



## Pre-lecture brain teaser

What do each of the reductions prove?

1.  $u - v$  shortest path  $\leq_P$  All pairs shortest path

$$\langle G, u, v \rangle \longrightarrow \langle G \rangle$$

2.  $\text{SAT} \leq_P \text{Longest path}$  <sup>1</sup>

$$(\text{NP-C}) \longrightarrow (\text{NP-Hard})$$

3. Shortest path  $\leq_P \text{SAT}$  <sup>2</sup>

$$(P)$$

---

<sup>1</sup>Given a graph  $G = (V, E)$  and integer  $k$ , is there a simple path that uses at least  $k$  vertices.

<sup>2</sup>[http://www.aloul.net/Papers/faloul\\_iceee06.pdf](http://www.aloul.net/Papers/faloul_iceee06.pdf).

# ECE-374-B: Lecture 22 - Decidability I

---

**Instructor:** Abhishek Kumar Umrawal

April 16, 2024

University of Illinois at Urbana-Champaign

## Pre-lecture brain teaser

What do each of the reductions prove?

1.  $u - v$  shortest path  $\leq_P$  All pairs shortest path

2. SAT  $\leq_P$  Longest path <sup>3</sup>

3. Shortest path  $\leq_P$  SAT <sup>4</sup>

---

<sup>3</sup>Given a graph  $G = (V, E)$  and integer  $k$ , is there a simple path that uses at least  $k$  vertices.

<sup>4</sup>[http://www.aloul.net/Papers/faloul\\_iceee06.pdf](http://www.aloul.net/Papers/faloul_iceee06.pdf).

# Cantor's diagonalization argument

---

# Diagonalization Intro

Published in 1891 by George Cantor, is a proof that sought to answer the following question.

Are all infinite sets ( $\mathbb{N}$ ,  $\mathbb{Q}$ ,  $\mathbb{Z}$ ,  $\mathbb{R}$ ,  $\mathbb{C}$ ) the same size?

# Diagonalization Intro

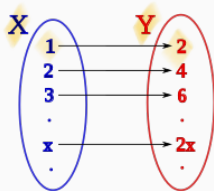
Published in 1891 by George Cantor, is a proof that sought to answer the following question.

Are all infinite sets ( $\mathbb{N}$ ,  $\mathbb{Q}$ ,  $\mathbb{Z}$ ,  $\mathbb{R}$ ,  $\mathbb{C}$ ) the same size?

Let's say two sets are the same size if there is a 1-1 mapping between them.

bijection (1:1 correspondence)

$\Rightarrow$  invertibility



First we need an anchor point ( $\mathbb{N}$ ). Let's say the set of natural numbers has a particular size  $\aleph_0$ .

$$|\mathbb{N}| = \aleph_0 \quad \text{Aleph zero}$$

# Countable Sets I

We say the set  $\mathbb{N}$  is countable because you can list out all its elements systematically, i.e., enumerate them.

$$\underline{1, 2, 3, 4, 5, 6, \dots} \quad (1)$$

$$\text{If: } f: S \rightarrow \mathbb{N}$$

1:1 map

$\Rightarrow S$  is countable.



# Countable Sets I

We say the set  $\mathbb{N}$  is countable because you can list out all its elements systematically, i.e., enumerate them.

$$1, 2, 3, 4, 5, 6, \dots \quad (1)$$

Set of integers is also countable.

$\mathbb{Z}$

$$f: \mathbb{Z} \rightarrow \mathbb{N}$$

1:1 map

Can you obtain such an  $f$ ? YES!

$\mathbb{Z}$ : All positive and negative integers and zero

## Countable Sets II

Set of rational numbers is also countable.

