The Diagonalization Method

November 18, 2015

Decidability of TM language

Problem: For M a Turing machine and w a string, does M accept w?

Decidability of TM language

Problem: For M a Turing machine and w a string, does M accept w?

Language: $A_{TM} = \{\langle M, w\rangle | M \text{ is a TM, } w \text{ is a string, } M \text{ accepts } w\}$

Theorem 4.11

A_{TM} is recognizable but not decidable

A recognizer of A_{TM} is the following TM called the Turing Universal Machine U:

$U =$ "On input $\langle M, w \rangle$, where M is a TM and w is a string:

Theorem 4.11

A_{TM} is recognizable but not decidable

A recognizer of A_{TM} is the following TM called the Turing Universal Machine U:

$U =$ "On input $\langle M, w \rangle$, where M is a TM and w is a string:

1. Simulates M on w.

Theorem 4.11

A_{TM} is recognizable but not decidable

A recognizer of A_{TM} is the following TM called the Turing Universal Machine U:

$U =$ "On input $\langle M, w \rangle$, where M is a TM and w is a string:

1. Simulates M on w.
2. If M ever enters its accept state, **accept**; if M ever enters its reject state, **reject**."

Theorem 4.11

A_{TM} is recognizable but not decidable

A recognizer of A_{TM} is the following TM called the Turing Universal Machine U:

$U =$ "On input $\langle M, w \rangle$, where M is a TM and w is a string:

1. Simulates M on w.
2. If M ever enters its accept state, **accept**; if M ever enters its reject state, **reject**."

Note: U is universal because it simulates any other TM from its description.

Note

- So far we have tackled only solvable (decidable) problems
- Theorem 4.11 states that A_{TM} is unsolvable (undecidable)
- Since A_{TM} is undecidable, to solve this problem we need to expand our problem solving methodology by a new method for proving undecidability.

Methodology (review)

To solve decidability problems concerning relations between languages one should proceed as follows:

Methodology (review)

To solve decidability problems concerning relations between languages one should proceed as follows:

- Understand the relationship

Methodology (review)

To solve decidability problems concerning relations between languages one should proceed as follows:

- Understand the relationship
- Transform the relationship into an expression using closure operators on decidable languages

Methodology (review)

To solve decidability problems concerning relations between languages one should proceed as follows:

- Understand the relationship
- Transform the relationship into an expression using closure operators on decidable languages
- Design a TM that constructs the language thus expressed

Methodology (review)

To solve decidability problems concerning relations between languages one should proceed as follows:

- Understand the relationship
- Transform the relationship into an expression using closure operators on decidable languages
- Design a TM that constructs the language thus expressed
- Run a TM that decide the language represented by the expression

Diagonalization

- The proof of undecidability of the halting problem uses Georg Cantor (1873) technique called **diagonalization**

Diagonalization

- The proof of undecidability of the halting problem uses Georg Cantor (1873) technique called **diagonalization**
- Cantor's problem was to measure the size of infinite sets

Diagonalization

- The proof of undecidability of the halting problem uses Georg Cantor (1873) technique called **diagonalization**
- Cantor's problem was to measure the size of infinite sets
- The size of finite sets is measured by counting the number of their elements.

Diagonalization

- The proof of undecidability of the halting problem uses Georg Cantor (1873) technique called **diagonalization**
- Cantor's problem was to measure the size of infinite sets
- The size of finite sets is measured by counting the number of their elements.

  Question: could we use the same method to measure the size of infinite sets?

Note

The size of infinite sets cannot be measured by counting their elements because this procedure does not halt

Example infinite sets

- The set of strings over $\{0, 1\}$ is an infinite set
- The set $\mathcal{N}$ of natural number is also an infinite set
- Both of them are larger than any finite set.

  How can we compare them?

Cantor's solution

- Two finite sets have the same size if their elements can be paired
- Since this method do not rely on counting elements it can be used for both finite and infinite sets

The correspondence

Consider two sets, A and B and $f : A \to B$ a function

The correspondence

Consider two sets, A and B and $f : A \to B$ a function

- f is **one-to-one** if it never maps two different elements of A into the same element of B, i.e., $\forall a, b \in A,\ a \neq b \Rightarrow f(a) \neq f(b)$.

The correspondence

Consider two sets, A and B and $f : A \rightarrow B$ a function

- f is **one-to-one** if it never maps two different elements of A into the same element of B, i.e., $\forall a, b \in A,\ a \neq b \Rightarrow f(a) \neq f(b)$.
- f is **onto** if it hits every element of B, i.e., $\forall b \in B$, $\exists a \in A$ such that $f(a) = b$

The correspondence

Consider two sets, A and B and $f : A \rightarrow B$ a function

- f is **one-to-one** if it never maps two different elements of A into the same element of B, i.e., $\forall a, b \in A,\ a \neq b \Rightarrow f(a) \neq f(b)$.
- f is **onto** if it hits every element of B, i.e., $\forall b \in B$, $\exists a \in A$ such that $f(a) = b$
- f is called a **correspondence** if it is both **one-to-one** and **onto**

Size comparison

Two sets A and B have the same size if there is a correspondence $F : A \to B$

Example correspondences

- Let $\mathcal{N}$ be the set of natural numbers, $\mathcal{N} = \{1, 2, 3, \ldots\}$ and $\mathcal{E}$ the set of even natural numbers, $\mathcal{E} = \{2, 4, 6, \ldots\}$
- Intuitively one may believe that $size(\mathcal{N}) > size(\mathcal{E})$. However, using Cantor method we can show that $\mathcal{N}$ and $\mathcal{E}$ have the same size by constructing the correspondence $f : \mathcal{N} \to \mathcal{E}$

