Teoria degli Algoritmi

Corso di Laurea Magistrale in Matematica Applicata

a.a. 2020-21

Gabriele Tolomei

Dipartimento di Informatica

Sapienza Università di Roma

tolomei@di.uniroma1.it

Lecture 3: Decidability

Table of Contents

1. Computability
2. Diagonalization
3. The Halting Problem
4. Beyond Undecidability
5. Summary

Table of Contents

1. Computability
2. Diagonalization
3. The Halting Problem
4. Beyond Undecidability
5. Summary

Decidability & Recognizability

- We introduced the Turing Machine (TM) as a model of general-purpose computation

Decidability & Recognizability

- We introduced the Turing Machine (TM) as a model of general-purpose computation
- We defined the (formal) notion of algorithm in terms of TMs leveraging the Church-Turing thesis

Decidability & Recognizability

- We introduced the Turing Machine (TM) as a model of general-purpose computation
- We defined the (formal) notion of algorithm in terms of TMs leveraging the Church-Turing thesis
- In other words, we identify the set of **computable functions** with those calculated by a Turing machine:
    - **partial** computable functions are those **recognized** by a TM (i.e., the machine may never halt)
    - **total** computable functions are those **decided** by a TM (i.e., the machine always halts)

Decidability & Recognizability

- We introduced the Turing Machine (TM) as a model of general-purpose computation
- We defined the (formal) notion of algorithm in terms of TMs leveraging the Church-Turing thesis
- In other words, we identify the set of **computable functions** with those calculated by a Turing machine:
    - **partial** computable functions are those **recognized** by a TM (i.e., the machine may never halt)
    - **total** computable functions are those **decided** by a TM (i.e., the machine always halts)
- We also refer to (**Turing-**)**recognizable** or **recursively-enumerable** languages and (**Turing-**)**decidable** or **recursive** languages

Uncomputable Functions

Question

A natural question arises:

is there any problem (i.e., function) that cannot be solved by **any** *algorithm?*

Uncomputable Functions

Question

A natural question arises:

is there any problem (i.e., function) that cannot be solved by **any** *algorithm?*

To put it differently, yet equivalently:

is there any function that cannot be computed by **any** *Turing machine?*

Uncomputable Functions: Why Bother?

- Why should we care about problems that cannot be algorithmically solved?

Uncomputable Functions: Why Bother?

- Why should we care about problems that cannot be algorithmically solved?
- After all, showing that a problem is unsolvable does not appear to be of any interest

Uncomputable Functions: Why Bother?

- Why should we care about problems that cannot be algorithmically solved?
- After all, showing that a problem is unsolvable does not appear to be of any interest
- Two reasons why we should bother of problems that cannot be solved by an algorithm:
    - To realize they must **simplified** first, before searching for an algorithmic solution to them

Uncomputable Functions: Why Bother?

- Why should we care about problems that cannot be algorithmically solved?
- After all, showing that a problem is unsolvable does not appear to be of any interest
- Two reasons why we should bother of problems that cannot be solved by an algorithm:
  - To realize they must **simplified** first, before searching for an algorithmic solution to them
  - To stimulate your imagination!

Table of Contents

1. Computability
2. Diagonalization
3. The Halting Problem
4. Beyond Undecidability
5. Summary

The Size of Sets

- A clever technique used to proof the existance of uncomputable functions/undecidable languages

The Size of Sets

- A clever technique used to proof the existance of uncomputable functions/undecidable languages
- This was discovered by Georg Cantor in 1873, who was concerned with the problem of measuring the size of (infinite) sets

The Size of Sets

- A clever technique used to proof the existance of uncomputable functions/undecidable languages
- This was discovered by Georg Cantor in 1873, who was concerned with the problem of measuring the size of (infinite) sets

Question

If we have two sets A and B, how can we tell if one is larger than the other or if they are of the same size?

The Size of Sets

- A clever technique used to proof the existance of uncomputable functions/undecidable languages
- This was discovered by Georg Cantor in 1873, who was concerned with the problem of measuring the size of (infinite) sets

Question

If we have two sets A and B, how can we tell if one is larger than the other or if they are of the same size?

- For finite sets, the answer is of course straightforward: just count the elements of each A and B!

The Size of Sets

- A clever technique used to proof the existance of uncomputable functions/undecidable languages
- This was discovered by Georg Cantor in 1873, who was concerned with the problem of measuring the size of (infinite) sets

Question

If we have two sets A and B, how can we tell if one is larger than the other or if they are of the same size?

- For finite sets, the answer is of course straightforward: just count the elements of each A and B!
- The same does not work for infinite set as we will never finish counting!

The Size of Sets: Example

Example

Consider the set of **even** natural $\mathbb{E} = \{n \in \mathbb{N} \mid n \mod 2 = 0\}$.
Then, consider the set of all possible binary strings of any (finite) length $\Sigma^* = \{0, 1\}^*$

The Size of Sets: Example

Example

Consider the set of **even** natural $\mathbb{E} = \{n \in \mathbb{N} \mid n \mod 2 = 0\}$.
Then, consider the set of all possible binary strings of any (finite) length $\Sigma^* = \{0, 1\}^*$

Question

Of course, both $\mathbb{E}$ and Σ^* are infinite (thus, larger than any finite set).
However, is one of the two larger than the other? Can we figure this out?

The Size of Sets: Cantor's Solution

- Cantor proposed a brilliant solution to the problem posed before

The Size of Sets: Cantor's Solution

- Cantor proposed a brilliant solution to the problem posed before
- He observed that two finite sets A and B have the same size if the elements of one set can be **paired** with the elements of the other set

The Size of Sets: Cantor's Solution

- Cantor proposed a brilliant solution to the problem posed before
- He observed that two finite sets A and B have the same size if the elements of one set can be **paired** with the elements of the other set
- This method compares the size of sets **without** resorting to counting!

The Size of Sets: Cantor's Solution

- Cantor proposed a brilliant solution to the problem posed before
- He observed that two finite sets A and B have the same size if the elements of one set can be **paired** with the elements of the other set
- This method compares the size of sets **without** resorting to counting!
- Interestingly enough, this approach extends also to **infinite sets**

The Size of Sets: Cantor's Solution

Definition (Same Size Sets)

Suppose we have two sets A and B, and a function $f : A \mapsto B$.
We say that f is **one-to-one** (or **injective**) if it never maps two different elements to the same place, i.e., $\forall a, a' \in A,\ a \neq a' \Rightarrow f(a) \neq f(a')$

The Size of Sets: Cantor's Solution

Definition (Same Size Sets)

Suppose we have two sets A and B, and a function $f : A \mapsto B$.
We say that f is **one-to-one** (or **injective**) if it never maps two different elements to the same place, i.e., $\forall a, a' \in A,\ a \neq a' \Rightarrow f(a) \neq f(a')$
Moreover, we say that f is **onto** (or **surjective**) if $\forall b \in B\ \exists a \in A$ s.t. $f(a) = b$.

The Size of Sets: Cantor's Solution

Definition (Same Size Sets)

Suppose we have two sets A and B, and a function $f : A \mapsto B$.
We say that f is **one-to-one** (or **injective**) if it never maps two different elements to the same place, i.e., $\forall a, a' \in A,\ a \neq a' \Rightarrow f(a) \neq f(a')$
Moreover, we say that f is **onto** (or **surjective**) if $\forall b \in B\ \exists a \in A$ s.t. $f(a) = b$.
We say that A and B are **same size sets** if there is a one-to-one, onto function $f : A \mapsto B$; such a function is called a **bijection** or **correspondence**.

The Size of Sets: Cantor's Solution

Definition (Same Size Sets)

Suppose we have two sets A and B, and a function $f : A \mapsto B$.
We say that f is **one-to-one** (or **injective**) if it never maps two different elements to the same place, i.e., $\forall a, a' \in A,\ a \neq a' \Rightarrow f(a) \neq f(a')$
Moreover, we say that f is **onto** (or **surjective**) if $\forall b \in B\ \exists a \in A$ s.t. $f(a) = b$.
We say that A and B are **same size sets** if there is a one-to-one, onto function $f : A \mapsto B$; such a function is called a **bijection** or **correspondence**.
In a correspondence between A and B, every element of A maps to a unique element of B and every element of B has a unique element of A that maps to it.