$$\mathbb{N} = 1, 2, 3, 4, 5, 6, \dots$$

$$\mathbb{Q} =$$

$\mathbb{N} \searrow \mathbb{N} \rightarrow$	<u>1</u>	<u>2</u>	<u>3</u>	<u>4</u>	<u>5</u>	<u>6</u>	...
<u>1</u>	$\frac{1}{1}$	$\frac{1}{2}$	$\frac{1}{3}$	$\frac{1}{4}$	$\frac{1}{5}$	$\frac{1}{6}$	...
<u>2</u>	$\frac{2}{1}$	$\frac{2}{2}$	$\frac{2}{3}$	$\frac{2}{4}$	$\frac{2}{5}$	$\frac{2}{6}$	...
<u>3</u>	$\frac{3}{1}$	$\frac{3}{2}$	$\frac{3}{3}$	$\frac{3}{4}$	$\frac{3}{5}$	$\frac{3}{6}$	...
<u>4</u>	$\frac{4}{1}$	$\frac{4}{2}$	$\frac{4}{3}$	$\frac{4}{4}$	$\frac{4}{5}$	$\frac{4}{6}$	...
<u>5</u>	$\frac{5}{1}$	$\frac{5}{2}$	$\frac{5}{3}$	$\frac{5}{4}$	$\frac{5}{5}$	$\frac{5}{6}$	...
<u>6</u>	$\frac{6}{1}$	$\frac{6}{2}$	$\frac{6}{3}$	$\frac{6}{4}$	$\frac{6}{5}$	$\frac{6}{6}$	...
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	

Focus on ordering numbers based on the diagonals.

# Countable Sets III

Is the set of complex integers countable?

↪  $x + iy$  where  $x, y \in \mathbb{N}$   
 $i = \sqrt{-1}$

$\mathbb{N} \downarrow \mathbb{N} \rightarrow$	1	2	3	4	5	6	...
1	$1+1i$ ↓	$1+2i$ ↗	$1+3i$ →	...			
2	$2+1i$	$2+2i$ ↖	$2+3i$ ←	...			
3	⋮	⋮	⋮				
4							
5							
6							
⋮							

## Countable Sets IV

Is  $\mathbb{R}$  countable?

1	0.	9	8	2	1	2	...
2	0.	4	8	6	8	5	...
3	0.	1	7	3	7	9	
4	0.	0	6	7	2	7	
5	0.	3	2	3	4	8	
6	0.	0	3	2	7	0	
$\vdots$							

How do we draw a 1-1 mapping between  $\mathbb{N}$  and  $\mathbb{R}$ ?

# Countable Sets IV

Is  $\mathbb{R}$  countable?

Arrangement (w.l.g.)

Focus on  $[0, 1)$

Q.  $D \in \text{Arrangement?}$

NO!

1 0. 9 8 2 1 2 ...

2 0. 4 8 6 8 5 ...

3 0. 1 7 3 7 9

4 0. 0 6 7 2 7

5 0. 3 2 3 4 8

6 0. 0 3 2 7 0

$\vdots$

D 0. 0 9 4 2 9 ...

## You can not count the real numbers II

$$I = (0, 1), \mathbb{N} = \{1, 2, 3, \dots\}. \quad (\text{R14})$$

### Claim (Cantor)

$|\mathbb{N}| \neq |I|$ , where  $I = (0, 1)$ .

### Proof.

Write every number in  $(0, 1)$  in its decimal expansion. E.g.,

$$1/3 = 0.33333333333333333333 \dots$$

Assume that  $|\mathbb{N}| = |I|$ . Then there exists a one-to-one mapping  $f : \mathbb{N} \rightarrow I$ . Let  $\beta_i$  be the  $i^{\text{th}}$  digit of  $f(i) \in (0, 1)$ .

$d_i =$  any number in  $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\} \setminus \{d_{i-1}, \beta_i\}$

$$D = 0.d_1d_2d_3 \dots \in (0, 1).$$

$D$  is a well defined unique number in  $(0, 1)$ ,

But there is no  $j$  such that  $f(j) = D$ . A contradiction.



# “Most General” computer?

- DFA's are simple model of computation.
- Accept only the regular languages.
- Is there a kind of computer that can accept any language, or compute any function?
- Recall counting argument. Set of all languages:  
 $\{L \mid L \subseteq \{0,1\}^*\}$  is ~~countably infinite~~ / uncountably infinite
- Set of all programs:  $\Sigma^*$   
 $\{P \mid P \text{ is a finite length computer program}\}$ :  
is countably infinite / ~~uncountably infinite~~.