Example correspondences

- Let $\mathcal{N}$ be the set of natural numbers, $\mathcal{N} = \{1, 2, 3, \ldots\}$ and $\mathcal{E}$ the set of even natural numbers, $\mathcal{E} = \{2, 4, 6, \ldots\}$
- Intuitively one may believe that $size(\mathcal{N}) > size(\mathcal{E})$. However, using Cantor method we can show that $\mathcal{N}$ and $\mathcal{E}$ have the same size by constructing the correspondence $f : \mathcal{N} \rightarrow \mathcal{E}$
- This correspondence is defined by $f(n) = 2n$, Figure 1.

n	$f(n)$
1	2
2	4
3	6
...	...

Figure 1 : $sizeof(\mathcal{N}) = sizeof(\mathcal{E})$

Definition 4.14

A set is countable if either it is finite or it has the same size as $\mathcal{N}$.

A complex correspondence

Let $\mathcal{Q}$ be the set of positive rational numbers, $\mathcal{Q} = \{\frac{m}{n} | m, n \in \mathcal{N}\}$

A complex correspondence

Let $\mathcal{Q}$ be the set of positive rational numbers, $\mathcal{Q} = \{\frac{m}{n} | m, n \in \mathcal{N}\}$

- Intuitively, $\mathcal{Q}$ seems to be much larger than $\mathcal{N}$
- Yet we can show that this two sets have the same size by constructing the correspondence in Figure 2:

Correspondence $\mathcal{Q} \leftrightarrow \mathcal{N}$

1. Put $\mathcal{N}$ on two axes
2. Line i contains all rational numbers that have numerator i, i.e. $\{\frac{i}{j} \in \mathcal{Q} | i \in \mathcal{N} \textit{ fixed}, \forall j \in \mathcal{N}\}$
3. Column j contains all rational numbers that have denominator j, i.e. $\{\frac{i}{j} \in \mathcal{Q} | \forall i \in \mathcal{N}, j \in \mathcal{N} \textit{ fixed}\}$
4. Number $\frac{i}{j}$ occurs in i-th row and j-th column

Turning $\{\frac{i}{j} | i, j \in \mathcal{N}\}$ into a list

Turning $\{\frac{i}{j} | i, j \in \mathcal{N}\}$ into a list

- Bad idea: list first elements of a line or a column. Lines and columns are labeled by $\mathcal{N}$, hence this would never end

Turning $\{\frac{i}{j} | i,j \in \mathcal{N}\}$ into a list

- Bad idea: list first elements of a line or a column. Lines and columns are labeled by $\mathcal{N}$, hence this would never end
- Good idea (Cantor's idea): use the diagonals:

Turning $\{\frac{i}{j} | i,j \in \mathcal{N}\}$ into a list

- Bad idea: list first elements of a line or a column. Lines and columns are labeled by $\mathcal{N}$, hence this would never end
- Good idea (Cantor's idea): use the diagonals:

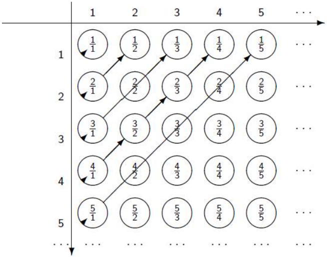

Figure 2 : A correspondence of $\mathcal{N}$ and $\mathcal{Q}$

Turning $\{\frac{i}{j} | i, j \in \mathcal{N}\}$ into a list

- Bad idea: list first elements of a line or a column. Lines and columns are labeled by $\mathcal{N}$, hence this would never end
- Good idea (Cantor's idea): use the diagonals:

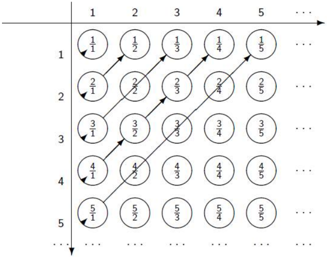

Figure 2 : A correspondence of $\mathcal{N}$ and $\mathcal{Q}$

1. First diagonal contains $\frac{1}{1}$, i.e, first element of the list is $\frac{1}{1}$

Turning $\{\frac{i}{j} | i, j \in \mathcal{N}\}$ into a list

- Bad idea: list first elements of a line or a column. Lines and columns are labeled by $\mathcal{N}$, hence this would never end
- Good idea (Cantor's idea): use the diagonals:

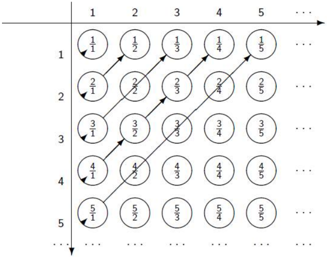

Figure 2 : A correspondence of $\mathcal{N}$ and $\mathcal{Q}$

1. First diagonal contains $\frac{1}{1}$, i.e, first element of the list is $\frac{1}{1}$
2. Continue the list with the elements of the next diagonal: $\frac{2}{1}, \frac{1}{2}$

Turning $\{\frac{i}{j} | i, j \in \mathcal{N}\}$ into a list

- Bad idea: list first elements of a line or a column. Lines and columns are labeled by $\mathcal{N}$, hence this would never end
- Good idea (Cantor's idea): use the diagonals:

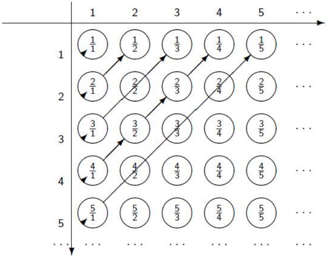

Figure 2 : A correspondence of $\mathcal{N}$ and $\mathcal{Q}$

1. First diagonal contains $\frac{1}{1}$, i.e, first element of the list is $\frac{1}{1}$
2. Continue the list with the elements of the next diagonal: $\frac{2}{1}, \frac{1}{2}$
3. Continue this way skipping the elements that may generate repetitions

The list of rational numbers

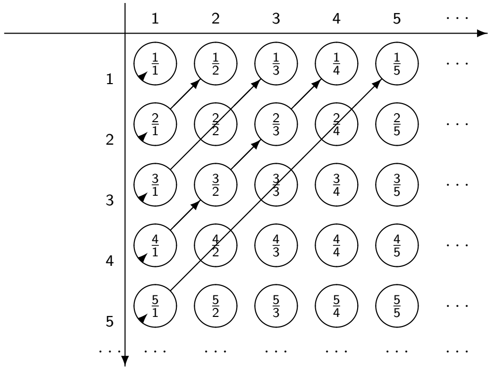

Figure 2 : A correspondence of $\mathcal{N}$ and $\mathcal{Q}$

Uncountable sets

A set for which no correspondence with $\mathcal{N}$ can be established is called *uncountable*

Uncountable sets

A set for which no correspondence with $\mathcal{N}$ can be established is called *uncountable*

Example of uncountable set: the set $\mathcal{R}$ of real numbers is uncountable

Uncountable sets

A set for which no correspondence with $\mathcal{N}$ can be established is called *uncountable*

Example of uncountable set: the set $\mathcal{R}$ of real numbers is uncountable

Proof: Cantor proved that $\mathcal{R}$ is uncountable using the diagonalization method

Theorem 4.17

$\mathcal{R}$ is uncountable

Theorem 4.17

$\mathcal{R}$ is uncountable

Proof: We will show that no correspondence exist between $\mathcal{N}$ and $\mathcal{R}$.

Theorem 4.17

$\mathcal{R}$ is uncountable

Proof: We will show that no correspondence exist between $\mathcal{N}$ and $\mathcal{R}$.

- Suppose that such a correspondence $f : \mathcal{N} \to \mathcal{R}$ exits and deduce a contradiction showing that f fail to work properly.

Theorem 4.17

$\mathcal{R}$ is uncountable

Proof: We will show that no correspondence exist between $\mathcal{N}$ and $\mathcal{R}$.

- Suppose that such a correspondence $f : \mathcal{N} \rightarrow \mathcal{R}$ exits and deduce a contradiction showing that f fail to work properly.
- We construct an $x \in \mathcal{R}$ that cannot be the image of any $n \in \mathcal{N}$.

Construction

- Since $f : \mathcal{N} \to \mathcal{R}$ is a correspondence $\mathcal{R}$ can be listed as seen in Figure 3

n	$f(n)$
1	3.14159 . . .
2	55.5555 . . .
3	0.1234 . . .
4	0.5000 . . .
. . .	. . .

Figure 3 : Listing $\mathcal{R}$

Notation: for $x \in R$, $d_i(x)$ is the i-th digit of x after the decimal.

Formal construction of x

Construct $x \in (0, 1)$ by the following procedure:

Formal construction of x

Construct $x \in (0, 1)$ by the following procedure:

- $x = 0.d_1 d_2 d_3 d_4 \dots$

Formal construction of x

Construct $x \in (0, 1)$ by the following procedure:

- $x = 0.d_1 d_2 d_3 d_4 \ldots$ where for each $i \in \mathcal{N}$ $d_i(x) \neq d_i(f(i))$

Formal construction of x

Construct $x \in (0,1)$ by the following procedure:

- $x = 0.d_1 d_2 d_3 d_4 \ldots$ where for each $i \in \mathcal{N}$ $d_i(x) \neq d_i(f(i))$

  Note: x has an infinite number of decimals constructed by the rule:

  $\forall i \in \mathcal{N}$ *chose* d_i *a digit different from the i-th digit of f(i)*

Formal construction of x

Construct $x \in (0, 1)$ by the following procedure:

- $x = 0.d_1 d_2 d_3 d_4 \ldots$ where for each $i \in \mathcal{N}$ $d_i(x) \neq d_i(f(i))$

  Note: x has an infinite number of decimals constructed by the rule:

  $\forall i \in \mathcal{N}$ *chose d_i a digit different from the i-th digit of f(i)*

- Consequence: $\forall i \in \mathcal{N}$, $x \neq f(i)$. Hence, x does not belong to the list $\mathcal{R}$ and thus f is not a correspondence.

Application

Theorem 4.17 shows that some languages are not decidable or even Turing recognizable.

Application

Theorem 4.17 shows that some languages are not decidable or even Turing recognizable.

Reason:

Application

Theorem 4.17 shows that some languages are not decidable or even Turing recognizable.

Reason:

- There are uncountable many languages yet only countable many Turing machines. (we need to prove this)

Application

Theorem 4.17 shows that some languages are not decidable or even Turing recognizable.