The Size of the Set of Even Natural Numbers

Example (The size of $\mathbb{N}$ vs. the size of $\mathbb{E}$)

Let $\mathbb{N}$ be the set of natural numbers, i.e., $\mathbb{N} = \{1, 2, 3, \ldots\}$, and let $\mathbb{E}$ be the set of **even** natural numbers, i.e., $\mathbb{E} = \{2, 4, 6, \ldots\}$.
Using Cantor's argument, prove that $\mathbb{N}$ and $\mathbb{E}$ have the same size.

The Size of the Set of Even Natural Numbers

Example (The size of $\mathbb{N}$ vs. the size of $\mathbb{E}$)

Let $\mathbb{N}$ be the set of natural numbers, i.e., $\mathbb{N} = \{1, 2, 3, \ldots\}$, and let $\mathbb{E}$ be the set of **even** natural numbers, i.e., $\mathbb{E} = \{2, 4, 6, \ldots\}$.
Using Cantor's argument, prove that $\mathbb{N}$ and $\mathbb{E}$ have the same size.

- Intuitively, this sounds odd: $\mathbb{E}$ seems "smaller" than $\mathbb{N}$, as the former is a proper subset of the latter

The Size of the Set of Even Natural Numbers

Example (The size of $\mathbb{N}$ vs. the size of $\mathbb{E}$)

Let $\mathbb{N}$ be the set of natural numbers, i.e., $\mathbb{N} = \{1, 2, 3, \ldots\}$, and let $\mathbb{E}$ be the set of **even** natural numbers, i.e., $\mathbb{E} = \{2, 4, 6, \ldots\}$.
Using Cantor's argument, prove that $\mathbb{N}$ and $\mathbb{E}$ have the same size.

- Intuitively, this sounds odd: $\mathbb{E}$ seems "smaller" than $\mathbb{N}$, as the former is a proper subset of the latter
- However, we can find a correspondence $f : \mathbb{N} \mapsto \mathbb{E}$ which maps each element of $\mathbb{N}$ to an element of $\mathbb{E}$:

$$\forall n \in \mathbb{N},\ f(n) = 2n \in \mathbb{E}$$

Countable Set

Definition (Countable Set)

A set A is **countable** if either it is finite or it has the same size of $\mathbb{N}$

The Size of the Set of Rational Numbers

Example (The size of $\mathbb{Q}$ vs. the size of $\mathbb{N}$)

Let $\mathbb{Q}$ be the set of (positive) rational numbers, i.e., $\mathbb{Q} = \{\frac{m}{n} \mid m, n \in \mathbb{N}\}$. Using Cantor's argument, prove that $\mathbb{Q}$ is countable as there exists a correspondence with $\mathbb{N}$.

The Size of the Set of Rational Numbers

Example (The size of $\mathbb{Q}$ vs. the size of $\mathbb{N}$)

Let $\mathbb{Q}$ be the set of (positive) rational numbers, i.e., $\mathbb{Q} = \{\frac{m}{n} \mid m, n \in \mathbb{N}\}$. Using Cantor's argument, prove that $\mathbb{Q}$ is countable as there exists a correspondence with $\mathbb{N}$.

- Intuitively, this sounds even stranger than the example before: $\mathbb{Q}$ seems "much larger" than $\mathbb{N}$

The Size of the Set of Rational Numbers

Example (The size of $\mathbb{Q}$ vs. the size of $\mathbb{N}$)

Let $\mathbb{Q}$ be the set of (positive) rational numbers, i.e., $\mathbb{Q} = \{\frac{m}{n} \mid m, n \in \mathbb{N}\}$. Using Cantor's argument, prove that $\mathbb{Q}$ is countable as there exists a correspondence with $\mathbb{N}$.

- Intuitively, this sounds even stranger than the example before: $\mathbb{Q}$ seems "much larger" than $\mathbb{N}$
- However, we can find a correspondence $f : \mathbb{Q} \mapsto \mathbb{N}$ which maps each element of $\mathbb{Q}$ to an element of $\mathbb{N}$

The Size of the Set of Rational Numbers

- An easy way to build the correspondence $f : \mathbb{Q} \mapsto \mathbb{N}$ is to
  - list **all** the elements of $\mathbb{Q} : \frac{1}{1}, \frac{1}{2}, \frac{1}{3}, \ldots, \frac{2}{1}, \frac{2}{2}, \ldots$
  - pair the first element of the list with the number 1 from $\mathbb{N}$, the second with 2, and so on and so forth
  - ensure that every member of $\mathbb{Q}$ appears exactly once (e.g., $\frac{1}{1} = \frac{2}{2} = \frac{3}{3} \ldots$)

The Infinite Matrix of Rational Numbers

- To build the list of all the elements of $\mathbb{Q}$, just make an infinite matrix containing all the positive rational numbers

The Infinite Matrix of Rational Numbers

- To build the list of all the elements of $\mathbb{Q}$, just make an infinite matrix containing all the positive rational numbers
- The i-th row contains all the numbers whose numerator is equal to i

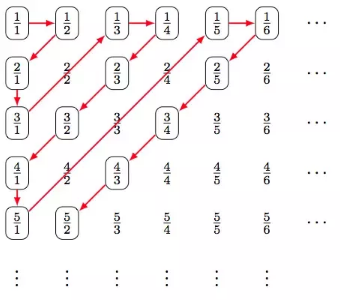
The Infinite Matrix of Rational Numbers

- To build the list of all the elements of $\mathbb{Q}$, just make an infinite matrix containing all the positive rational numbers
- The i-th row contains all the numbers whose numerator is equal to i
- The j-th column contains all the numbers whose denominator is equal to j

The Infinite Matrix of Rational Numbers

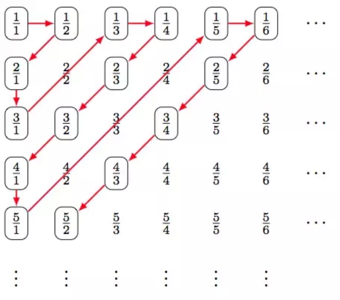
- To build the list of all the elements of $\mathbb{Q}$, just make an infinite matrix containing all the positive rational numbers
- The i-th row contains all the numbers whose numerator is equal to i
- The j-th column contains all the numbers whose denominator is equal to j
- In other words, the rational number $q = \frac{i}{j}$ is located on the i-th row and the j-th column of the matrix

The Infinite Matrix of Rational Numbers



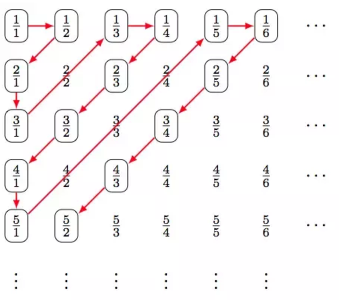
- We must turn the infinite matrix into a **list**

The Infinite Matrix of Rational Numbers



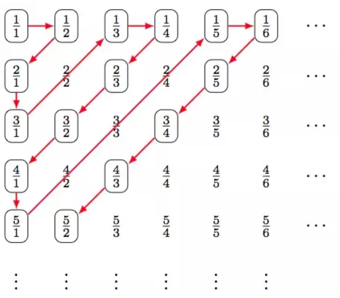
- We must turn the infinite matrix into a **list**
- A first (bad) attempt of doing this would be to begin the list with all the elements in the first row

The Infinite Matrix of Rational Numbers



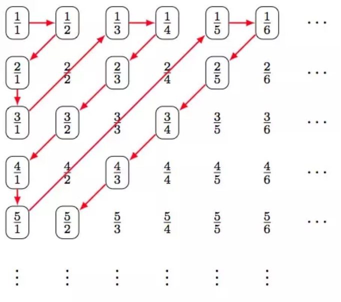
- We must turn the infinite matrix into a **list**
- A first (bad) attempt of doing this would be to begin the list with all the elements in the first row
- That would not work as the first row is infinite and we would never get to the second row!

