simulate any classical computer.

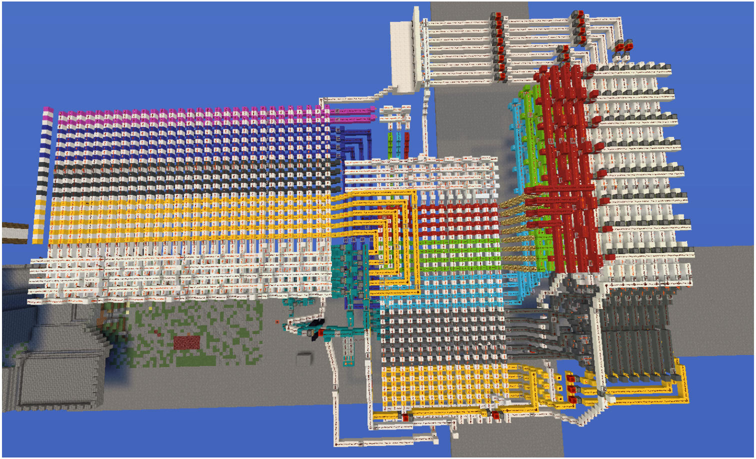
Turing completeness

Conway's Game of Life can simulate itself!



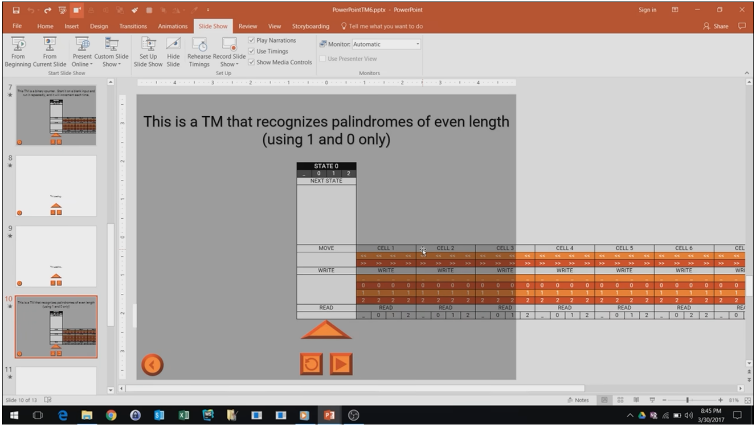
Turing completeness

Computers can be built in Minecraft using redstone!



Turing completeness

Microsoft PowerPoint can simulate Turing machines!



Summary

- Automata theory is the study of abstract computational models.

Summary

- Automata theory is the study of abstract computational models.
- It is is a foundational part of theoretical computer science, with automata appearing everywhere from the analysis of algorithms, and the design of programming languages.

Summary

- Automata theory is the study of abstract computational models.
- It is is a foundational part of theoretical computer science, with automata appearing everywhere from the analysis of algorithms, and the design of programming languages.
- Finite automata are the simplest model of computation, while still being extremely useful.

Summary

- Automata theory is the study of abstract computational models.
- It is is a foundational part of theoretical computer science, with automata appearing everywhere from the analysis of algorithms, and the design of programming languages.
- Finite automata are the simplest model of computation, while still being extremely useful.
- By adding an infinite tape to a finite automaton, we get a Turing machine, which are equivalent in power to computers.

Summary

- Automata theory is the study of abstract computational models.
- It is is a foundational part of theoretical computer science, with automata appearing everywhere from the analysis of algorithms, and the design of programming languages.
- Finite automata are the simplest model of computation, while still being extremely useful.
- By adding an infinite tape to a finite automaton, we get a Turing machine, which are equivalent in power to computers.
- Instead of making a program in Python, write it in PowerPoint!

Thank you!
Any questions?