Reason:

- There are uncountable many languages yet only countable many Turing machines. (we need to prove this)
- Because each Turing machine can recognize a single language and there are more languages than Turing machines some languages are not recognized by any Turing machine

Application

Theorem 4.17 shows that some languages are not decidable or even Turing recognizable.

Reason:

- There are uncountable many languages yet only countable many Turing machines. (we need to prove this)
- Because each Turing machine can recognize a single language and there are more languages than Turing machines some languages are not recognized by any Turing machine
- Such languages are not Turing recognizable

Corollary 4.18

Some languages are not Turing-recognizable.

Corollary 4.18

Some languages are not Turing-recognizable.

Proof:

- First we show that the set of Turing machines is countable

Corollary 4.18

Some languages are not Turing-recognizable.

Proof:

- First we show that the set of Turing machines is countable
  1. The set of all strings Σ^* is countable, for any alphabet Σ.

Some languages are not Turing-recognizable.

Proof:

- First we show that the set of Turing machines is countable
  1. The set of all strings Σ^* is countable, for any alphabet Σ.

     Proof: we may form a list Σ^* by writing down all strings of length 0, length 1, length 2, an so on

Corollary 4.18

Some languages are not Turing-recognizable.

Proof:

- First we show that the set of Turing machines is countable
  1. The set of all strings Σ^* is countable, for any alphabet Σ.

     Proof: we may form a list Σ^* by writing down all strings of length 0, length 1, length 2, an so on
  2. Each Turing machine M has an encoding into a string $\langle M \rangle$

Corollary 4.18

Some languages are not Turing-recognizable.

Proof:

- First we show that the set of Turing machines is countable
  1. The set of all strings Σ^* is countable, for any alphabet Σ.

     Proof: we may form a list Σ^* by writing down all strings of length 0, length 1, length 2, an so on
  2. Each Turing machine M has an encoding into a string $\langle M \rangle$
  3. If we omit those strings that are not Turing machines we can obtain a list of all Turing machines

Fact 1

The set of all languages is uncountable

Fact 1

The set of all languages is uncountable

Proof idea: To show that the set of all languages is uncountable we show first that the set of all infinite binary sequences is uncountable

Infinite binary sequences

Let $\mathcal{B}$ be the set of all infinite binary sequences.

Infinite binary sequences

Let $\mathcal{B}$ be the set of all infinite binary sequences.

- Assuming that $\mathcal{B}$ is countable we can set it into a list $f_b : \mathcal{N} \to \mathcal{B}$.

Infinite binary sequences

Let $\mathcal{B}$ be the set of all infinite binary sequences.

- Assuming that $\mathcal{B}$ is countable we can set it into a list $f_b : \mathcal{N} \to \mathcal{B}$.
- By the method of diagonalization we can construct an infinite binary string y, such that $y \neq f_b(i)$ for any $i \in \mathcal{N}$.

Infinite binary sequences

Let $\mathcal{B}$ be the set of all infinite binary sequences.

- Assuming that $\mathcal{B}$ is countable we can set it into a list $f_b : \mathcal{N} \rightarrow \mathcal{B}$.
- By the method of diagonalization we can construct an infinite binary string y, such that $y \neq f_b(i)$ for any $i \in \mathcal{N}$.

  We can chose $y = d_1 d_2 \ldots d_j \ldots$
  such that for each i, d_i is different than i^{th} digit of $f_b(i)$

Proof of fact 1

Fact 1: the set of all languages is uncountable

Fact 1: the set of all languages is uncountable

Let $\mathcal{L}$ be the set of all languages over Σ.

Proof of fact 1

Fact 1: the set of all languages is uncountable

Let $\mathcal{L}$ be the set of all languages over Σ.

- We will show that $\mathcal{L}$ is uncountable by constructing a correspondence $\mathcal{B} \to \mathcal{L}$.

Proof of fact 1

Fact 1: the set of all languages is uncountable

Let $\mathcal{L}$ be the set of all languages over Σ.

- We will show that $\mathcal{L}$ is uncountable by constructing a correspondence $\mathcal{B} \to \mathcal{L}$.
- Since $\mathcal{B}$ is uncountable, and the same size with $\mathcal{L}$ then $\mathcal{L}$ is uncountable

Characteristic sequences

Characteristic sequences

- Since Σ is an alphabet, Σ^* is countable, $\Sigma^* = \{s_1, s_2, s_3, \ldots\}$

Characteristic sequences

- Since Σ is an alphabet, Σ^* is countable, $\Sigma^* = \{s_1, s_2, s_3, \ldots\}$
- Each language $A \in \mathcal{L}$ has a unique infinite binary sequence $\chi_A \in \mathcal{B}$ constructed by:

  the i-th bit of χ_A, $\chi_A(i) = 1$ if $s_i \in A$ and $\chi_A(i) = 0$ if $s_i \notin A$.

Characteristic sequences

- Since Σ is an alphabet, Σ^* is countable, $\Sigma^* = \{s_1, s_2, s_3, \ldots\}$
- Each language $A \in \mathcal{L}$ has a unique infinite binary sequence $\chi_A \in \mathcal{B}$ constructed by:

  the i-th bit of χ_A, $\chi_A(i) = 1$ if $s_i \in A$ and $\chi_A(i) = 0$ if $s_i \notin A$.
- χ_A is the characteristic function of A in Σ^*

Characteristic sequences

- Since Σ is an alphabet, Σ^* is countable, $\Sigma^* = \{s_1, s_2, s_3, \ldots\}$
- Each language $A \in \mathcal{L}$ has a unique infinite binary sequence $\chi_A \in \mathcal{B}$ constructed by:

  the i-th bit of χ_A, $\chi_A(i) = 1$ if $s_i \in A$ and $\chi_A(i) = 0$ if $s_i \notin A$.
- χ_A is the characteristic function of A in Σ^*
- The function $f : \mathcal{L} \rightarrow \mathcal{B}$ where $f(A) = \chi_A$ is one-to-one and onto and hence it is a correspondence.