# “Most General” computer?

- DFA are simple model of computation.
- Accept only the regular languages.
- Is there a kind of computer that can accept any language, or compute any function?
- Recall counting argument. Set of all languages:  
 $\{L \mid L \subseteq \{0,1\}^*\}$  is ~~countably infinite~~ / uncountably infinite
- Set of all programs:  
 $\{P \mid P \text{ is a finite length computer program}\}$ :  
is countably infinite / ~~uncountably infinite~~.
- **Conclusion:** There are languages for which there are no programs.



# Program Diagonalization

How do we know that there are languages that cannot be represented by programs? Use Cantor!

# Program Diagonalization

How do we know that there are languages that cannot be represented by programs? Use Cantor! Recall a program can be represented by a string where:

- $M$  is the Turing machine (program), and
- $\langle M \rangle$  is the string representation of the TM  $M$ .

# Program Diagonalization

Define  $f(i, j) = 1$  if  $M_i$  accepts  $\langle M_j \rangle$ , else 0.

	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\langle M_5 \rangle$	$\langle M_6 \rangle$	...
$M_1$	0	1	1	1	1	1	
$M_2$	1	1	0	0	0	0	
$M_3$	0	0	0	1	0	0	
$M_4$	1	1	1	0	1	1	
$M_5$	1	0	0	0	1	0	
$M_6$	0	1	0	1	1	0	
$\vdots$							

# Program Diagonalization

Let's define a new program as follows.

$$D = \{\langle M \rangle \mid M \text{ does not accept } \langle M \rangle\}$$

# Program Diagonalization

Let's define a new program as follows.

$$D = \{\langle M \rangle \mid M \text{ does not accept } \langle M \rangle\}$$

	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\langle M_5 \rangle$	$\langle M_6 \rangle$	...	$\langle M_D \rangle$
$M_1$	0	1	1	1	1	1		1
$M_2$	1	1	0	0	0	0		1
$M_3$	0	0	0	1	0	0		1
$M_4$	1	1	1	0	1	1		0
$M_5$	1	0	0	0	1	0		0
$M_6$	0	1	0	1	1	0		1
$\vdots$								
$M_D$	1	0	1	1	0	1	$\frac{1}{0}$ <input type="checkbox"/>	

## Recap of decidability

---

# Recursive vs. Recursively Enumerable

- Recursively enumerable (aka RE) languages:

$$L = \{L(M) \mid M \text{ some Turing machine}\}.$$

- Recursive / decidable languages:

$$L = \{L(M) \mid M \text{ some Turing machine that halts on all inputs}\}.$$

## Recursive vs. Recursively Enumerable

- Recursively enumerable (aka RE) languages: (bad)

$$L = \{L(M) \mid M \text{ some Turing machine}\}.$$

- Recursive / decidable languages: (good)

$$L = \{L(M) \mid M \text{ some Turing machine that halts on all inputs}\}.$$



# Recursive vs. Recursively Enumerable

- Recursively enumerable (aka RE) languages: (bad)

$$L = \{L(M) \mid M \text{ some Turing machine}\}.$$

- Recursive / decidable languages: (good)

$$L = \{L(M) \mid M \text{ some Turing machine that halts on all inputs}\}.$$

- Fundamental questions:
  - What languages are RE?
  - Which are recursive?
  - What is the difference?
  - What makes a language decidable?

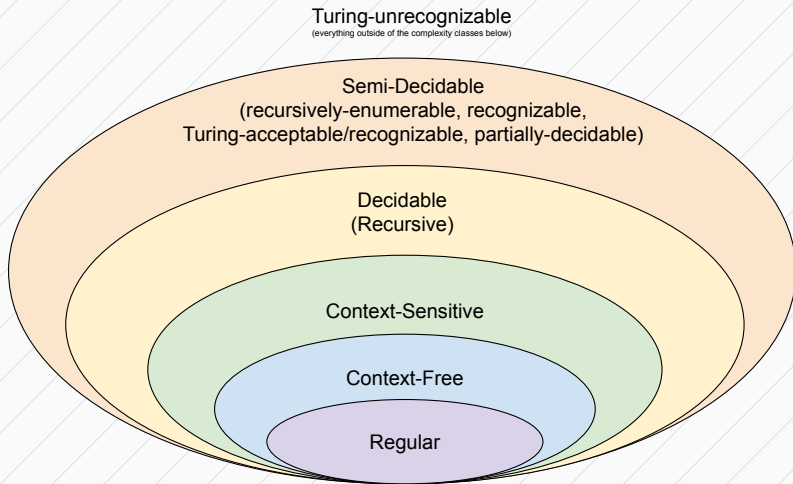
# Decidable vs recursively-enumerable

A semi-decidable problem (equivalent of recursively enumerable) could be:

- **Decidable** - equivalent of recursive (TM always accepts or rejects).
- **Undecidable** - Problem is not recursive (doesn't always halt on negative)

There are undecidable problems that are not semi-decidable (recursively enumerable).

# Problem (Language) Space



Like in the case of NP-complete-ness, we need an anchor point to compare languages to to determine whether they are decidable (or not)!

# Introduction to the halting theorem

---

# The halting problem

**Halting problem:** Given a program  $Q$ , if we run it would it stop?

# The halting problem

**Halting problem:** Given a program  $Q$ , if we run it would it stop?

**Q:** Can one build a program  $P$ , that always stops, and solves the halting problem.

**Theorem (“Halting theorem”)**

*There is no program that always stops and solves the halting problem.*

# Intuition, why solving the Halting problem is really hard

## Definition

*An integer number  $n$  is a weird number if*

- the sum of the proper divisors (including 1 but not itself) of  $n$  the number is  $> n$ ,*
- no subset of those divisors sums to the number itself.*

70 is weird. Its divisors are 1, 2, 5, 7, 10, 14, 35.

$1 + 2 + 5 + 7 + 10 + 14 + 35 = 74$ . No subset of them adds up to 70.



# Intuition, why solving the Halting problem is really hard

## Definition

An integer number  $n$  is a weird number if

- the sum of the proper divisors (including 1 but not itself) of  $n$  the number is  $> n$ ,
- no subset of those divisors sums to the number itself.

70 is weird. Its divisors are 1, 2, 5, 7, 10, 14, 35.

$1 + 2 + 5 + 7 + 10 + 14 + 35 = 74$ . No subset of them adds up to 70.

**Open question:** Are there any odd weird numbers?

# Intuition, why solving the Halting problem is really hard

## Definition

An integer number  $n$  is a weird number if

- the sum of the proper divisors (including 1 but not itself) of  $n$  the number is  $> n$ ,
- no subset of those divisors sums to the number itself.

70 is weird. Its divisors are 1, 2, 5, 7, 10, 14, 35.

$1 + 2 + 5 + 7 + 10 + 14 + 35 = 74$ . No subset of them adds up to 70.

**Open question:** Are there any odd weird numbers?

---

Write a program  $P$  that tries all odd numbers in order, and check if they are weird. The program stops if it found such number.

---

# Intuition, why solving the Halting problem is really hard

## Definition

An integer number  $n$  is a weird number if

- the sum of the proper divisors (including 1 but not itself) of  $n$  the number is  $> n$ ,
- no subset of those divisors sums to the number itself.

70 is weird. Its divisors are 1, 2, 5, 7, 10, 14, 35.

$1 + 2 + 5 + 7 + 10 + 14 + 35 = 74$ . No subset of them adds up to 70.

**Open question:** Are there any odd weird numbers?

---

Write a program  $P$  that tries all odd numbers in order, and check if they are weird. The program stops if it found such number.

---

If can solve halting problem  $\implies$  can resolve this open problem.

## If you can halt, you can prove or disprove anything...

- Consider any math claim  $C$ .
- **Prover** algorithm  $P_C$ :
  - (A) Generate sequence of all possible proofs (sequence of strings) into a pipe/queue.

## If you can halt, you can prove or disprove anything...

- Consider any math claim  $C$ .
- **Prover** algorithm  $P_C$ :
  - (A) Generate sequence of all possible proofs (sequence of strings) into a pipe/queue.
  - (B)  $\langle p \rangle \leftarrow \text{pop top of queue}$ .

