CS 420 / CS 620

Reducibility

Wednesday, November 19, 2025

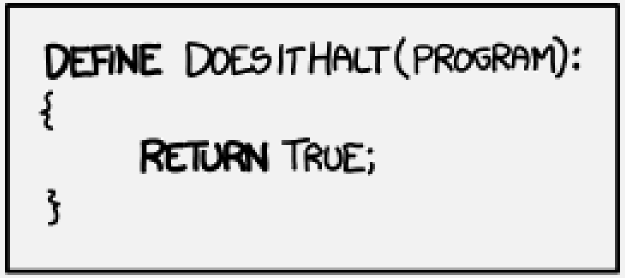
UMass Boston Computer Science

Announcements

- HW 11
  - Out: **Mon 11/17 12pm (noon)**
  - Due: **Mon 11/24 12pm (noon)**

- HW 12
  - Out: **Mon 11/24 12pm (noon)**
  - **Thanksgiving: 11/26-11/30**
  - Due: **Fri 12/5 12pm (noon)**

- HW 13
  - Out: **Fri 12/5 12pm (noon)**
  - Due: **Fri 12/12 12pm (noon)** (classes end)
  - Late due: **Mon 12/15 12pm (noon)** (exams start)
    - **Nothing accepted after this** (please don't ask)



THE BIG PICTURE SOLUTION TO THE HALTING PROBLEM

Last Time

Diagonalization with Turing Machines

Diagonal: Result of **Giving a TM its own Encoding** as **Input**

All TM Encodings

	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\cdots$	$\langle D \rangle$	$\cdots$
M_1	<u>*accept*</u>	*reject*	*accept*	*reject*		*accept*	
M_2	*accept*	<u>*accept*</u>	*accept*	*accept*	$\cdots$	*accept*	$\cdots$
M_3	*reject*	*reject*	<u>*reject*</u>	*reject*		*reject*	
M_4	*accept*	*accept*	*reject*	<u>*reject*</u>		*accept*	
$\vdots$			$\vdots$		$\ddots$		
~~D~~	~~*reject*~~	~~*reject*~~	~~*accept*~~	~~*accept*~~		<u>?</u>	
$\vdots$			.				

opposites

All TMs

Try to construct this: **"opposite" TM D**

TM D can't exist!

What should happen here?

It must both accept and reject!

Last Time

3 Easy Steps!

Thm: A_{TM} is undecidable

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

Proof by contradiction:

1. Assume A_{TM} is decidable. So there exists a decider H for it:

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

2. Use H to define another TM … the impossible "opposite" machine:

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. That is, if H accepts, *reject*; and if H rejects, *accept*."

(does opposite of what input TM would do if given itself)

H computes: M's result with itself as input

Do the opposite

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

(from prev slide)
This TM can't be defined!

Last Time

3 Easy Steps!

Thm: A_{TM} is undecidable

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

Proof by contradiction:

This cannot be true

1. **Assume** A_{TM} **is decidable.** So **there exists a decider** H **for it:**

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

2. **Use** H to **define another TM …** the **impossible "opposite" machine:**

$D =$ "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. That is, if H accepts, *reject*; and if H rejects, *accept*."

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

TM D can't be defined!

3. But D **does not exist! Contradiction!** So **the assumption is false.**

Last Time

Easier Undecidability Proofs

- **We proved** $A_{\mathsf{TM}} = \{\langle M, w\rangle \mid M \text{ is a TM and } M \text{ accepts } w\}$ **undecidable …**
- … by **contradiction:**
  - <u>Use</u> hypothetical A_{TM} decider to create an <u>impossible</u> decider "D"!

reduce "D problem" to A_{TM}

- Step # 1: **coming up with "D"** --- <u>hard!</u>
  - Need to invent **diagonalization**

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

Unknown lang

Known undecidable lang!