Characteristic sequences

- Since Σ is an alphabet, Σ^* is countable, $\Sigma^* = \{s_1, s_2, s_3, \ldots\}$
- Each language $A \in \mathcal{L}$ has a unique infinite binary sequence $\chi_A \in \mathcal{B}$ constructed by:

  the i-th bit of χ_A, $\chi_A(i) = 1$ if $s_i \in A$ and $\chi_A(i) = 0$ if $s_i \notin A$.
- χ_A is the characteristic function of A in Σ^*
- The function $f : \mathcal{L} \rightarrow \mathcal{B}$ where $f(A) = \chi_A$ is one-to-one and onto and hence it is a correspondence.
  - f is one-to-one:

Characteristic sequences

- Since Σ is an alphabet, Σ^* is countable, $\Sigma^* = \{s_1, s_2, s_3, \ldots\}$
- Each language $A \in \mathcal{L}$ has a unique infinite binary sequence $\chi_A \in \mathcal{B}$ constructed by:

  the i-th bit of χ_A, $\chi_A(i) = 1$ if $s_i \in A$ and $\chi_A(i) = 0$ if $s_i \notin A$.
- χ_A is the characteristic function of A in Σ^*
- The function $f : \mathcal{L} \to \mathcal{B}$ where $f(A) = \chi_A$ is one-to-one and onto and hence it is a correspondence.
  - f is one-to-one: $\forall L_1, L_2 \in \mathcal{L}$, $L_1 \neq L_2$

Characteristic sequences

- Since Σ is an alphabet, Σ^* is countable, $\Sigma^* = \{s_1, s_2, s_3, \ldots\}$
- Each language $A \in \mathcal{L}$ has a unique infinite binary sequence $\chi_A \in \mathcal{B}$ constructed by:

  the i-th bit of χ_A, $\chi_A(i) = 1$ if $s_i \in A$ and $\chi_A(i) = 0$ if $s_i \notin A$.
- χ_A is the characteristic function of A in Σ^*
- The function $f : \mathcal{L} \to \mathcal{B}$ where $f(A) = \chi_A$ is one-to-one and onto and hence it is a correspondence.
  - f is one-to-one: $\forall L_1, L_2 \in \mathcal{L}$, $L_1 \neq L_2 \Rightarrow \chi_{L_1} \neq \chi_{L_2}$

Characteristic sequences

- Since Σ is an alphabet, Σ^* is countable, $\Sigma^* = \{s_1, s_2, s_3, \ldots\}$
- Each language $A \in \mathcal{L}$ has a unique infinite binary sequence $\chi_A \in \mathcal{B}$ constructed by:

  the i-th bit of χ_A, $\chi_A(i) = 1$ if $s_i \in A$ and $\chi_A(i) = 0$ if $s_i \notin A$.
- χ_A is the characteristic function of A in Σ^*
- The function $f : \mathcal{L} \to \mathcal{B}$ where $f(A) = \chi_A$ is one-to-one and onto and hence it is a correspondence.
  - f is one-to-one: $\forall L_1, L_2 \in \mathcal{L}$, $L_1 \neq L_2 \Rightarrow \chi_{L_1} \neq \chi_{L_2}$
  - f is onto:

Characteristic sequences

- Since Σ is an alphabet, Σ^* is countable, $\Sigma^* = \{s_1, s_2, s_3, \ldots\}$
- Each language $A \in \mathcal{L}$ has a unique infinite binary sequence $\chi_A \in \mathcal{B}$ constructed by:

  the i-th bit of χ_A, $\chi_A(i) = 1$ if $s_i \in A$ and $\chi_A(i) = 0$ if $s_i \notin A$.
- χ_A is the characteristic function of A in Σ^*
- The function $f : \mathcal{L} \to \mathcal{B}$ where $f(A) = \chi_A$ is one-to-one and onto and hence it is a correspondence.
  - f is one-to-one: $\forall L_1, L_2 \in \mathcal{L}$, $L_1 \neq L_2 \Rightarrow \chi_{L_1} \neq \chi_{L_2}$
  - f is onto: $\forall \chi \in \mathcal{B}$ there is a language $L_\chi \in \mathcal{L}$ with $f(L_\chi) = \chi$.

Characteristic sequences

- Since Σ is an alphabet, Σ^* is countable, $\Sigma^* = \{s_1, s_2, s_3, \ldots\}$
- Each language $A \in \mathcal{L}$ has a unique infinite binary sequence $\chi_A \in \mathcal{B}$ constructed by:

  the i-th bit of χ_A, $\chi_A(i) = 1$ if $s_i \in A$ and $\chi_A(i) = 0$ if $s_i \notin A$.
- χ_A is the characteristic function of A in Σ^*
- The function $f : \mathcal{L} \to \mathcal{B}$ where $f(A) = \chi_A$ is one-to-one and onto and hence it is a correspondence.
  - f is one-to-one: $\forall L_1, L_2 \in \mathcal{L}$, $L_1 \neq L_2 \Rightarrow \chi_{L_1} \neq \chi_{L_2}$
  - f is onto: $\forall \chi \in \mathcal{B}$ there is a language $L_\chi \in \mathcal{L}$ with $f(L_\chi) = \chi$. For $\Sigma^* = \{s_1, s_2, \ldots\}$, $L_\chi = \{s_i | s_i \in \Sigma^*$ and i-th digit of χ is 1 $\}$

Conclusion

Since $\mathcal{B}$ is uncountable, $\mathcal{L}$ is uncountable.