## If you can halt, you can prove or disprove anything...

- Consider any math claim  $C$ .
- **Prover** algorithm  $P_C$ :
  - (A) Generate sequence of all possible proofs (sequence of strings) into a pipe/queue.
  - (B)  $\langle p \rangle \leftarrow \text{pop top of queue}$ .
  - (C) Feed  $\langle p \rangle$  and  $\langle C \rangle$ , into a proof verifier (“easy”).

## If you can halt, you can prove or disprove anything...

- Consider any math claim  $C$ .
- **Prover** algorithm  $P_C$ :
  - (A) Generate sequence of all possible proofs (sequence of strings) into a pipe/queue.
  - (B)  $\langle p \rangle \leftarrow$  pop top of queue.
  - (C) Feed  $\langle p \rangle$  and  $\langle C \rangle$ , into a proof verifier (“easy”).
  - (D) If  $\langle p \rangle$  valid proof of  $\langle C \rangle$ , then stop and accept.
  - (E) Go to (B).
- $P_C$  halts  $\iff C$  is true and has a proof.
- If halting is decidable, then can decide if any claim in math is true.

## Turing machines...

TM = Turing machine = program.



## Reminder: Undecidability

### Definition

Language  $L \subseteq \Sigma^*$  is undecidable if no program  $P$ , given  $w \in \Sigma^*$  as input, can **always stop** and output whether  $w \in L$  or  $w \notin L$ .

(Usually defined using **TM** not programs. But equivalent.)

## Reminder: The following language is undecidable.

Decide if given a program  $M$ , and an input  $w$ , does  $M$  accepts  $w$ .  
Formally, the corresponding language is

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

## Reminder: The following language is undecidable.

Decide if given a program  $M$ , and an input  $w$ , does  $M$  accept  $w$ .  
Formally, the corresponding language is

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

### Definition

A decider for a language  $L$ , is a program (or a  $TM$ ) that always stops, and outputs for any input string  $w \in \Sigma^*$  whether or not  $w \in L$ .

A language that has a decider is decidable.

## Reminder: The following language is undecidable.

Decide if given a program  $M$ , and an input  $w$ , does  $M$  accept  $w$ .  
Formally, the corresponding language is

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

### Definition

A decider for a language  $L$ , is a program (or a  $TM$ ) that always stops, and outputs for any input string  $w \in \Sigma^*$  whether or not  $w \in L$ .

A language that has a decider is decidable.

Turing proved the following.

### Theorem

$A_{TM}$  is undecidable.

# The halting problem

---

## $A_{TM}$ is not TM decidable!

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

**Theorem (The halting theorem.)**

$A_{TM}$  is not Turing decidable.

## $A_{TM}$ is not TM decidable!

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

**Theorem (The halting theorem.)**

$A_{TM}$  is not Turing decidable.

**Proof:** Assume  $A_{TM}$  is TM decidable.

## $A_{TM}$ is not TM decidable!

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

**Theorem (The halting theorem.)**

$A_{TM}$  is not Turing decidable.

**Proof:** Assume  $A_{TM}$  is TM decidable.

**Halt:** TM deciding  $A_{TM}$ . **Halt** always halts, and works as follows.

$$\mathbf{Halt}(\langle M, w \rangle) = \begin{cases} \text{accept} & M \text{ accepts } w \\ \text{reject} & M \text{ does not accept } w. \end{cases}$$



# Halting theorem proof continued 1

We build the following new function.

```
Flipper(  $\langle M \rangle$ )  
  res  $\leftarrow$  Halt( $\langle M, \langle M \rangle \rangle$ )  
  if res is accept then  
    reject  
  else  
    accept
```

# Halting theorem proof continued 1

We build the following new function.

```
Flipper(  $\langle M \rangle$  )  
  res  $\leftarrow$  Halt(  $\langle M, \langle M \rangle \rangle$  )  
  if res is accept then  
    reject  
  else  
    accept
```

**Flipper** always stops.

$$\mathbf{Flipper}(\langle M \rangle) = \begin{cases} \text{reject} & M \text{ accepts } \langle M \rangle \\ \text{accept} & M \text{ does not accept } \langle M \rangle. \end{cases}$$

## Halting theorem proof continued 2

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

**Flipper** is a **TM** (duh!), and as such it has an encoding  $\langle \text{Flipper} \rangle$ .

Run **Flipper** on itself.

