

The Diagonalization Method

Teodor Rus

`rus@cs.uiowa.edu`

The University of Iowa, Department of Computer Science

Decidability of TM language

Problem:

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

Language:

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

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:

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.

Facts

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

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?

Fact

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?

Design a mapping $f : \mathcal{N} \rightarrow \{0, 1\}^*$ by:

1. $f(0) = \epsilon$;
2. $f(n) = \{x \in \{0, 1\}^* \mid |x| = n\}$.

Note: f actually maps \mathcal{N} on $\mathcal{P}(\{0, 1\}^*)$; since $|x| = |y|$ for any $x, y \in f(n)$, \parallel is an equivalence relation on $\{0, 1\}^*$ and we can consider $f : \mathcal{N} \rightarrow \{0, 1\}^* / \parallel$.

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 $f : A \rightarrow B$ 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*.

Note: a correspondence is a mechanism that allows us to pair elements of two sets.

Size comparison

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

Example correspondences

- Let \mathcal{N} be the set of natural numbers, $\mathcal{N} = \{1, 2, 3, \dots\}$ and \mathcal{E} the set of even natural numbers, $\mathcal{E} = \{2, 4, 6, \dots\}$;
- 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.

$$\text{sizeof}(\mathcal{N}) = \text{sizeof}(\mathcal{E})$$

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

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

Definition 4.12

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,

$$\mathcal{Q} = \left\{ \frac{m}{n} \mid m, n \in \mathcal{N} \right\}.$$

- Intuitively, \mathcal{Q} seems to be much larger than \mathcal{N} ;
- Yet we can show that these 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. $\{\frac{i}{j} \in \mathcal{Q} | i \in \mathcal{N} \text{ fixed}, \forall j \in \mathcal{N}\}$;
3. Column j contains all rational numbers that have denominator j , i.e. $\{\frac{i}{j} \in \mathcal{Q} | \forall i \in \mathcal{N}, j \in \mathcal{N} \text{ fixed}\}$;
4. Number $\frac{i}{j}$ occurs in i -th line and j -th column.

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:
 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, such $\frac{i}{i}$ that would generate a copy of $\frac{1}{1}$.

The list of rational numbers

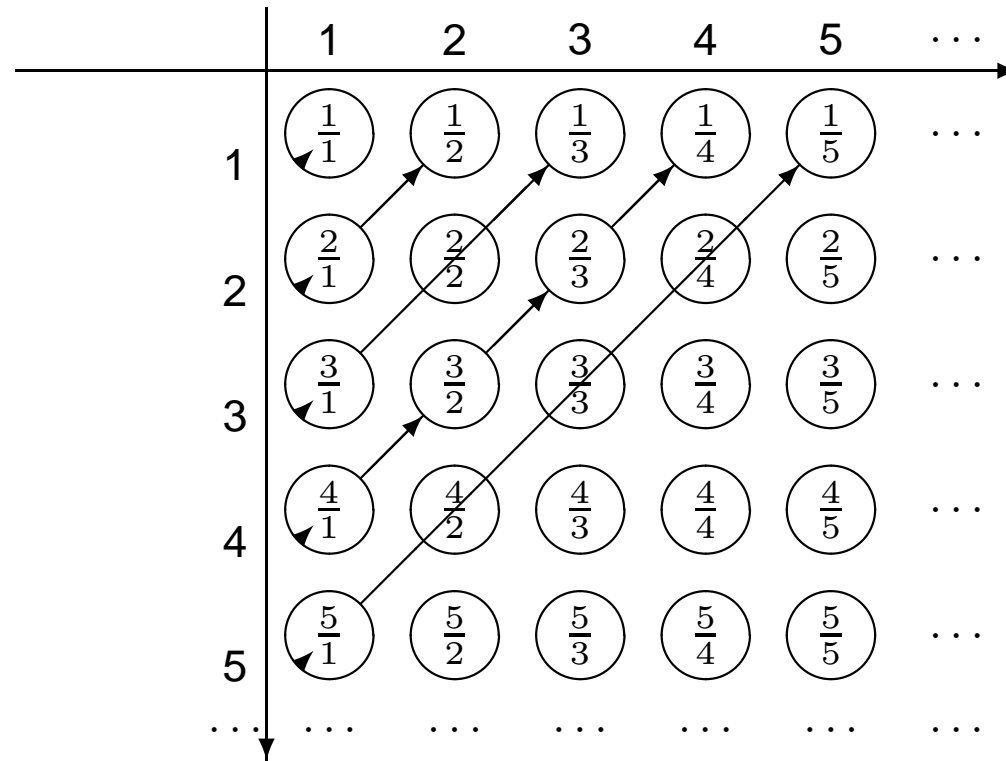


Figure 2: A correspondence of \mathcal{N} and \mathcal{Q}

Uncountable sets

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

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 (4.14)