Back to the original problem

We are ready to prove that the language

$A_{TM} = \{\langle M, w\rangle | M \textit{ is a TM and M accepts } w\}$

is undecidable.

Proof

Proceeds by contradiction, assuming that A_{TM} is decidable.

Proof

Proceeds by contradiction, assuming that A_{TM} is decidable.

- Suppose that H is a decider of A_{TM}.

Proof

Proceeds by contradiction, assuming that A_{TM} is decidable.

- Suppose that H is a decider of A_{TM}.
- On input $\langle M, w\rangle$ where M is a TM and w is a string, H halts and accepts if M accepts w.

Proof

Proceeds by contradiction, assuming that A_{TM} is decidable.

- Suppose that H is a decider of A_{TM}.
- On input $\langle M, w \rangle$ where M is a TM and w is a string, H halts and accepts if M accepts w.
- Furthermore, H halts and reject if M fails to accept w.

Equational expression of H

$$H(\langle M, w \rangle) = \begin{cases} \mathit{accept}, & \text{if } M \text{ accepts } w; \\ \mathit{reject}, & \text{if } M \text{ does not accept } w. \end{cases}$$

Proof, continuation

Construct a new TM D that uses H as a subroutine.

Proof, continuation

Construct a new TM D that uses H as a subroutine.

- D calls H to determine what M does when its input is $\langle M \rangle$

Proof, continuation

Construct a new TM D that uses H as a subroutine.

- D calls H to determine what M does when its input is $\langle M \rangle$
- If M accepts $\langle M \rangle$ then D rejects;

Proof, continuation

Construct a new TM D that uses H as a subroutine.

- D calls H to determine what M does when its input is $\langle M \rangle$
- If M accepts $\langle M \rangle$ then D rejects;
  if M rejects $\langle M \rangle$ then D accepts

The machine D

$D =$ "On input $\langle M \rangle$, where M is a TM:

The machine D

$D =$ "On input $\langle M \rangle$, where M is a TM:

1. Run H on input $\langle M, \langle M \rangle \rangle$

The machine D

$D =$ "On input $\langle M \rangle$, where M is a TM:

1. Run H on input $\langle M, \langle M \rangle \rangle$
2. Output the opposite of what H outputs:
   if H rejects **accept** and if H accepts then **reject**."

Note

- Running a machine on its own description is a common technique in computer sciences.
- Example, running a compiler on its own description allows compiler implementation and optimization.

In conclusion

$$D(\langle M \rangle) = \begin{cases} \mathit{accept}, & \text{if } M \text{ does not accept } \langle M \rangle; \\ \mathit{reject}, & \text{if } M \text{ accepts } \langle M \rangle. \end{cases}$$

In conclusion

$$D(\langle M \rangle) = \begin{cases} \mathit{accept}, & \text{if } M \text{ does not accept } \langle M \rangle; \\ \mathit{reject}, & \text{if } M \text{ accepts } \langle M \rangle. \end{cases}$$

What happens when we ran D on $\langle D \rangle$?

In conclusion

$$D(\langle M\rangle) = \begin{cases} accept, & \text{if } M \text{ does not accept } \langle M\rangle; \\ reject, & \text{if } M \text{ accepts } \langle M\rangle. \end{cases}$$

What happens when we ran D on $\langle D\rangle$?

$$D(\langle D\rangle) = \begin{cases} accept, & \text{if } D \text{ does not accept } \langle D\rangle; \\ reject, & \text{if } D \text{ does not reject } \langle D\rangle. \end{cases}$$

In conclusion

$$D(\langle M \rangle) = \begin{cases} \mathit{accept}, & \text{if } M \text{ does not accept } \langle M \rangle; \\ \mathit{reject}, & \text{if } M \text{ accepts } \langle M \rangle. \end{cases}$$

What happens when we ran D on $\langle D \rangle$?

$$D(\langle D \rangle) = \begin{cases} \mathit{accept}, & \text{if } D \text{ does not accept } \langle D \rangle; \\ \mathit{reject}, & \text{if } D \text{ does not reject } \langle D \rangle. \end{cases}$$

This is a contradiction and consequently neither TM D nor TM H do exist.

Summarizing

Summarizing

- Assume that H decides A_{TM}

Summarizing

- Assume that H decides A_{TM}
- Use H to build D that accepts $\langle M \rangle$ when M rejects and rejects $\langle M \rangle$ when M accepts

Summarizing

- Assume that H decides A_{TM}
- Use H to build D that accepts $\langle M \rangle$ when M rejects and rejects $\langle M \rangle$ when M accepts
- H and D performs as follows:

Summarizing

- Assume that H decides A_{TM}
- Use H to build D that accepts $\langle M \rangle$ when M rejects and rejects $\langle M \rangle$ when M accepts
- H and D performs as follows:
  - H accepts $\langle M, w \rangle$ exactly when M accepts w