- Step # 2: **reduce** "~~D~~" **problem** to ~~A_{TM}~~ --- <u>easier!</u>

- <u>From now on</u>: **undecidability proofs** <u>**only need** step # 2</u>!
  - And we now **have two "impossible" problems to choose from**

Let's add more!

The Halting Problem

$HALT_{\mathsf{TM}} = \{\langle M, w\rangle \mid M \text{ is a TM and } M \text{ halts on input } w\}$

Thm: $HALT_{\mathsf{TM}}$ is undecidable

Proof, by **contradiction**:

FAQ: "**Do we need Examples Table?**"

(hypothetical)

- Assume: $HALT_{\mathsf{TM}}$ has *decider* R; use it to create decider for A_{TM}:

(undecidable, no decider) $A_{\mathsf{TM}} = \{\langle M, w\rangle \mid M \text{ is a TM and } M \text{ accepts } w\}$

A: Yes, to **Justify** a **Statement** like "**machine** X **decides lang** L"

$S =$ "On input $\langle M, w\rangle$, an encoding of a TM M and a string w:

(even if it leads to contradiction)

1. Run TM R on input $\langle M, w\rangle$.
2. If R rejects, *reject*.
3. If R accepts, simulate M on w until it halts.
4. If M has accepted, *accept*; if M h

ALSO: **Example Table**(s) tell you how to **solve the problem**!

Examples also help to **understand the problem** (needed before solving)

The Halting Problem

$HALT_{\mathsf{TM}} = \{\langle M, w\rangle \mid M \text{ is a TM and } M \text{ halts on input } w\}$

Thm: $HALT_{\mathsf{TM}}$ is undecidable

Proof, by **contradiction**:

- Assume: $HALT_{\mathsf{TM}}$ has *decider* R; use it to create decider for A_{TM}:

Examples for R

Input $\langle M, w\rangle$ will be $\langle M_i, w_i\rangle$ where:
- M_i is some **TM** described in table and
- w_i is some **string**

String	M_i on w_i	R on $\langle M, w\rangle$	In lang $HALT_{\mathsf{TM}}$?
$\langle M_1, w_1\rangle$	(halt and) Accept	Accept	Yes
$\langle M_2, w_2\rangle$	(halt and) Reject	Accept	Yes
$\langle M_3, w_3\rangle$	Loop	Reject	No

R lets us know when **a TM would loop on some input** (without running the TM) … so we can **avoid it!**

The Halting Problem

$HALT_{\mathsf{TM}} = \{\langle M, w\rangle |\ M \text{ is a TM and } M \text{ halts on input } w\}$

Thm: $HALT_{\mathsf{TM}}$ is undecidable

Proof, by **contradiction**:

Using our hypothetical $HALT_{\mathsf{TM}}$ decider R

Loop is not accept

- Assume: $HALT_{\mathsf{TM}}$ has *decider* R; use it to create decider for A_{TM}:

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

S = "On input $\langle M, w\rangle$, an encoding of a TM M and a string w:

1. Run TM R on input $\langle M, w\rangle$.
2. If R rejects, *reject*.
3. If R accepts, simulate M on w until it halts.
4. If M has accepted, *accept*; if M has rejected, *reject*."

(doesn't accept)

If R rejects $\langle M,w\rangle$, M loops on w, so S should reject it

This step always halts

Examples Table??

Termination argument:
Step 1: R is a decider so always halts
Step 3: M always halts because R said so

The Halt[ing Problem]

Need help from ($HALT_{TM}$ **decider**) R to **decide** A_{TM}!!