$$\text{Flipper}(\langle \text{Flipper} \rangle) = \begin{cases} \text{reject} & \text{Flipper accepts } \langle \text{Flipper} \rangle \\ \text{accept} & \text{Flipper does not accept } \langle \text{Flipper} \rangle. \end{cases}$$

## Halting theorem proof continued 2

$$\mathbf{Flipper}(\langle M \rangle) = \begin{cases} \text{reject} & M \text{ accepts } \langle M \rangle \\ \text{accept} & M \text{ does not accept } \langle M \rangle. \end{cases}$$

**Flipper** is a **TM** (duh!), and as such it has an encoding  $\langle \mathbf{Flipper} \rangle$ .  
Run **Flipper** on itself.

$$\mathbf{Flipper}(\langle \mathbf{Flipper} \rangle) = \begin{cases} \text{reject} & \mathbf{Flipper} \text{ accepts } \langle \mathbf{Flipper} \rangle \\ \text{accept} & \mathbf{Flipper} \text{ does not accept } \langle \mathbf{Flipper} \rangle. \end{cases}$$

This is can't be correct.

## Halting theorem proof continued 2

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

**Flipper** is a **TM** (duh!), and as such it has an encoding  $\langle \text{Flipper} \rangle$ .  
Run **Flipper** on itself.

$$\text{Flipper}(\langle \text{Flipper} \rangle) = \begin{cases} \text{reject} & \text{Flipper accepts } \langle \text{Flipper} \rangle \\ \text{accept} & \text{Flipper does not accept } \langle \text{Flipper} \rangle. \end{cases}$$

This is can't be correct.

Assumption that **Halt** exists is false.  $\implies A_{\text{TM}}$  is not **TM**  
decidable. □

**Unrecognizable**

---

## Definition

*Language  $L$  is **TM** decidable if there exists  $M$  that always stops, such that  $L(M) = L$ .*

# TM recognizable

## Definition

Language  $L$  is **TM** decidable if there exists  $M$  that always stops, such that  $L(M) = L$ .

## Definition

Language  $L$  is **TM** recognizable if there exists  $M$  that stops on some inputs, such that  $L(M) = L$ .



# TM recognizable

## Definition

Language  $L$  is **TM** decidable if there exists  $M$  that always stops, such that  $L(M) = L$ .

## Definition

Language  $L$  is **TM** recognizable if there exists  $M$  that stops on some inputs, such that  $L(M) = L$ .

## Theorem (Halting)

$A_{\text{TM}} = \left\{ \langle M, w \rangle \mid M \text{ is a TM and } M \text{ accepts } w \right\}$  is **TM** recognizable, but not decidable.

## Lemma

*If  $L$  and  $\bar{L} = \Sigma^* \setminus L$  are both **TM** recognizable, then  $L$  and  $\bar{L}$  are decidable.*

# TM recognizable

## Lemma

If  $L$  and  $\bar{L} = \Sigma^* \setminus L$  are both **TM** recognizable, then  $L$  and  $\bar{L}$  are decidable.

## Proof.

$M$ : **TM** recognizing  $L$ .

$M_c$ : **TM** recognizing  $\bar{L}$ .

Given input  $x$ , using **UTM** simulating running  $M$  and  $M_c$  on  $x$  in parallel. One of them must stop and accept. Return result.

$\implies L$  is decidable.



## Complement language for $A_{TM}$

$$\overline{A_{TM}} = \Sigma^* \setminus \left\{ \langle M, w \rangle \mid M \text{ is a } TM \text{ and } M \text{ accepts } w \right\}.$$

## Complement language for $A_{TM}$

$$\overline{A_{TM}} = \Sigma^* \setminus \left\{ \langle M, w \rangle \mid M \text{ is a } TM \text{ and } M \text{ accepts } w \right\}.$$

But don't really care about invalid inputs. So, really:

$$\overline{A_{TM}} = \left\{ \langle M, w \rangle \mid M \text{ is a } TM \text{ and } M \text{ does **not** accept } w \right\}.$$

# Complement language for $A_{TM}$ is not TM-recognizable

## Theorem

*The language*

$$\overline{A_{TM}} = \left\{ \langle M, w \rangle \mid M \text{ is a } TM \text{ and } M \text{ does not accept } w \right\}.$$

*is not  $TM$  recognizable.*

# Complement language for $A_{TM}$ is not TM-recognizable

## Theorem

*The language*

$$\overline{A_{TM}} = \left\{ \langle M, w \rangle \mid M \text{ is a } TM \text{ and } M \text{ does not accept } w \right\}.$$

*is not  $TM$  recognizable.*

## Proof.

$A_{TM}$  is  $TM$ -recognizable.