Summarizing

- Assume that H decides A_{TM}
- Use H to build D that accepts $\langle M \rangle$ when M rejects and rejects $\langle M \rangle$ when M accepts
- H and D performs as follows:
  - H accepts $\langle M, w \rangle$ exactly when M accepts w
  - D rejects $\langle M \rangle$ exactly when M accepts $\langle M \rangle$

Summarizing

- Assume that H decides A_{TM}
- Use H to build D that accepts $\langle M \rangle$ when M rejects and rejects $\langle M \rangle$ when M accepts
- H and D performs as follows:
  - H accepts $\langle M, w \rangle$ exactly when M accepts w
  - D rejects $\langle M \rangle$ exactly when M accepts $\langle M \rangle$
  - D rejects $\langle D \rangle$ exactly when D accepts $\langle D \rangle$

Summarizing

- Assume that H decides A_{TM}
- Use H to build D that accepts $\langle M \rangle$ when M rejects and rejects $\langle M \rangle$ when M accepts
- H and D performs as follows:
  - H accepts $\langle M, w \rangle$ exactly when M accepts w
  - D rejects $\langle M \rangle$ exactly when M accepts $\langle M \rangle$
  - D rejects $\langle D \rangle$ exactly when D accepts $\langle D \rangle$

This is a contradiction and neither H nor D can exist

Where is diagonalization?

To make the use of diagonalization obvious we construct the list of all Turing machines running on Turing machines as input in Figures 4,5,6.

	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\cdots$
M_1	*accept*		*accept*		
M_2	*accept*	*accept*	*accept*	*accept*	$\cdots$
M_3					$\cdots$
M_4	*accept*	*accept*			

Figure 4 : Entry (i,j) is accept if M_i accepts $\langle M_j \rangle$

Running H

Figure 5 shows the result of running H on the machine in Figure 4

	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\cdots$
M_1	*accept*	*reject*	*accept*	*reject*	$\cdots$
M_2	*accept*	*accept*	*accept*	*accept*	$\cdots$
M_3	*reject*	*reject*	*reject*	*reject*	$\cdots$
M_4	*accept*	*accept*	*reject*	*reject*	$\cdots$

Figure 5 : Entry (i,j) is the value of H on $\langle M_i, \langle M_j \rangle \rangle$

Running D on $\langle D \rangle$

Figure 6 shows the result of running H on the machine in Figure 4 when D is present.

	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\cdots$	$\langle D \rangle$	$\cdots$
M_1	*accept*	*reject*	*accept*	*reject*	$\cdots$	*accept*	$\cdots$
M_2	*accept*	*accept*	*accept*	*accept*	$\cdots$	*accept*	$\cdots$
M_3	*reject*	*reject*	*reject*	*reject*	$\cdots$	*reject*	$\cdots$
M_4	*accept*	*accept*	*reject*	*reject*	$\cdots$	*accept*	$\cdots$
D							

Running D on $\langle D \rangle$

Figure 6 shows the result of running H on the machine in Figure 4 when D is present.

	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\cdots$	$\langle D \rangle$	$\cdots$
M_1	*accept*	*reject*	*accept*	*reject*	$\cdots$	*accept*	$\cdots$
M_2	*accept*	*accept*	*accept*	*accept*	$\cdots$	*accept*	$\cdots$
M_3	*reject*	*reject*	*reject*	*reject*	$\cdots$	*reject*	$\cdots$
M_4	*accept*	*accept*	*reject*	*reject*	$\cdots$	*accept*	$\cdots$
D	*reject*						

Running D on $\langle D \rangle$

Figure 6 shows the result of running H on the machine in Figure 4 when D is present.

	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\cdots$	$\langle D \rangle$	$\cdots$
M_1	*accept*	*reject*	*accept*	*reject*	$\cdots$	*accept*	$\cdots$
M_2	*accept*	*accept*	*accept*	*accept*	$\cdots$	*accept*	$\cdots$
M_3	*reject*	*reject*	*reject*	*reject*	$\cdots$	*reject*	$\cdots$
M_4	*accept*	*accept*	*reject*	*reject*	$\cdots$	*accept*	$\cdots$
D	*reject*	*reject*					

Running D on $\langle D \rangle$

Figure 6 shows the result of running H on the machine in Figure 4 when D is present.

	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\cdots$	$\langle D \rangle$	$\cdots$
M_1	*accept*	*reject*	*accept*	*reject*	$\cdots$	*accept*	$\cdots$
M_2	*accept*	*accept*	*accept*	*accept*	$\cdots$	*accept*	$\cdots$
M_3	*reject*	*reject*	*reject*	*reject*	$\cdots$	*reject*	$\cdots$
M_4	*accept*	*accept*	*reject*	*reject*	$\cdots$	*accept*	$\cdots$
D	*reject*	*reject*	*accept*				

Running D on $\langle D \rangle$

Figure 6 shows the result of running H on the machine in Figure 4 when D is present.

	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\cdots$	$\langle D \rangle$	$\cdots$
M_1	*accept*	*reject*	*accept*	*reject*	$\cdots$	*accept*	$\cdots$
M_2	*accept*	*accept*	*accept*	*accept*	$\cdots$	*accept*	$\cdots$
M_3	*reject*	*reject*	*reject*	*reject*	$\cdots$	*reject*	$\cdots$
M_4	*accept*	*accept*	*reject*	*reject*	$\cdots$	*accept*	$\cdots$
D	*reject*	*reject*	*accept*	*accept*			

Running D on $\langle D \rangle$

Figure 6 shows the result of running H on the machine in Figure 4 when D is present.