The Infinite Matrix of Rational Numbers



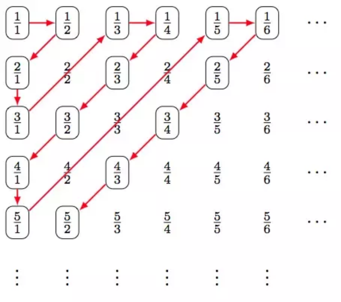
- We must turn the infinite matrix into a **list**
- A first (bad) attempt of doing this would be to begin the list with all the elements in the first row
- That would not work as the first row is infinite and we would never get to the second row!
- Instead, we list the elements following the diagonals, starting from the top-left corner

The Infinite Matrix of Rational Numbers



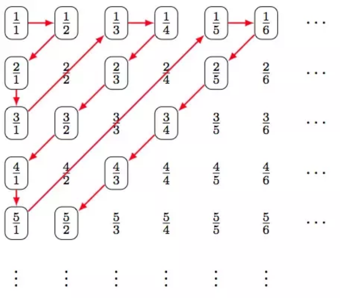
- The first diagonal contains the single element $\frac{1}{1}$

The Infinite Matrix of Rational Numbers



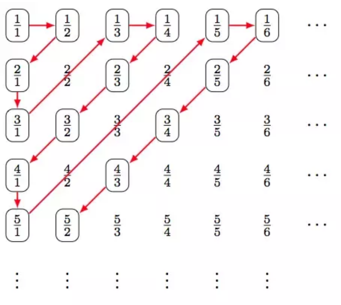
- The first diagonal contains the single element $\frac{1}{1}$
- The second diagonal, instead, contains two elements: $\frac{1}{2}$ and $\frac{2}{1}$

The Infinite Matrix of Rational Numbers



- The first diagonal contains the single element $\frac{1}{1}$
- The second diagonal, instead, contains two elements: $\frac{1}{2}$ and $\frac{2}{1}$
- The first three elements of our list are: $\frac{1}{1}, \frac{1}{2}, \frac{2}{1}$

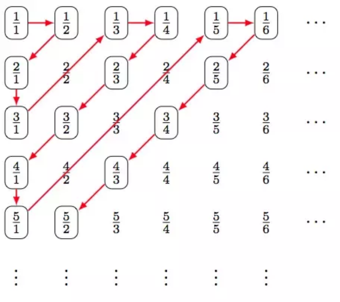
The Infinite Matrix of Rational Numbers



- The first diagonal contains the single element $\frac{1}{1}$
- The second diagonal, instead, contains two elements: $\frac{1}{2}$ and $\frac{2}{1}$
- The first three elements of our list are: $\frac{1}{1}, \frac{1}{2}, \frac{2}{1}$
- In the third diagonal, things get a little bit more complicated as this contains: $\frac{1}{3}, \frac{2}{2}, \frac{3}{1}$

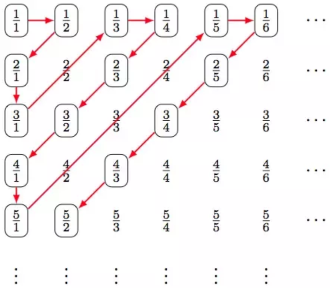
SAPIENZA UNIVERSITÀ DI ROMA

The Infinite Matrix of Rational Numbers



- The first diagonal contains the single element $\frac{1}{1}$
- The second diagonal, instead, contains two elements: $\frac{1}{2}$ and $\frac{2}{1}$
- The first three elements of our list are: $\frac{1}{1}, \frac{1}{2}, \frac{2}{1}$
- In the third diagonal, things get a little bit more complicated as this contains: $\frac{1}{3}, \frac{2}{2}, \frac{3}{1}$
- We can't add those to our list as we would repeat $\frac{1}{1} = \frac{2}{2}$, so we add only $\frac{3}{1}$ and $\frac{1}{3}$

The Infinite Matrix of Rational Numbers



- The first diagonal contains the single element $\frac{1}{1}$
- The second diagonal, instead, contains two elements: $\frac{1}{2}$ and $\frac{2}{1}$
- The first three elements of our list are: $\frac{1}{1}, \frac{1}{2}, \frac{2}{1}$
- In the third diagonal, things get a little bit more complicated as this contains: $\frac{1}{3}, \frac{2}{2}, \frac{3}{1}$
- We can't add those to our list as we would repeat $\frac{1}{1} = \frac{2}{2}$, so we add only $\frac{3}{1}$ and $\frac{1}{3}$
- If we keep going this way we will obtain the list of all the elements of $\mathbb{Q}$

Uncountable Sets

- We may be tempted to conclude that **any** two infinite sets have the same size...

Uncountable Sets

- We may be tempted to conclude that **any** two infinite sets have the same size...
- After all, we just need to show that a **correspondence** between the two exists

Uncountable Sets

- We may be tempted to conclude that **any** two infinite sets have the same size...
- After all, we just need to show that a **correspondence** between the two exists
- However, for some infinite sets no correspondence (with the set $\mathbb{N}$) exists

Uncountable Sets

- We may be tempted to conclude that **any** two infinite sets have the same size...
- After all, we just need to show that a **correspondence** between the two exists
- However, for some infinite sets no correspondence (with the set $\mathbb{N}$) exists
- We call those sets **uncountable**

The Set of Real Numbers $\mathbb{R}$

- The set of real numbers $\mathbb{R}$ is an example of an uncountable set

The Set of Real Numbers $\mathbb{R}$

- The set of real numbers $\mathbb{R}$ is an example of an uncountable set
- A real number is one that has a **decimal representation**
  - For example: $\pi = 3.1415926\ldots$ or $\sqrt{2} = 1.4142135\ldots$ are real numbers

The Set of Real Numbers $\mathbb{R}$

- The set of real numbers $\mathbb{R}$ is an example of an uncountable set
- A real number is one that has a **decimal representation**
  - For example: $\pi = 3.1415926\ldots$ or $\sqrt{2} = 1.4142135\ldots$ are real numbers
- Cantor proved that $\mathbb{R}$ is uncountable, and to do so he introduced the so-called **diagonalization method**

$\mathbb{R}$ is Uncountable: Cantor's Proof

- To show that $\mathbb{R}$ is uncountable, we must show that no correspondence exists between $\mathbb{N}$ and $\mathbb{R}$

$\mathbb{R}$ is Uncountable: Cantor's Proof

- To show that $\mathbb{R}$ is uncountable, we must show that no correspondence exists between $\mathbb{N}$ and $\mathbb{R}$
- The proof works by contradiction, i.e., assuming that a correspondence $f : \mathbb{N} \mapsto \mathbb{R}$ exists

$\mathbb{R}$ is Uncountable: Cantor's Proof

- To show that $\mathbb{R}$ is uncountable, we must show that no correspondence exists between $\mathbb{N}$ and $\mathbb{R}$
- The proof works by contradiction, i.e., assuming that a correspondence $f : \mathbb{N} \mapsto \mathbb{R}$ exists
- Eventually, we want to get to an absurd!

$\mathbb{R}$ is Uncountable: Cantor's Proof

- To show that $\mathbb{R}$ is uncountable, we must show that no correspondence exists between $\mathbb{N}$ and $\mathbb{R}$
- The proof works by contradiction, i.e., assuming that a correspondence $f : \mathbb{N} \mapsto \mathbb{R}$ exists
- Eventually, we want to get to an absurd!
- For f to be a correspondence it has to pair **all** the elements of $\mathbb{N}$ with **all** the elements of $\mathbb{R}$

$\mathbb{R}$ is Uncountable: Cantor's Proof

- To show that $\mathbb{R}$ is uncountable, we must show that no correspondence exists between $\mathbb{N}$ and $\mathbb{R}$
- The proof works by contradiction, i.e., assuming that a correspondence $f : \mathbb{N} \mapsto \mathbb{R}$ exists
- Eventually, we want to get to an absurd!
- For f to be a correspondence it has to pair **all** the elements of $\mathbb{N}$ with **all** the elements of $\mathbb{R}$
- The idea is to find an element $x \in \mathbb{R}$ that is not paired with any element of $\mathbb{N}$

$\mathbb{R}$ is Uncountable: Cantor's Proof

- To find the counterexample $x \in \mathbb{R}$ that leads to a contradiction, we use an example

$\mathbb{R}$ is Uncountable: Cantor's Proof

- To find the counterexample $x \in \mathbb{R}$ that leads to a contradiction, we use an example
- Suppose (again) that f exists and some of its values are:
  - $f(1) = \pi = 3.14159\ldots$
  - $f(2) = 55.55555\ldots$
  - $f(3) = 0.12345\ldots$
  - $f(4) = 0.50000\ldots$
  - $\ldots$

$\mathbb{R}$ is Uncountable: Cantor's Proof

- To find the counterexample $x \in \mathbb{R}$ that leads to a contradiction, we use an example
- Suppose (again) that f exists and some of its values are:
  - $f(1) = \pi = 3.14159\ldots$
  - $f(2) = 55.55555\ldots$
  - $f(3) = 0.12345\ldots$
  - $f(4) = 0.50000\ldots$
  - $\ldots$
- Now, we construct our counterexample x by giving its decimal representation as follows:
  - It is a number between 0 and 1, so all its significant digits are decimals