If  $\overline{A_{TM}}$  is  $TM$ -recognizable

# Complement language for $A_{TM}$ is not TM-recognizable

## Theorem

*The language*

$$\overline{A_{TM}} = \left\{ \langle M, w \rangle \mid M \text{ is a } TM \text{ and } M \text{ does not accept } w \right\}.$$

*is not TM recognizable.*

## Proof.

$A_{TM}$  is TM-recognizable.

If  $\overline{A_{TM}}$  is TM-recognizable

$\implies$  (by Lemma)

$A_{TM}$  is decidable. A contradiction.





# Reductions

---

# Reduction

**Meta definition:** Problem **X** reduces to problem **B**, if given a solution to **B**, then it implies a solution for **X**. Namely, we can solve **Y** then we can solve **X**. We will done this by  $\mathbf{X} \implies \mathbf{Y}$ .

# Reduction

**Meta definition:** Problem **X** reduces to problem **B**, if given a solution to **B**, then it implies a solution for **X**. Namely, we can solve **Y** then we can solve **X**. We will done this by  $\mathbf{X} \implies \mathbf{Y}$ .

## Definition

oracle ORAC for language  $L$  is a function that receives as a word  $w$ , returns **TRUE**  $\iff w \in L$ .

# Reduction

**Meta definition:** Problem **X** reduces to problem **B**, if given a solution to **B**, then it implies a solution for **X**. Namely, we can solve **Y** then we can solve **X**. We will done this by  $X \implies Y$ .

## Definition

oracle ORAC for language  $L$  is a function that receives as a word  $w$ , returns **TRUE**  $\iff w \in L$ .

## Lemma

A language  $X$  reduces to a language  $Y$ , if one can construct a **TM** decider for  $X$  using a given oracle  $\text{ORAC}_Y$  for  $Y$ .

We will denote this fact by  $X \implies Y$ .

## Reduction proof technique

- **Y**: Problem/language for which we want to prove undecidable.

# Reduction proof technique

- **Y**: Problem/language for which we want to prove undecidable.
- Proof via reduction. Result in a proof by contradiction.

# Reduction proof technique

- **Y**: Problem/language for which we want to prove undecidable.
- Proof via reduction. Result in a proof by contradiction.
- $L$ : language of **Y**.

# Reduction proof technique

- **Y**: Problem/language for which we want to prove undecidable.
- Proof via reduction. Result in a proof by contradiction.
- $L$ : language of **Y**.
- Assume  $L$  is decided by **TM**  $M$ .



# Reduction proof technique

- **Y**: Problem/language for which we want to prove undecidable.
- Proof via reduction. Result in a proof by contradiction.
- $L$ : language of **Y**.
- Assume  $L$  is decided by **TM**  $M$ .
- Create a decider for known undecidable problem **X** using  $M$ .

# Reduction proof technique

- **Y**: Problem/language for which we want to prove undecidable.
- Proof via reduction. Result in a proof by contradiction.
- $L$ : language of **Y**.
- Assume  $L$  is decided by **TM**  $M$ .
- Create a decider for known undecidable problem **X** using  $M$ .
- Result in decider for **X** (i.e.,  $A_{\text{TM}}$ ).

# Reduction proof technique

- **Y**: Problem/language for which we want to prove undecidable.
- Proof via reduction. Result in a proof by contradiction.
- $L$ : language of **Y**.
- Assume  $L$  is decided by **TM**  $M$ .
- Create a decider for known undecidable problem **X** using  $M$ .
- Result in decider for **X** (i.e.,  $A_{\text{TM}}$ ).
- Contradiction **X** is not decidable.