\mathcal{R} is uncountable.

Proof: by contradiction using diagonalization.

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

- Suppose that such a correspondence $f : \mathcal{N} \rightarrow \mathcal{R}$ exists 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 $f : \mathcal{N} \rightarrow \mathcal{R}$ 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 $x \in \mathcal{R}$, $d_i(x)$ is the i -th digit of x .

Formal construction of x

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

- $x = 0.d_1d_2d_3d_4 \dots$ where $\forall i \in \mathcal{N}[d_i(x) \neq d_i(f(i))]$

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.

Reason:

- There are uncountable many languages yet only countable many Turing machines.
- 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.

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, and 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.

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.

- 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 $i \in \mathcal{N}$;

Construction: $y = d_1 d_2 \dots d_j \dots$ such that

$$\forall i \forall j [d_j \in \{0, 1\} \wedge d_j \neq d_j(f_b(i))]$$

Facts

Fact 1: the set of all languages is uncountable.

Proof: by construction:

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

- We will show that \mathcal{L} is uncountable by constructing a correspondence $\mathcal{B} \rightarrow \mathcal{L}$.
- Since \mathcal{B} is uncountable, and has the same size as \mathcal{L} it result that \mathcal{L} is uncountable.

Characteristic sequences

- Since Σ is an alphabet, Σ^* is countable, $\Sigma^* = \{s_1, s_2, s_3, \dots\}$ be the set of all strings over Σ ;
- 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: $\forall L_1, L_2 \in \mathcal{L}, L_1 \neq L_2 \Rightarrow \chi_{L_1} \neq \chi_{L_2}$;
 - f is onto: $\forall \chi \in \mathcal{B}$ there is a language $L_\chi \in \mathcal{L}$ with $f(L) = \chi$.
For $\Sigma^* = \{s_1, s_2, \dots\}$, $L_\chi = \{s_i \mid s_i \in \Sigma^* \wedge d_i(\chi) = 1\}$.

Conclusion

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

Example characteristic function

Let $\Sigma = \{0, 1\}$ and A be the language of all strings starting with 0 over Σ .

$\Sigma^* = \{e, 0, 1, 00, 01, 11, 000, 001, 010, 011, 100, \dots\}$;

$A = \{0, 00, 01, 000, 001, 010, 011, \dots\}$;

$\chi_A = \{0 \ 1 \ 0 \ 1 \ 1 \ 0 \ 1 \ 1 \ 1 \ 1 \ \dots\}$

Fact 2

For M an TM, the language $L(M)$ is undecidable.

We are ready to prove that the language

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

is undecidable.

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

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

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:

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

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

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

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

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

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

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.

	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	\dots
M_1	<i>accept</i>		<i>accept</i>		
M_2	<i>accept</i>	<i>accept</i>	<i>accept</i>	<i>accept</i>	\dots
M_3					\dots
M_4	<i>accept</i>	<i>accept</i>			

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.

	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	\dots
M_1	<i>accept</i>	<i>reject</i>	<i>accept</i>	<i>reject</i>	\dots
M_2	<i>accept</i>	<i>accept</i>	<i>accept</i>	<i>accept</i>	\dots
M_3	<i>reject</i>	<i>reject</i>	<i>reject</i>	<i>reject</i>	\dots
M_4	<i>accept</i>	<i>accept</i>	<i>reject</i>	<i>reject</i>	\dots

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.