	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\cdots$	$\langle D \rangle$	$\cdots$
M_1	*accept*	*reject*	*accept*	*reject*	$\cdots$	*accept*	$\cdots$
M_2	*accept*	*accept*	*accept*	*accept*	$\cdots$	*accept*	$\cdots$
M_3	*reject*	*reject*	*reject*	*reject*	$\cdots$	*reject*	$\cdots$
M_4	*accept*	*accept*	*reject*	*reject*	$\cdots$	*accept*	$\cdots$
D	*reject*	*reject*	*accept*	*accept*	$\cdots$		

Running D on $\langle D \rangle$

Figure 6 shows the result of running H on the machine in Figure 4 when D is present.

	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\cdots$	$\langle D \rangle$	$\cdots$
M_1	*accept*	*reject*	*accept*	*reject*	$\cdots$	*accept*	$\cdots$
M_2	*accept*	*accept*	*accept*	*accept*	$\cdots$	*accept*	$\cdots$
M_3	*reject*	*reject*	*reject*	*reject*	$\cdots$	*reject*	$\cdots$
M_4	*accept*	*accept*	*reject*	*reject*	$\cdots$	*accept*	$\cdots$
D	*reject*	*reject*	*accept*	*accept*	$\cdots$	???	$\cdots$

Figure 6 : A contradiction occurs at $\langle D, \langle D \rangle \rangle$

Note

We can construct a Turing-unrecognizable language

Note

We can construct a Turing-unrecognizable language

- A_{TM} is an example of Turing undecidable language. But it is Turing recognizable

Note

We can construct a Turing-unrecognizable language

- A_{TM} is an example of Turing undecidable language. But it is Turing recognizable
- Now we construct a language which is Turing-unrecognizable.

Note

We can construct a Turing-unrecognizable language

- A_{TM} is an example of Turing undecidable language. But it is Turing recognizable
- Now we construct a language which is Turing-unrecognizable.
- This construction relies on the fact that if both a language and its complement are Turing-recognizable the language is decidable

Note

We can construct a Turing-unrecognizable language

- A_{TM} is an example of Turing undecidable language. But it is Turing recognizable
- Now we construct a language which is Turing-unrecognizable.
- This construction relies on the fact that if both a language and its complement are Turing-recognizable the language is decidable

That is: *for any undecidable language, either the language or its complement is not Turing-recognizable*

A new concept

Co-Turing recognizable languages

A new concept

Co-Turing recognizable languages

- Complement of a language A is the language consisting of all strings that does not belong to A.

A new concept

Co-Turing recognizable languages

- Complement of a language A is the language consisting of all strings that does not belong to A.
- A language is co-Turing-recognizable if it is the complement of a Turing-recognizable language

Theorem 4.22

A language is decidable iff it is both Turing-recognizable and co-Turing recognizable

Theorem 4.22

A language is decidable iff it is both Turing-recognizable and co-Turing recognizable

i.e., a language A is decidable iff both A and $\overline{A}$ are Turing-recognizable

Proof

Proof

if Assume that A is decidable. Since complement of a decidable language is decidable it result that both A and $\overline{A}$ are Turing-recognizable.

Proof

if Assume that A is decidable. Since complement of a decidable language is decidable it result that both A and $\overline{A}$ are Turing-recognizable.

only if Assume that both A and $\overline{A}$ are Turing-recognizable. Let M_1 be a recognizer for A and M_2 a recognizer for $\overline{A}$. Then the following TM M is a decider for A

Construction

$M =$ "On input w:

Construction

$M =$ "On input w:

Construction

$M =$ "On input w:

1. Run both M_1 and M_2 on w in parallel

Construction

$M =$ "On input w:

1. Run both M_1 and M_2 on w in parallel
2. If M_1 accepts w **accept**; if M_2 accepts w **reject**."

Note

Note

- Running two machines M_1 and M_2 by a machine M in parallel means that M has two tapes, one for simulating M_1 and other for simulating M_2

Note

- Running two machines M_1 and M_2 by a machine M in parallel means that M has two tapes, one for simulating M_1 and other for simulating M_2
- M takes turns, simulating one step of each machine, which continues until one of the machines halts.

Note

- Running two machines M_1 and M_2 by a machine M in parallel means that M has two tapes, one for simulating M_1 and other for simulating M_2
- M takes turns, simulating one step of each machine, which continues until one of the machines halts.
- Because $w \in A$ or $w \in \overline{A}$ either M_1 or M_2 must accepts w.

Note

- Running two machines M_1 and M_2 by a machine M in parallel means that M has two tapes, one for simulating M_1 and other for simulating M_2
- M takes turns, simulating one step of each machine, which continues until one of the machines halts.
- Because $w \in A$ or $w \in \overline{A}$ either M_1 or M_2 must accepts w.
- Because M halts whenever M_1 or M_2 accepts, M always halts, so it is a decider. Further, it accepts all strings from A and rejects all strings not in A.

Conclusion

M is a decider for A, thus A is decidable

Corollary

$\overline{A_{TM}}$ is not Turing-recognizable

Corollary

$\overline{A_{TM}}$ is not Turing-recognizable

Proof: We know that A_{TM} is Turing-recognizable. If $\overline{A_{TM}}$ also were Turing-recognizable then A_{TM} would be decidable. But we have proved (Theorem 4.11) that A_{TM} is not decidable. Hence, $\overline{A_{TM}}$ must not be Turing-recognizable.