$\mathbb{R}$ is Uncountable: Cantor's Proof

- To find the counterexample $x \in \mathbb{R}$ that leads to a contradiction, we use an example
- Suppose (again) that f exists and some of its values are:
  - $f(1) = \pi = 3.14159\ldots$
  - $f(2) = 55.55555\ldots$
  - $f(3) = 0.12345\ldots$
  - $f(4) = 0.50000\ldots$
  - $\ldots$
- Now, we construct our counterexample x by giving its decimal representation as follows:
  - It is a number between 0 and 1, so all its significant digits are decimals
  - Our goal is to ensure that $\forall n \in \mathbb{N}, x \neq f(n)$

$\mathbb{R}$ is Uncountable: Cantor's Proof

- To ensure that $x \neq f(1)$, we let the **first** digit of x to be anything different from the **first** decimal digit of $f(1)$ (i.e., 1): let it be **4**

$\mathbb{R}$ is Uncountable: Cantor's Proof

- To ensure that $x \neq f(1)$, we let the **first** digit of x to be anything different from the **first** decimal digit of $f(1)$ (i.e., 1): let it be **4**
- To ensure that $x \neq f(2)$, we let the **second** digit of x to be anything different from the **second** decimal digit of $f(2)$ (i.e., 5): let it be **6**

$\mathbb{R}$ is Uncountable: Cantor's Proof

- To ensure that $x \neq f(1)$, we let the **first** digit of x to be anything different from the **first** decimal digit of $f(1)$ (i.e., 1): let it be **4**
- To ensure that $x \neq f(2)$, we let the **second** digit of x to be anything different from the **second** decimal digit of $f(2)$ (i.e., 5): let it be **6**
- To ensure that $x \neq f(3)$, we let the **third** digit of x to be anything different from the **third** decimal digit of $f(3)$ (i.e., 3): let it be **7**

$\mathbb{R}$ is Uncountable: Cantor's Proof

- To ensure that $x \neq f(1)$, we let the **first** digit of x to be anything different from the **first** decimal digit of $f(1)$ (i.e., 1): let it be **4**
- To ensure that $x \neq f(2)$, we let the **second** digit of x to be anything different from the **second** decimal digit of $f(2)$ (i.e., 5): let it be **6**
- To ensure that $x \neq f(3)$, we let the **third** digit of x to be anything different from the **third** decimal digit of $f(3)$ (i.e., 3): let it be **7**
- We keep doing this all the way down the "diagonal" of the table for f

$\mathbb{R}$ is Uncountable: Cantor's Proof

- To ensure that $x \neq f(1)$, we let the **first** digit of x to be anything different from the **first** decimal digit of $f(1)$ (i.e., 1): let it be **4**
- To ensure that $x \neq f(2)$, we let the **second** digit of x to be anything different from the **second** decimal digit of $f(2)$ (i.e., 5): let it be **6**
- To ensure that $x \neq f(3)$, we let the **third** digit of x to be anything different from the **third** decimal digit of $f(3)$ (i.e., 3): let it be **7**
- We keep doing this all the way down the "diagonal" of the table for f
- Eventually, we build $x = \mathbf{0.467}\ldots$ which is not equal to **any** $f(n)$, as it differs from it in its n-th decimal digit

$\mathbb{R}$ is Uncountable: Implications

- The fact that $\mathbb{R}$ is uncountable has profound implications also for the theory of computation

$\mathbb{R}$ is Uncountable: Implications

- The fact that $\mathbb{R}$ is uncountable has profound implications also for the theory of computation
- Indeed, it tells us that there exist some numbers that cannot be **computed**

$\mathbb{R}$ is Uncountable: Implications

- The fact that $\mathbb{R}$ is uncountable has profound implications also for the theory of computation
- Indeed, it tells us that there exist some numbers that cannot be **computed**
- In other words, there exist some languages that are **not decidable** (**nor even recognizable**)

Table of Contents

1. Computability
2. Diagonalization
3. The Halting Problem
4. Beyond Undecidability
5. Summary

The Halting Problem

- This is one of the most fundamental problem in the theory of computation

The Halting Problem

- This is one of the most fundamental problem in the theory of computation
- Informally, it aims to find an algorithm that can tell whether a Turing machine M **halts and accepts** or not when this is given an input x

The Halting Problem

- This is one of the most fundamental problem in the theory of computation
- Informally, it aims to find an algorithm that can tell whether a Turing machine M **halts and accepts** or not when this is given an input x
- Using the same Cantor's diagonalization argument, we will prove that the halting problem is algorithmically **undecidable**:
  - There is no such an algorithm (i.e., Turing machine) that can **decide** if another Turing machine will ever halt and accept on a given input

The Halting Problem

- This is one of the most fundamental problem in the theory of computation
- Informally, it aims to find an algorithm that can tell whether a Turing machine M **halts and accepts** or not when this is given an input x
- Using the same Cantor's diagonalization argument, we will prove that the halting problem is algorithmically **undecidable**:
  - There is no such an algorithm (i.e., Turing machine) that can **decide** if another Turing machine will ever halt and accept on a given input
- This result has profound philosophical and practical implications, showing that computers are inherently limited in a fundamental way

The Halting Problem: Formal Definition (1)

Definition (The Halting Problem)

We aim to find a **total computable function** $HALT_{ACC} : \Sigma^* \mapsto \Sigma$ such that for every string representation of a Turing machine along with its input $\langle M, x \rangle \in \Sigma^*$:

$$HALT_{ACC}(\langle M, x \rangle) = \begin{cases} 1 & \text{if } M(x) = 1 \\ 0 & \text{otherwise} \end{cases}$$

The Halting Problem: Formal Definition (1)

Definition (The Halting Problem)

We aim to find a **total computable function** $HALT_{ACC} : \Sigma^* \mapsto \Sigma$ such that for every string representation of a Turing machine along with its input $\langle M, x \rangle \in \Sigma^*$:

$$HALT_{ACC}(\langle M, x \rangle) = \begin{cases} 1 & \text{if } M(x) = 1 \\ 0 & \text{otherwise} \end{cases}$$

Note

$\langle M, x \rangle$ can be thought of as the concatenation of two encodings: the Turing machine $\langle M \rangle$ and its input $\langle x \rangle$.
$HALT_{ACC}$ is the boolean function that outputs 1 if M halts (and accepts) x, i.e., $M(x) = 1$, or 0 otherwise (i.e., if $M(x) = 0$ **or** $M(x) = \perp$)

The Halting Problem: Formal Definition (2)

Definition (The Halting Problem)

Consider the **language** $A_{TM} : \{\langle M, x\rangle \in \Sigma^* \mid M \text{ is a TM and } M(x) = 1\}$.
We can also define:

$$HALT_{ACC}(\langle M, x\rangle) = \begin{cases} 1 & \text{if } \langle M, x\rangle \in A_{TM} \\ 0 & \text{if } \langle M, x\rangle \notin A_{TM} \end{cases}$$

The Halting Problem: Formal Definition (2)

Definition (The Halting Problem)

Consider the **language** $A_{TM} : \{\langle M, x\rangle \in \Sigma^* \mid M \text{ is a TM and } M(x) = 1\}$.
We can also define:

$$HALT_{ACC}(\langle M, x\rangle) = \begin{cases} 1 & \text{if } \langle M, x\rangle \in A_{TM} \\ 0 & \text{if } \langle M, x\rangle \notin A_{TM} \end{cases}$$

Note

Finding whether the **total computable function** $HALT_{ACC}$ exists is equivalent to determine whether the language A_{TM} is **decidable**

The Halting Problem: Clarification

- The actual definition of the halting problem is subtly different from the one given above

The Halting Problem: Clarification

- The actual definition of the halting problem is subtly different from the one given above
- Our definition asks the function to output 1 if and only if the input TM M halts **and** accepts its input x

The Halting Problem: Clarification

- The actual definition of the halting problem is subtly different from the one given above
- Our definition asks the function to output 1 if and only if the input TM M halts **and** accepts its input x
- However, a more accurate definition would be provided by the following $HALT$ function, which outputs 1 whenever M halts on x (no matter if it accepts or rejects):

$$HALT(\langle M, x \rangle) = \begin{cases} 1 & \text{if } M(x) = 1 \lor M(x) = 0 \\ 0 & M(x) = \perp \end{cases}$$

The Halting Problem: Clarification

- The actual definition of the halting problem is subtly different from the one given above
- Our definition asks the function to output 1 if and only if the input TM M halts **and** accepts its input x
- However, a more accurate definition would be provided by the following $HALT$ function, which outputs 1 whenever M halts on x (no matter if it accepts or rejects):

$$HALT(\langle M, x \rangle) = \begin{cases} 1 & \text{if } M(x) = 1 \vee M(x) = 0 \\ 0 & M(x) = \perp \end{cases}$$

- We will see that the two definitions are equivalent, i.e., both $HALT_{ACC}$ and $HALT$ are undecidable

The Halting Problem is not Decidable

Theorem (The Halting Problem is not Decidable)

The function $HALT_{ACC}$ as defined above is **not total and computable**. *Equivalently, the language A_{TM} is* **not decidable**.