These must match (like before)

BUT A_{TM} **undecidable!**
Cant actually **compute this!**

$HALT_{TM} = \{\langle M, w \rangle \mid M$ is a TM and M halts on input $w\}$

Input $\langle M, w\rangle = \langle M_i, w_i\rangle$ where:
- M_i is **TM** described in table
- w_i is some **string**

String	M_i on w_i	$HALT_{TM}$ decider R on $\langle M, w\rangle$	A_{TM} decider S on $\langle M, w\rangle$	In lang A_{TM}?
$\langle M_1, w_1\rangle$	Accept	Accept	Accept	Yes
$\langle M_2, w_2\rangle$	Reject	Accept	Reject	No
$\langle M_3, w_3\rangle$	Loop	Reject	Reject	No

Loop is not accept

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

Need to use this …

… to figure out this

Thm: H[ALT_TM is undecidable]

Proof, b[y contradiction]

- Assum[e HALT_TM has decider R; use it to create decider for A_TM:]

$S =$ "On input $\langle M, w \rangle$, an encoding of a TM M and a string w:
1. Run TM R on input $\langle M, w \rangle$.
2. If R rejects, *reject*.
3. If R accepts, simulate M on w until it halts.
4. If M has accepted, *accept*; if M has rejected, *reject*."

Example Table
Justifying Statement
"S **decides** A_{TM}"

The Halting Problem

Undecidability Proof Technique #1:
Reduce from **known undecidable language** (by **creating its decider**)

$HALT_{\mathsf{TM}} = \{\langle M, w\rangle \mid M \text{ is a TM and } M \text{ halts on input } w\}$

Thm: $HALT_{TM}$ is undecidable

Proof, by **contradiction**:

FFFAQ: "Do we need S/J????"

- Assume: $HALT_{TM}$ has *decider* R; use it to create decider for A_{TM}:

"You never showed us how????"

S = "On input $\langle M, w\rangle$, an encoding of a TM M and a string w:

1. Run TM R on input $\langle M, w\rangle$.
2. If R rejects, *reject*.
3. If R accepts, simulate M on w until it halts.
4. If M has accepted, *accept*; if M has rejected, *reject*."

Now we have three known undecidable langs, i.e., three "impossible" deciders, to choose from

- But A_{TM} is undecidable (has no decider)! I.e., this decider does not exist!
  - So $HALT_{TM}$ is also undecidable!

The Halting Problem … As Statements / Justifications

$HALT_{\mathsf{TM}} = \{\langle M, w\rangle \mid M \text{ is a TM and } M \text{ halts on input } w\}$

(Proof by contradiction)

Statements	**Justifications**
1. $HALT_{\mathsf{TM}}$ is decidable	1. Opposite of statement to prove
2. $HALT_{\mathsf{TM}}$ has decider R	2. Definition of decidable langs
3. Construct decider S using R ("see below")	3. Definition of TMs and deciders (incl termination argument)
4. Decider S decides A_{TM}	4. See Examples Table
5. A_{TM} is undecidable (i.e, it has no decider)	5. Theorem from last lecture (Sipser Theorem 4.11)
6. $HALT_{\mathsf{TM}}$ is undecidable	6. Contradiction of Stmts #4 & #5

Summary: The Limits of Algorithms

- $A_{\text{DFA}} = \{\langle B, w\rangle \mid B \text{ is a DFA that accepts input string } w\}$ Decidable
- $A_{\text{CFG}} = \{\langle G, w\rangle \mid G \text{ is a CFG that generates string } w\}$ Decidable
- $A_{\text{TM}} = \{\langle M, w\rangle \mid M \text{ is a TM and } M \text{ accepts } w\}$ **Undecidable**
- $HALT_{\text{TM}} = \{\langle M, w\rangle \mid M \text{ is a TM and } M \text{ halts on input } w\}$ **Undecidable**

Similar languages

It's straightforward to use hypothetical $HALT_{\text{TM}}$ decider to create A_{TM} decider