# Reduction proof technique

- **Y**: Problem/language for which we want to prove undecidable.
- Proof via reduction. Result in a proof by contradiction.
- $L$ : language of **Y**.
- Assume  $L$  is decided by **TM**  $M$ .
- Create a decider for known undecidable problem **X** using  $M$ .
- Result in decider for **X** (i.e.,  $A_{\text{TM}}$ ).
- Contradiction **X** is not decidable.
- Thus,  $L$  must be not decidable.

## Reduction implies decidability

### Lemma

*Let  $X$  and  $Y$  be two languages, and assume that  $X \implies Y$ . If  $Y$  is decidable then  $X$  is decidable.*

### Proof.

Let  $T$  be a decider for  $Y$  (i.e., a program or a **TM**). Since  $X$  reduces to  $Y$ , it follows that there is a procedure  $T_{X|Y}$  (i.e., decider) for  $X$  that uses an oracle for  $Y$  as a subroutine. We replace the calls to this oracle in  $T_{X|Y}$  by calls to  $T$ . The resulting program  $T_X$  is a decider and its language is  $X$ . Thus  $X$  is decidable (or more formally **TM** decidable). □

## The contrapositive...

### **Lemma**

*Let  $X$  and  $Y$  be two languages, and assume that  $X \implies Y$ . If  $X$  is undecidable then  $Y$  is undecidable.*

# Halting

---

# The halting problem

Language of all pairs  $\langle M, w \rangle$  such that  $M$  halts on  $w$ :

$$A_{\text{Halt}} = \left\{ \langle M, w \rangle \mid M \text{ is a } TM \text{ and } M \text{ stops on } w \right\}.$$

Similar to language already known to be undecidable:

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



## On way to proving that Halting is undecidable...

### Lemma

*The language  $A_{TM}$  reduces to  $A_{Halt}$ . Namely, given an oracle for  $A_{Halt}$  one can build a decider (that uses this oracle) for  $A_{TM}$ .*

# On way to proving that Halting is undecidable...

## Proof.

Let  $\text{ORAC}_{\text{Halt}}$  be the given oracle for  $A_{\text{Halt}}$ . We build the following decider for  $A_{\text{TM}}$ .

```
AnotherDecider- $A_{\text{TM}}(\langle M, w \rangle)$   
   $\text{res} \leftarrow \text{ORAC}_{\text{Halt}}(\langle M, w \rangle)$   
  // if  $M$  does not halt on  $w$  then reject.  
  if  $\text{res} = \text{reject}$  then  
    halt and reject.  
  //  $M$  halts on  $w$  since  $\text{res} = \text{accept}$ .  
  // Simulating  $M$  on  $w$  terminates in finite time.  
   $\text{res}_2 \leftarrow \text{Simulate } M \text{ on } w$ .  
  return  $\text{res}_2$ .
```

This procedure always return and as such its a decider for  $A_{\text{TM}}$ .



# The Halting problem is not decidable

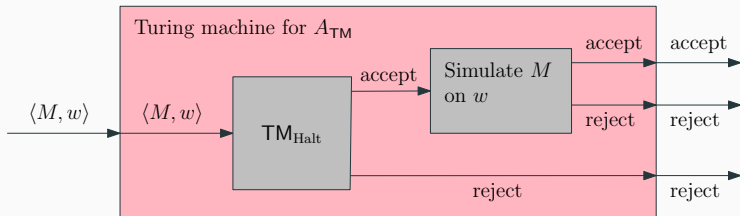
## Theorem

*The language  $A_{\text{Halt}}$  is not decidable.*

## Proof.

Assume, for the sake of contradiction, that  $A_{\text{Halt}}$  is decidable. As such, there is a TM, denoted by  $TM_{\text{Halt}}$ , that is a decider for  $A_{\text{Halt}}$ . We can use  $TM_{\text{Halt}}$  as an implementation of an oracle for  $A_{\text{Halt}}$ , which would imply that one can build a decider for  $A_{TM}$ . However,  $A_{TM}$  is undecidable. A contradiction. It must be that  $A_{\text{Halt}}$  is undecidable. □

## The same proof by figure...



... if  $A_{Halt}$  is decidable, then  $A_{TM}$  is decidable, which is impossible.

**More reductions next time**

---