The Halting Problem is not Decidable

Theorem (The Halting Problem is not Decidable)

The function $HALT_{ACC}$ as defined above is **not total and computable***.*
Equivalently, the language A_{TM} is **not decidable***.*

Note

Before getting into the proof, let us first observe that $HALT_{ACC}$ is **partial and computable** (i.e., A_{TM} is **recognizable**)

The Halting Problem is Recognizable

Theorem (The Halting Problem is Recognizable)

The following Turing machine U computes a partial function $HALT_{ACC}$ (i.e., recognizes A_{TM}):

U = On input $\langle M, x \rangle$, where M is a TM and x an input to it:

1. *Simulate M on input x;*
2. *If M ever halts and* accepts *x (i.e., $M(x) = 1$) then return 1; if M ever halts and* rejects *x (i.e., $M(x) = 0$) then return 0*

The Halting Problem is Recognizable

Theorem (The Halting Problem is Recognizable)

The following Turing machine U computes a partial function $HALT_{ACC}$ (i.e., recognizes A_{TM}):

$U =$ *On input $\langle M, x\rangle$, where M is a TM and x an input to it:*

1. *Simulate M on input x;*
2. *If M ever halts and* accepts *x (i.e., $M(x) = 1$) then return* 1*; if M ever halts and* rejects *x (i.e., $M(x) = 0$) then return* 0

Note

The Turing machine U above loops on input $\langle M, x\rangle$ if M loops on x.
This is why U computes a partial but **not** a total function (i.e., U recognizes but **not** decides A_{TM})

The Halting Problem: Considerations

- The halting problem is not just a purely theoretical exercise!

The Halting Problem: Considerations

- The halting problem is not just a purely theoretical exercise!
- For example, think about managing the Apple's App Store or Google's Play Store: given some app code, you would like to know whether this gets into an infinite loop!

The Halting Problem: Considerations

- The halting problem is not just a purely theoretical exercise!
- For example, think about managing the Apple's App Store or Google's Play Store: given some app code, you would like to know whether this gets into an infinite loop!
- If U had some way to determine that M would loop forever on x, it could output 0

The Halting Problem: Considerations

- The halting problem is not just a purely theoretical exercise!
- For example, think about managing the Apple's App Store or Google's Play Store: given some app code, you would like to know whether this gets into an infinite loop!
- If U had some way to determine that M would loop forever on x, it could output 0
- Unfortunately, no algorithm exists that can make this determination **for every possible TMs and their inputs**

The Halting Problem is not Decidable

Theorem (The Halting Problem is not Decidable)

Consider the **language** $A_{TM} : \{\langle M, x\rangle \in \Sigma^* \mid M \text{ is a TM and } M(x) = 1\}$.
We can also define:

$$HALT_{ACC}(\langle M, x\rangle) = \begin{cases} 1 & \text{if } \langle M, x\rangle \in A_{TM} \\ 0 & \text{if } \langle M, x\rangle \notin A_{TM} \end{cases}$$

The function $HALT_{ACC}$ as defined above is **not total and computable**.
Equivalently, the language A_{TM} is **not decidable**.

The Halting Problem is not Decidable: Proof

- We assume that A_{TM} is decidable and obtain a contradiction

The Halting Problem is not Decidable: Proof

- We assume that A_{TM} is decidable and obtain a contradiction
- Let H be a decider for A_{TM}, such that on input $\langle M, x \rangle$, where M is a TM and x is a (binary) string:
  - H **halts and outputs 1** if M outputs 1 on x
  - H **halts and outputs 0** if M fails to output 1 on x (i.e., either outputs 0 or loop forever)

The Halting Problem is not Decidable: Proof

- We assume that A_{TM} is decidable and obtain a contradiction
- Let H be a decider for A_{TM}, such that on input $\langle M, x\rangle$, where M is a TM and x is a (binary) string:
  - H **halts and outputs 1** if M outputs 1 on x
  - H **halts and outputs 0** if M fails to output 1 on x (i.e., either outputs 0 or loop forever)
- In other words:

$$H(\langle M, x\rangle) = \begin{cases} 1 & \text{if } M(x) = 1 \\ 0 & \text{if } M(x) = 0 \vee M(x) = \perp \end{cases}$$

The Halting Problem is not Decidable: Proof

- Let's construct another TM called D, which uses H as a subroutine

The Halting Problem is not Decidable: Proof

- Let's construct another TM called D, which uses H as a subroutine
- This new TM calls H to determine what M does when the input to M is its own description $\langle M \rangle$

The Halting Problem is not Decidable: Proof

- Let's construct another TM called D, which uses H as a subroutine
- This new TM calls H to determine what M does when the input to M is its own description $\langle M \rangle$
- Once D has determined this information, it does the **opposite**, i.e., it outputs 0 if $H(\langle M, \langle M \rangle \rangle) = 1$ or 1 if $H(\langle M, \langle M \rangle \rangle) = 0$

$$D(\langle M \rangle) = \begin{cases} 1 & \text{if } H(\langle M, \langle M \rangle \rangle) = 0 \Leftrightarrow M(\langle M \rangle) = 0 \vee M(\langle M \rangle) = \perp \\ 0 & \text{if } H(\langle M, \langle M \rangle \rangle) = 1 \Leftrightarrow M(\langle M \rangle) = 1 \end{cases}$$

The Halting Problem is not Decidable: Proof

What happens if we run D with its own description $\langle D \rangle$ as input?

$$D(\langle D \rangle) = \begin{cases} 1 & \text{if } H(\langle D, \langle D \rangle \rangle) = 0 \Leftrightarrow D(\langle D \rangle) = 0 \vee D(\langle D \rangle) = \perp \\ 0 & \text{if } H(\langle D, \langle D \rangle \rangle) = 1 \Leftrightarrow D(\langle D \rangle) = 1 \end{cases}$$

The Halting Problem is not Decidable: Proof

What happens if we run D with its own description $\langle D\rangle$ as input?

$$D(\langle D\rangle) = \begin{cases} 1 & \text{if } H(\langle D, \langle D\rangle\rangle) = 0 \Leftrightarrow D(\langle D\rangle) = 0 \vee D(\langle D\rangle) = \perp \\ 0 & \text{if } H(\langle D, \langle D\rangle\rangle) = 1 \Leftrightarrow D(\langle D\rangle) = 1 \end{cases}$$

Note

No matter what D does, it is forced to do the opposite, which leads us to a contradiction! As such, neither D nor H can exist!

The Halting Problem is not Decidable: Proof

- Where is the **diagonalization argument** in this proof?

The Halting Problem is not Decidable: Proof

- Where is the **diagonalization argument** in this proof?
- We must examine the tables of behavior for the two TMs used: H and D

The Halting Problem is not Decidable: Proof

- Where is the **diagonalization argument** in this proof?
- We must examine the tables of behavior for the two TMs used: H and D
- We start first by showing a more general table of behavior of **all** TMs when they are input with the description of some (other) TM

The Halting Problem is not Decidable: Proof

- Where is the **diagonalization argument** in this proof?
- We must examine the tables of behavior for the two TMs used: H and D
- We start first by showing a more general table of behavior of **all** TMs when they are input with the description of some (other) TM
- In other words, given a generic TM M_i the table shows the behavior of M_i when this is input with $\langle M_j \rangle$

The Halting Problem is not Decidable: Proof

- Where is the **diagonalization argument** in this proof?
- We must examine the tables of behavior for the two TMs used: H and D
- We start first by showing a more general table of behavior of **all** TMs when they are input with the description of some (other) TM
- In other words, given a generic TM M_i the table shows the behavior of M_i when this is input with $\langle M_j \rangle$
- Without loss of generality, the output of M_i when input with $\langle M_j \rangle$ is either: 1 (*accept*), 0 (*reject*), or $\perp$ (*does not halt*)