Summary: The Limits of Algorithms

$A_{\text{DFA}} = \{\langle B, w\rangle \mid B \text{ is a DFA that accepts input string } w\}$	Decidable
$A_{\text{CFG}} = \{\langle G, w\rangle \mid G \text{ is a CFG that generates string } w\}$	Decidable
$A_{\text{TM}} = \{\langle M, w\rangle \mid M \text{ is a TM and } M \text{ accepts } w\}$	**Undecidable**
$HALT_{\text{TM}} = \{\langle M, w\rangle \mid M \text{ is a TM and } M \text{ halts on input } w\}$	**Undecidable**
$E_{\text{DFA}} = \{\langle A\rangle \mid A \text{ is a DFA and } L(A) = \emptyset\}$	Decidable
$E_{\text{CFG}} = \{\langle G\rangle \mid G \text{ is a CFG and } L(G) = \emptyset\}$	Decidable
next $E_{\text{TM}} = \{\langle M\rangle \mid M \text{ is a TM and } L(M) = \emptyset\}$	**Undecidable**

Not as similar languages

How can we **use a hypothetical E_{TM} decider** to **create A_{TM} or $HALT_{\text{TM}}$ decider?**

Reducibility: Modifying the TM

$E_{\mathsf{TM}} = \{\langle M \rangle \mid M \text{ is a TM and } L(M) = \emptyset\}$

Thm: E_{TM} is undecidable

Proof, by **contradiction**:

- Assume E_{TM} **has** ***decider*** R**; use it to create** ***decider*** **for** A_{TM}**:**

  $S =$ "On input $\langle M, w \rangle$, an encoding of a TM M and a string w:

  - Run R on input $\langle M \rangle$
  - If R accepts, *reject* (because it means $\langle M \rangle$ doesn't accept anything)
  - if R rejects, **then** $\boxed{\textbf{???}}$ ($\langle M \rangle$ accepts something, $\boxed{\text{but is it } w\text{???}}$)

- Idea: Use **Examples (Table)** for guidance!

  (Will tell us how to solve the problem!)

A_{TM} Examples Table

- Assume E_{TM} has *decider* R; use it to create *decider* for A_{TM}

S = “On input $\langle M, w \rangle$, an e

Remember:
A_{TM} undecidable
(has no decider)!

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

So cannot compute this
… without “help”, i.e., R

Input <M, w>	R result on <M_1>	Want R result to tell us…	S on <M, w>	$\in A_{TM}$?
<M', w'>, where M' not accept w'	accept, means: $L(M_1)=\emptyset$,	M not accept w	reject	no
<M'', w''>, where M'' accept w''	reject, means: $L(M_1)\neq\emptyset$,	M accept w	accept	yes

- Idea: Use **Examples (Table)** for guidance!

$E_{\mathsf{TM}} = \{\langle M \rangle \mid M \text{ is a TM and } L(M) = \emptyset\}$

R is E_{TM} **decider** so:

- Accept $\boxed{M_1}$ means: $L(M_1)=\emptyset$
- *Not accept* M_1 means: $L(M_1)\neq\emptyset$

- Assume E_{TM} **has *decider* R**; use it to create *decider* for A_{TM}

$S =$ "On input $\langle M, w\rangle$, an en[coding of a TM] M and a string w:

So cannot compute this … without "help", i.e., R

Input <M, w>	R result on <M_1>	Want R result to tell us…	S on <M, w>	$\in A_{\mathsf{TM}}$?
<M', w'>, where M' not [accept w']	accept, means: $L(M_1)=\emptyset$, implies	M not accept w	reject	no
<M'', w''>, where M'' accept w''	reject, means: $L(M_1)\neq\emptyset$, implies	M accept w	accept	yes

Want:

IDEA: **modify M** (into M_1) so R gives the **needed information**

$M_1 =$ "On input x:

1. If $x \neq w$, *reject*.
2. If $x = w$, run M on input w and *accept* if M does."

Want:

- M_1 accept nothing, implies M **not accept** w — $L(M_1)=\emptyset$
- M_1 accept something, implies M **accept** w — $L(M_1)\neq\emptyset$

- Assume E_{TM} has *decider* R; use it to create *decider* for A_{TM}

S = "On input $\langle M, w\rangle$, an encoding of a TM M and a string w:

Input <M, w>	R result on <M_1>	Want R result to tell us…	S on <M, w>	$\in A_{TM}$?
<M', w'>, where M' not acce…	accept, means: $L(M_1)=\emptyset$, implies	M not accept w	reject	no
<M'', w''>, where M'' accept w''	reject, means: $L(M_1)\neq\emptyset$, implies	M accept w	accept	yes

☑

- Idea: Create **Examples (Table)** for guid…

M_1 = "On input x:

(almost nothing)

1. If $x \neq w$, *reject*.

2. If $x = w$, run M on input w and *accept* if M does."

~~Want:~~ Got: $L(M_1)=\emptyset$

- M_1 accept nothing, implies M **not accept** w
- M_1 accept ~~something~~ w, implies M **accept** w

$L(M_1) \neq \emptyset = \{w\}$

(nothing or just w)

$$E_{TM} = \{\langle M\rangle \mid M \text{ is a TM and } L(M) = \emptyset\}$$

Thm: E_{TM} is undecidable

Proof, by contradiction:

- Assume E_{TM} has *decider* R; use it to create *decider* for A_{TM}:

S = "On input $\langle M, w\rangle$, an encoding of a TM M and a string w:

First, construct M_1

- Run R on input $\langle M_1\rangle$
- If R accepts, *reject* (because it means $\langle M\rangle$ doesn't accept w)
- if R rejects, then *accept* ($\langle M\rangle$ accepts something, and it is w !)

Note: M_1 is only used to **get needed info from** R; (never run!)

M_1 = "On input x:

1. If $x \neq w$, *reject*.
2. If $x = w$, run M on input w and *accept* if M does."

Got: $L(M_1)=\emptyset$

- M_1 accept nothing, implies M **not accept** w
- M_1 accept w, implies M **accept** w

$L(M_1) \neq \emptyset = \{w\}$

Undecidability Proof Technique #2

Reducibility: Modifying the TM

$E_{\mathsf{TM}} = \{\langle M\rangle \mid M \text{ is a TM and } L(M) = \emptyset\}$

<u>Thm</u>: E_{TM} is undecidable

<u>Proof</u>, by **contradiction**:

Contradiction because: A_{TM} **is undecidable** and **has no decider!**

- Assume E_{TM} has *decider* R; use it to create *decider* for A_{TM}:

S = "On input $\langle M, w\rangle$, an encoding of a TM M and a string w:

<u>First</u>, construct M_1

- Run R on input $\langle M_1\rangle$
- If R accepts, *reject* (because it means $\langle M\rangle$ doesn't accept w)
- if R rejects, then *accept* ($\langle M\rangle$ accepts something, and it is w !)

- <u>Idea</u>: Wrap $\langle M\rangle$ in a new TM that <u>can *only* accept *w*:</u>

M_1 = "On input x:
1. If $x \neq w$, *reject*.
2. If $x = w$, run M on input w and *accept* if M does."

Summary: The Limits of Algorithms

- $A_{\mathsf{DFA}} = \{\langle B, w\rangle \mid B \text{ is a DFA that accepts input string } w\}$ — Decidable
- $A_{\mathsf{CFG}} = \{\langle G, w\rangle \mid G \text{ is a CFG that generates string } w\}$ — Decidable
- $A_{\mathsf{TM}} = \{\langle M, w\rangle \mid M \text{ is a TM and } M \text{ accepts } w\}$ — **Undecidable**
- $E_{\mathsf{DFA}} = \{\langle A\rangle \mid A \text{ is a DFA and } L(A) = \emptyset\}$ — Decidable
- $E_{\mathsf{CFG}} = \{\langle G\rangle \mid G \text{ is a CFG and } L(G) = \emptyset\}$ — Decidable
- $E_{\mathsf{TM}} = \{\langle M\rangle \mid M \text{ is a TM and } L(M) = \emptyset\}$ — **Undecidable**

needs

- $EQ_{\mathsf{DFA}} = \{\langle A, B\rangle \mid A \text{ and } B \text{ are DFAs and } L(A) = L(B)\}$ — Decidable
- $EQ_{\mathsf{CFG}} = \{\langle G, H\rangle \mid G \text{ and } H \text{ are CFGs and } L(G) = L(H)\}$ — **Undecidable**

next

- $EQ_{\mathsf{TM}} = \{\langle M_1, M_2\rangle \mid M_1 \text{ and } M_2 \text{ are TMs and } L(M_1) = L(M_2)\}$ — **Undecidable**