	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	\dots	$\langle D \rangle$	\dots
M_1	<i>accept</i>	<i>reject</i>	<i>accept</i>	<i>reject</i>	\dots	<i>accept</i>	\dots
M_2	<i>accept</i>	<i>accept</i>	<i>accept</i>	<i>accept</i>	\dots	<i>accept</i>	\dots
M_3	<i>reject</i>	<i>reject</i>	<i>reject</i>	<i>reject</i>	\dots	<i>reject</i>	\dots
M_4	<i>accept</i>	<i>accept</i>	<i>reject</i>	<i>reject</i>	\dots	<i>accept</i>	\dots
D	<i>reject</i>	<i>reject</i>	<i>accept</i>	<i>accept</i>	\dots	???	\dots

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

Observation

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 language.

- 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 (4.16)

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 $\mathcal{C}(A)$ are Turing-recognizable

Proof

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

For L decidable, decided by M , the TM \overline{M} that decides \overline{L} is:

\overline{M} = "On any input w :

1. Run M on w
2. If M accepts *reject*,
if M rejects *accept*".

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

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*."

Observations

- 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 $w \in \mathcal{C}(A)$ either M_1 or M_2 must accept 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

$\mathcal{C}(A_{TM})$ is not Turing-recognizable.

Proof: We know that A_{TM} is Turing-recognizable.

1. If $\mathcal{C}(A_{TM})$ also were Turing-recognizable then A_{TM} would be decidable.
2. But we have proved (Theorem 4.11) that A_{TM} is not decidable.
3. Hence, $\mathcal{C}(A_{TM})$ must not be Turing-recognizable.

Application 1

Let A be a Turing-recognizable language consisting of descriptions of Turing machines, i.e.,

$$A = \{\langle M_1 \rangle, \langle M_2 \rangle, \dots\}$$

where every M_i is a decider.

Prove that some decidable language L_D is not decided by any decider M_i whose description appears in A .

Solution sketch

Use the diagonalization method to construct a decider D whose language is not among those decided by M_i , $i = 1, 2, \dots$

Since A is Turing-recognizable there is an enumerator E that enumerates the elements of A . Let $\langle M_1 \rangle, \langle M_2 \rangle, \dots$ be the output of E . Let $\Lambda = \{s_1, s_2, \dots\}$ be a list of all strings over the alphabets of A .

The algorithm for D is:

$D =$ "On input w :

1. Let i be the index of w on the list of strings Λ , i.e., $s_i = w$.
2. Run $\langle M_i \rangle$ on input w
3. If $\langle M_i \rangle$ accepts, *reject*;
if $\langle M_i \rangle$ rejects, *accept*".

Note: D is a decider because each M_i is a decider. But D does not appear on the list enumerated for A , because $\forall i$, D differs from M_i on input s_i

Application 2

Let A and B be two disjoint languages. Say that a language C *separates* A and B if $A \subseteq C$ and $B \subseteq \overline{C}$. Show that any two disjoint co-Turing-recognizable languages are separable by some decidable language.

Solution sketch

Let A and B be two languages such that $A \cap B = \emptyset$ and \bar{A} and \bar{B} are Turing-recognizable (by definition).

Let J be the TM that recognizes \bar{A} and K be the TM that recognizes \bar{B} . Then the language C decided by the following TM T separates A and B :

$T =$ "On input w :

1. Simulate J and K on w by alternating the steps of the two machines;
2. If J accepts first, *reject*. If K accepts first, *accept*".

$L(T)$ separates A and B

1. $\bar{A} = \Sigma^* \setminus A$, $\bar{B} = \Sigma^* \setminus B$, $\bar{A} \cup \bar{B} = \Sigma^* \setminus A \cap B$. Since $A \cap B = \emptyset$ it results that $\bar{A} \cup \bar{B} = \Sigma^*$.
2. $\bar{A} \cup \bar{B} = \Sigma^*$ implies that T is a decider because for any $w \in \Sigma^*$, $w \in \bar{A}$ or $w \in \bar{B}$. Hence, either J or K will accept w .
3. $A \subseteq L(T)$ because if $w \in A$, w will not be recognized by J and will be accepted by K first, i.e., $w \in L(T)$.
4. $B \subseteq \overline{L(T)}$ because if $w \in B$, w will not be recognized by K and will be accepted by J first, i.e., $w \in \overline{L(T)}$.

Conclusion: by the definition of separability, language $C = L(T)$ separates A and B