The Behavior of All Turing Machines

	$<M_1>$	$<M_2>$	$<M_3>$	$<M_4>$	…	$<M_k>$	…
M_1	accept	reject	accept	doesn't halt	…	reject	…
M_2	accept	accept	accept	accept	…	doesn't halt	…
M_3	reject	doesn't halt	doesn't halt	reject	…	accept	…
M_4	accept	accept	reject	reject	…	doesn't halt	…
…	…	…	…	…	…	…	…
M_k	doesn't halt	reject	accept	accept		accept	…
…	…	…	…	…	…	…	…

Entry i, j contains the output of TM M_i on input $\langle M_j \rangle$

The Behavior of H

The table below shows the behavior of TM H on inputs from the table above: if M_i accepts $\langle M_j \rangle$ so does H, if M_i either rejects or doesn't halt on $\langle M_j \rangle$ then H rejects

The Behavior of H

The table below shows the behavior of TM H on inputs from the table above: if M_i accepts $\langle M_j \rangle$ so does H, if M_i either rejects or doesn't halt on $\langle M_j \rangle$ then H rejects

	<M_1>	<M_2>	<M_3>	<M_4>	…	<M_k>	…
M_1	accept	reject	accept	reject	…	reject	…
M_2	accept	accept	accept	accept	…	reject	…
M_3	reject	reject	reject	reject	…	accept	…
M_4	accept	accept	reject	reject	…	reject	…
…	…	…	…	…	…	…	…
M_k	reject	reject	accept	accept		accept	…
…	…	…	…	…	…	…	…

The Behavior of D

Assuming H exists, so does D! As such, D must be located somewhere in the list of **all** TMs

The Behavior of *D*

Assuming *H* exists, so does *D*! As such, *D* must be located somewhere in the list of **all** TMs

	<M_1>	<M_2>	<M_3>	<M_4>	…	<D>	…
M_1	<u>accept</u>	reject	accept	reject	…	reject	…
M_2	accept	<u>accept</u>	accept	accept	…	reject	…
M_3	reject	reject	<u>reject</u>	reject	…	accept	…
M_4	accept	accept	reject	<u>reject</u>	…	reject	…
…	…	…	…	…	…	…	…
D	reject	reject	accept	accept		?	…
…	…	…	…	…	…	…	…

The Behavior of D

Since D computes the opposite of the diagonal entries, a contradiction occurs in correspondence of column $\langle D \rangle$, where the entry must be the opposite of itself!

	<M1>	<M2>	<M3>	<M4>	…	<D>	…
M1	**accept**	reject	accept	reject	…	reject	…
M2	accept	**accept**	accept	accept	…	reject	…
M3	reject	reject	**reject**	reject	…	accept	…
M4	accept	accept	reject	**reject**	…	reject	…
…	…	…	…	…	…	…	…
D	reject	reject	accept	accept		?	…
…	…	…	…	…	…	…	…

Table of Contents

1. Computability
2. Diagonalization
3. The Halting Problem
4. Beyond Undecidability
5. Summary

Turing-unrecognizable Languages

- We have shown that the halting problem (i.e., the language it defines A_{TM}) is **undecidable** (or semi-decidable)

Turing-unrecognizable Languages

- We have shown that the halting problem (i.e., the language it defines A_{TM}) is **undecidable** (or semi-decidable)
- We know that A_{TM} is **recognizable**

Turing-unrecognizable Languages

- We have shown that the halting problem (i.e., the language it defines A_{TM}) is **undecidable** (or semi-decidable)
- We know that A_{TM} is **recognizable**
- However, there exist languages that are not even recognized by a TM

Turing-unrecognizable Languages

- We have shown that the halting problem (i.e., the language it defines A_{TM}) is **undecidable** (or semi-decidable)
- We know that A_{TM} is **recognizable**
- However, there exist languages that are not even recognized by a TM
- We will prove that using the same diagonalization argument

Turing-unrecognizable Languages

Corollary (Turing-unrecognizable Languages)

- *There are* **countably** *many Turing machines*

Turing-unrecognizable Languages

Corollary (Turing-unrecognizable Languages)

- *There are* **countably** *many Turing machines*
- *There are* **uncountably** *many languages*

Turing-unrecognizable Languages

Corollary (Turing-unrecognizable Languages)

- *There are* **countably** *many Turing machines*
- *There are* **uncountably** *many languages*
- *Each Turing machine can* **recognize** *a single language*

Turing-unrecognizable Languages

Corollary (Turing-unrecognizable Languages)

- *There are* **countably** *many Turing machines*
- *There are* **uncountably** *many languages*
- *Each Turing machine can* **recognize** *a single language*

$$\Downarrow$$

Some languages are not Turing-recognizable

Turing-unrecognizable Languages: Proof

- To prove the corollary above we need to show that:

Turing-unrecognizable Languages: Proof

- To prove the corollary above we need to show that:
  1. The set of **all** Turing machines is **countable**

Turing-unrecognizable Languages: Proof

- To prove the corollary above we need to show that:
  1. The set of **all** Turing machines is **countable**
  2. The set of **all** languages is **uncountable**

Turing-unrecognizable Languages: Proof

- To prove the corollary above we need to show that:
  1. The set of **all** Turing machines is **countable**
  2. The set of **all** languages is **uncountable**
- Let's start with 1!

1) The Set of All TMs is Countable

- We first observe that for any finite alphabet Σ the (infinite) set of all the finite-length strings Σ^* is countable

1) The Set of All TMs is Countable

- We first observe that for any finite alphabet Σ the (infinite) set of all the finite-length strings Σ^* is countable
- We can indeed make a list of all the elements of Σ^* by noticing that there are finitely many strings of each specific length:
  - We enumerate all the strings of length 0, then those of length 1, length 2, etc.

1) The Set of All TMs is Countable

- We first observe that for any finite alphabet Σ the (infinite) set of all the finite-length strings Σ^* is countable
- We can indeed make a list of all the elements of Σ^* by noticing that there are finitely many strings of each specific length:
  - We enumerate all the strings of length 0, then those of length 1, length 2, etc.
- The set of all Turing machines is countable because each TM has a finite (binary) string encoding $\langle M \rangle \in \Sigma^*$, where $\Sigma = \{0, 1\}$

1) The Set of All TMs is Countable

- We first observe that for any finite alphabet Σ the (infinite) set of all the finite-length strings Σ^* is countable
- We can indeed make a list of all the elements of Σ^* by noticing that there are finitely many strings of each specific length:
  - We enumerate all the strings of length 0, then those of length 1, length 2, etc.
- The set of all Turing machines is countable because each TM has a finite (binary) string encoding $\langle M \rangle \in \Sigma^*$, where $\Sigma = \{0, 1\}$
- Therefore, we can list all those encodings $\langle M_1 \rangle, \langle M_2 \rangle, \ldots$ as above

1) The Set of All TMs is Countable

- We first observe that for any finite alphabet Σ the (infinite) set of all the finite-length strings Σ^* is countable
- We can indeed make a list of all the elements of Σ^* by noticing that there are finitely many strings of each specific length:
  - We enumerate all the strings of length 0, then those of length 1, length 2, etc.
- The set of all Turing machines is countable because each TM has a finite (binary) string encoding $\langle M \rangle \in \Sigma^*$, where $\Sigma = \{0, 1\}$
- Therefore, we can list all those encodings $\langle M_1 \rangle, \langle M_2 \rangle, \ldots$ as above
- In other words, we can easily find a correspondence between the set of natural numbers $\mathbb{N}$ and the (infinite) set of **all** Turing machines (encoded as binary strings)

2) The Set of All Languages is Uncountable

- Two approaches to show that this statement is true:
  1. using the countable set of all the finite-length binary strings and Cantor's diagonalization
  2. using the uncountable set of all the infinite-length binary strings

2) The Set of All Languages is Uncountable

- Two approaches to show that this statement is true:
  1. using the countable set of all the finite-length binary strings and Cantor's diagonalization
  2. using the uncountable set of all the infinite-length binary strings
- Let's start from 1!

2) The Set of All Languages is Uncountable: First Proof

- Let $\Sigma^* = \{\sigma_1, \sigma_2, \ldots\}$ be the set of all the finite-length binary strings

2) The Set of All Languages is Uncountable: First Proof

- Let $\Sigma^* = \{\sigma_1, \sigma_2, \ldots\}$ be the set of all the finite-length binary strings
- We already shown that Σ^* is **countable**

2) The Set of All Languages is Uncountable: First Proof

- Let $\Sigma^* = \{\sigma_1, \sigma_2, \ldots\}$ be the set of all the finite-length binary strings
- We already shown that Σ^* is **countable**
- Any language L can be seen as a subset of Σ^*

2) The Set of All Languages is Uncountable: First Proof

- Let $\Sigma^* = \{\sigma_1, \sigma_2, \ldots\}$ be the set of all the finite-length binary strings
- We already shown that Σ^* is **countable**
- Any language L can be seen as a subset of Σ^*
- We must show that the set $\mathcal{L}$ of **all** languages is **uncountable**

2) The Set of All Languages is Uncountable: First Proof

- Suppose that $\mathcal{L}$ is, instead, countable

2) The Set of All Languages is Uncountable: First Proof

- Suppose that $\mathcal{L}$ is, instead, countable
- Therefore, there exists a correspondence $f : \mathbb{N} \mapsto \mathcal{L}$

2) The Set of All Languages is Uncountable: First Proof

- Suppose that $\mathcal{L}$ is, instead, countable
- Therefore, there exists a correspondence $f : \mathbb{N} \mapsto \mathcal{L}$
- We can define $f(k)$ as the k-th element of $\mathcal{L}$, denoted by L_k (**Remember:** each element of $\mathcal{L}$ is actually a subset of Σ^*)

2) The Set of All Languages is Uncountable: First Proof

- Suppose that $\mathcal{L}$ is, instead, countable
- Therefore, there exists a correspondence $f : \mathbb{N} \mapsto \mathcal{L}$
- We can define $f(k)$ as the k-th element of $\mathcal{L}$, denoted by L_k (**Remember:** each element of $\mathcal{L}$ is actually a subset of Σ^*)
- Now, we apply Cantor's diagonalization to build a new element $L_j \in \mathcal{L}$ as follows:
  - If $\sigma_k \in L_k$ then $\sigma_k \notin L_j$
  - If $\sigma_k \notin L_k$ then $\sigma_k \in L_j$

2) The Set of All Languages is Uncountable: First Proof

- Suppose that $\mathcal{L}$ is, instead, countable
- Therefore, there exists a correspondence $f : \mathbb{N} \mapsto \mathcal{L}$
- We can define $f(k)$ as the k-th element of $\mathcal{L}$, denoted by L_k (**Remember:** each element of $\mathcal{L}$ is actually a subset of Σ^*)
- Now, we apply Cantor's diagonalization to build a new element $L_j \in \mathcal{L}$ as follows:
  - If $\sigma_k \in L_k$ then $\sigma_k \notin L_j$
  - If $\sigma_k \notin L_k$ then $\sigma_k \in L_j$
- Then, we have found an element $L_j \in \mathcal{L}$, such that there is no $j \in \mathbb{N}, f(j) = L_j$

2) The Set of All Languages is Uncountable: First Proof

- To find the counterexample $L_j \in \mathcal{L}$ that leads to a contradiction, let us assume that f exists and some of its values are:
  - $f(1) = L_1 = \{\epsilon, 010, 11101, \ldots\}$
  - $f(2) = L_2 = \{1, 00, 101, 00010, \ldots\}$
  - $f(3) = L_3 = \{10, 11, 000, 10101, \ldots\}$
  - $f(4) = L_4 = \{\epsilon, 1, 00, 01010, \ldots\}$
  - $\ldots$

2) The Set of All Languages is Uncountable: First Proof

- To find the counterexample $L_j \in \mathcal{L}$ that leads to a contradiction, let us assume that f exists and some of its values are:
  - $f(1) = L_1 = \{\epsilon, 010, 11101, \ldots\}$
  - $f(2) = L_2 = \{1, 00, 101, 00010, \ldots\}$
  - $f(3) = L_3 = \{10, 11, 000, 10101, \ldots\}$
  - $f(4) = L_4 = \{\epsilon, 1, 00, 01010, \ldots\}$
  - $\ldots$
- Suppose also that $\sigma_1 = \epsilon$, $\sigma_2 = 0$, $\sigma_3 = 1$, $\sigma_4 = 00, \ldots$

2) The Set of All Languages is Uncountable: First Proof

- To find the counterexample $L_j \in \mathcal{L}$ that leads to a contradiction, let us assume that f exists and some of its values are:
  - $f(1) = L_1 = \{\epsilon, 010, 11101, \ldots\}$
  - $f(2) = L_2 = \{1, 00, 101, 00010, \ldots\}$
  - $f(3) = L_3 = \{10, 11, 000, 10101, \ldots\}$
  - $f(4) = L_4 = \{\epsilon, 1, 00, 01010, \ldots\}$
  - $\ldots$
- Suppose also that $\sigma_1 = \epsilon$, $\sigma_2 = 0$, $\sigma_3 = 1$, $\sigma_4 = 00, \ldots$
- σ_i will be included in L_j iff $\sigma_i \notin f(i) = L_i$

2) The Set of All Languages is Uncountable: First Proof

- To find the counterexample $L_j \in \mathcal{L}$ that leads to a contradiction, let us assume that f exists and some of its values are:
  - $f(1) = L_1 = \{\epsilon, 010, 11101, \ldots\}$
  - $f(2) = L_2 = \{1, 00, 101, 00010, \ldots\}$
  - $f(3) = L_3 = \{10, 11, 000, 10101, \ldots\}$
  - $f(4) = L_4 = \{\epsilon, 1, 00, 01010, \ldots\}$
  - $\ldots$
- $\sigma_1 = \epsilon \notin L_j$ as $\sigma_1 \in L_1$, therefore $L_j = \{\} \neq f(1)$
- $\sigma_2 = 0 \in L_j$ as $\sigma_2 \notin L_2$, therefore $L_j = \{0\} \neq f(2)$
- $\sigma_3 = 1 \in L_j$ as $\sigma_3 \notin L_3$, therefore $L_j = \{0, 1\} \neq f(3)$
- $\sigma_4 = 00 \notin L_j$ as $\sigma_4 \in L_4$, therefore $L_j = \{0, 1\} \neq f(4)$
- $\ldots$

2) The Set of All Languages is Uncountable: Second Proof

- In this second proof, instead, we start from a different perspective

2) The Set of All Languages is Uncountable: Second Proof

- In this second proof, instead, we start from a different perspective
- We first observe that the set of all **infinite** binary sequences is uncountable

2) The Set of All Languages is Uncountable: Second Proof

- In this second proof, instead, we start from a different perspective
- We first observe that the set of all **infinite** binary sequences is uncountable
- An infinite binary sequence is a never-ending sequence of binary symbols (0s and 1s)

2) The Set of All Languages is Uncountable: Second Proof

- In this second proof, instead, we start from a different perspective
- We first observe that the set of all **infinite** binary sequences is uncountable
- An infinite binary sequence is a never-ending sequence of binary symbols (0s and 1s)
- Let $\mathcal{B}$ be the set of all such infinite binary sequences: we can show $\mathcal{B}$ is uncountable using Cantor's diagonalization

Note

The infinite set of finite-length binary strings $\Sigma^* = \{0, 1\}^*$ is **not equal** to the set of infinite-length binary strings $\mathcal{B}$

2) The Set of All Languages is Uncountable: Second Proof

- To show that $\mathcal{B}$ is uncountable we assume it is, in fact, countable

2) The Set of All Languages is Uncountable: Second Proof

- To show that $\mathcal{B}$ is uncountable we assume it is, in fact, countable
- If that is the case, we will be able to list all the infinite-length binary sequences

2) The Set of All Languages is Uncountable: Second Proof

- To show that $\mathcal{B}$ is uncountable we assume it is, in fact, countable
- If that is the case, we will be able to list all the infinite-length binary sequences
- In other words, we can create a correspondence $f : \mathbb{N} \mapsto \mathcal{B}$, e.g.:
  - $f(1) = 00000\ldots$
  - $f(2) = 11010\ldots$
  - $f(3) = 00101\ldots$
  - $\ldots$

2) The Set of All Languages is Uncountable: Second Proof

- To show that $\mathcal{B}$ is uncountable we assume it is, in fact, countable
- If that is the case, we will be able to list all the infinite-length binary sequences
- In other words, we can create a correspondence $f : \mathbb{N} \mapsto \mathcal{B}$, e.g.:
  - $f(1) = 00000\ldots$
  - $f(2) = 11010\ldots$
  - $f(3) = 00101\ldots$
  - $\ldots$
- However, we can build another binary sequence b such that its i-th bit is the opposite of the i-th bit of each $f(i)$. In the example above: $b = 100\ldots$

2) The Set of All Languages is Uncountable: Second Proof

- To show that $\mathcal{B}$ is uncountable we assume it is, in fact, countable
- If that is the case, we will be able to list all the infinite-length binary sequences
- In other words, we can create a correspondence $f : \mathbb{N} \mapsto \mathcal{B}$, e.g.:
  - $f(1) = 00000\ldots$
  - $f(2) = 11010\ldots$
  - $f(3) = 00101\ldots$
  - $\ldots$
- However, we can build another binary sequence b such that its i-th bit is the opposite of the i-th bit of each $f(i)$. In the example above: $b = 100\ldots$
- We have found a sequence b which is not paired with any natural number listed $\Longrightarrow \mathcal{B}$ is **uncountable**

2) The Set of All Languages is Uncountable: Second Proof

- So far we have shown that $\mathcal{B}$ is uncountable

2) The Set of All Languages is Uncountable: Second Proof

- So far we have shown that $\mathcal{B}$ is uncountable
- Now we want to show that the set $\mathcal{L}$ all languages over a finite alphabet Σ is uncountable as well

2) The Set of All Languages is Uncountable: Second Proof

- So far we have shown that $\mathcal{B}$ is uncountable
- Now we want to show that the set $\mathcal{L}$ all languages over a finite alphabet Σ is uncountable as well
- To do so, we build a correspondence $f : \mathcal{L} \mapsto \mathcal{B}$, thus showing they both have the same size

2) The Set of All Languages is Uncountable: Second Proof

- So far we have shown that $\mathcal{B}$ is uncountable
- Now we want to show that the set $\mathcal{L}$ all languages over a finite alphabet Σ is uncountable as well
- To do so, we build a correspondence $f : \mathcal{L} \mapsto \mathcal{B}$, thus showing they both have the same size
- Let $\Sigma^* = \{\sigma_1, \sigma_2, \ldots\}$ the list of finite-length strings over Σ (e.g., $\Sigma = \{0, 1\}$)

2) The Set of All Languages is Uncountable: Second Proof

- So far we have shown that $\mathcal{B}$ is uncountable
- Now we want to show that the set $\mathcal{L}$ all languages over a finite alphabet Σ is uncountable as well
- To do so, we build a correspondence $f : \mathcal{L} \mapsto \mathcal{B}$, thus showing they both have the same size
- Let $\Sigma^* = \{\sigma_1, \sigma_2, \ldots\}$ the list of finite-length strings over Σ (e.g., $\Sigma = \{0, 1\}$)
- Each language $L \in \mathcal{L}$ can be described as a unique sequence in $\mathcal{B}$, where the i-th bit of that sequence is 1 if $\sigma_i \in L$, or 0 if $\sigma_i \notin L$

Note

Without loss of generality, each language $L \in \mathcal{L}$ is indeed composed of infinitely many finite-length strings, therefore represented as an infinite binary sequence

2) The Set of All Languages is Uncountable: Second Proof

Definition (Characteristic Sequence)

We call the infinite binary sequence associated with each $L \in \mathcal{L}$ the **characteristic sequence** of L, denoted by χ_L

2) The Set of All Languages is Uncountable: Second Proof

Definition (Characteristic Sequence)

We call the infinite binary sequence associated with each $L \in \mathcal{L}$ the **characteristic sequence** of L, denoted by χ_L

Example

Suppose L is the language of all strings starting with a 0 over the alphabet $\Sigma = \{0, 1\}$. Then, its **characteristic sequence** χ_L will be as follows:

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

$$L = \{0, 00, 01, 000, 001, \ldots\}$$

$$\chi_L = \{0, 1, 0, 1, 1, 0, 0, 1, 1, \ldots\}$$

2) The Set of All Languages is Uncountable: Second Proof

- The function $f : \mathcal{L} \mapsto \mathcal{B}$, where $f(L) = \chi_L$ equals the characteristic sequence of L is one-to-one and onto, thus a correspondence

2) The Set of All Languages is Uncountable: Second Proof

- The function $f : \mathcal{L} \mapsto \mathcal{B}$, where $f(L) = \chi_L$ equals the characteristic sequence of L is one-to-one and onto, thus a correspondence
- Given that $\mathcal{B}$ is uncountable and we are able to put it into a correspondence with $\mathcal{L}$ then $\mathcal{L}$ is **uncountable** as well

2) The Set of All Languages is Uncountable: Second Proof

- The function $f : \mathcal{L} \mapsto \mathcal{B}$, where $f(L) = \chi_L$ equals the characteristic sequence of L is one-to-one and onto, thus a correspondence
- Given that $\mathcal{B}$ is uncountable and we are able to put it into a correspondence with $\mathcal{L}$ then $\mathcal{L}$ is **uncountable** as well
- We have shown (again) that the set of all languages $\mathcal{L}$ cannnot be put into correspondence with the set of all Turing machines

2) The Set of All Languages is Uncountable: Second Proof

- The function $f : \mathcal{L} \mapsto \mathcal{B}$, where $f(L) = \chi_L$ equals the characteristic sequence of L is one-to-one and onto, thus a correspondence
- Given that $\mathcal{B}$ is uncountable and we are able to put it into a correspondence with $\mathcal{L}$ then $\mathcal{L}$ is **uncountable** as well
- We have shown (again) that the set of all languages $\mathcal{L}$ cannnot be put into correspondence with the set of all Turing machines
- We can conclude that some languages are **not** recognized by any Turing machine

The Set of All Languages is Uncountable: Extra Proof

Theorem (Cantor's Theorem)

For **any** *set A, the set of all subsets of A - also known as the* **power set** *of A, denoted by $\mathcal{P}(A)$ - has a strictly greater cardinality than A itself:*

$$|\mathcal{P}(A)| > |A|$$

The Set of All Languages is Uncountable: Extra Proof

Theorem (Cantor's Theorem)

*For **any** set A, the set of all subsets of A - also known as the **power set** of A, denoted by $\mathcal{P}(A)$ - has a strictly greater cardinality than A itself:*

$$|\mathcal{P}(A)| > |A|$$

Example (Countably Finite Sets)

If A is countable and finite, the relation trivially holds:
Let $|A| = n$, then $|\mathcal{P}(A)| = 2^n$

SAPIENZA UNIVERSITÀ DI ROMA

The Set of All Languages is Uncountable: Extra Proof

Theorem (Cantor's Theorem)

*For **any** set A, the set of all subsets of A - also known as the **power set** of A, denoted by $\mathcal{P}(A)$ - has a strictly greater cardinality than A itself:*

$$|\mathcal{P}(A)| > |A|$$

Example (Countably Infinite Sets)

If A is countable and infinite, the relation still holds:
Let $A = \mathbb{N}$, then $|\mathbb{N}| = \aleph_0$ and $|\mathcal{P}(\mathbb{N})| = 2^{\aleph_0}$
In particular, the power set of the set of natural numbers is uncountably infinite and has the same size as the set of real numbers, whose cardinality $\mathfrak{c} = 2^{\aleph_0}$ is referred to as the **cardinality of the continuum**:

$$|\mathcal{P}(\mathbb{N})| = |\mathbb{R}| = \mathfrak{c} = 2^{\aleph_0} > \aleph_0 = |\mathbb{N}|$$

Table of Contents

1. Computability
2. Diagonalization
3. The Halting Problem
4. Beyond Undecidability
5. Summary

Summary

- We ask ourselves whether there exists any problem that is not algorithmically solvable

Summary

- We ask ourselves whether there exists any problem that is not algorithmically solvable
- We consider as a reference example the well-known **Halting Problem**

Summary

- We ask ourselves whether there exists any problem that is not algorithmically solvable
- We consider as a reference example the well-known **Halting Problem**
- We proved the Halting Problem is **undecidable** (or semi-decidable), although it is of course **recognizable**

Summary

- We ask ourselves whether there exists any problem that is not algorithmically solvable
- We consider as a reference example the well-known **Halting Problem**
- We proved the Halting Problem is **undecidable** (or semi-decidable), although it is of course **recognizable**
- The proof is based on Cantor's diagonalization argument used to compare the size of (infinite) sets

Summary

- We ask ourselves whether there exists any problem that is not algorithmically solvable
- We consider as a reference example the well-known **Halting Problem**
- We proved the Halting Problem is **undecidable** (or semi-decidable), although it is of course **recognizable**
- The proof is based on Cantor's diagonalization argument used to compare the size of (infinite) sets
- Diagonalization allows us to prove that there are functions/languages that cannot be even recognized (i.e., we have countably many TMs yet uncountably many languages)