Undecidability Proof Technique #3

Reduce to something else: EQ_{TM} is undecidable

$EQ_{\mathsf{TM}} = \{\langle M_1, M_2 \rangle |\ M_1 \text{ and } M_2 \text{ are TMs and } L(M_1) = L(M_2)\}$

Proof, by **contradiction**:

- Assume: EQ_{TM} has *decider* R; use it to create *decider* for ~~A_{TM}~~ E_{TM}:

$E_{\mathsf{TM}} = \{\langle M \rangle |\ M \text{ is a TM and } L(M) = \emptyset\}$

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

1. Run R on input $\langle M, M_1 \rangle$, where M_1 is a TM that rejects all inputs.
2. If R accepts, *accept*; if R rejects, *reject*."

Reduce to something else: EQ_{TM} is undecidable

$EQ_{\mathsf{TM}} = \{\langle M_1, M_2 \rangle |\ M_1 \text{ and } M_2 \text{ are TMs and } L(M_1) = L(M_2)\}$

Proof, by **contradiction**:

- Assume: EQ_{TM} has *decider* R; use it to create *decider* for E_{TM}:

$= \{\langle M \rangle |\ M \text{ is a TM and } L(M) = \emptyset\}$

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

1. Run R on input $\langle M, M_1 \rangle$, where M_1 is a TM that rejects all inputs.
2. If R accepts, *accept*; if R rejects, *reject*."

- But E_{TM} is undecidable!

Summary: Undecidability Proof Techniques

- Proof Technique #1:
  - Use hypothetical decider to implement impossible A_{TM} decider
  - Example Proof: $HALT_{\mathsf{TM}} = \{\langle M, w\rangle \mid M \text{ is a TM and } M \text{ halts on input } w\}$

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

Reduce

- Proof Technique #2:
  - Use hypothetical decider to implement impossible A_{TM} decider
  - But first modify the input M
  - Example Proof: $E_{\mathsf{TM}} = \{\langle M\rangle \mid M \text{ is a TM and } L(M) = \emptyset\}$

Reduce

Can also combine these techniques

- Proof Technique #3:
  - Use hypothetical decider to implement non-A_{TM} impossible decider
  - Example Proof: $EQ_{\mathsf{TM}} = \{\langle M_1, M_2\rangle \mid M_1 \text{ and } M_2 \text{ are TMs and } L(M_1) = L(M_2)\}$

Summary: Decidability and Undecidability

$A_{\textsf{DFA}} = \{\langle B, w\rangle \mid B \text{ is a DFA that accepts input string } w\}$	Decidable
$A_{\textsf{CFG}} = \{\langle G, w\rangle \mid G \text{ is a CFG that generates string } w\}$	Decidable
$A_{\textsf{TM}} = \{\langle M, w\rangle \mid M \text{ is a TM and } M \text{ accepts } w\}$	**Undecidable**
$E_{\textsf{DFA}} = \{\langle A\rangle \mid A \text{ is a DFA and } L(A) = \emptyset\}$	Decidable
$E_{\textsf{CFG}} = \{\langle G\rangle \mid G \text{ is a CFG and } L(G) = \emptyset\}$	Decidable
$E_{\textsf{TM}} = \{\langle M\rangle \mid M \text{ is a TM and } L(M) = \emptyset\}$	**Undecidable**
$EQ_{\textsf{DFA}} = \{\langle A, B\rangle \mid A \text{ and } B \text{ are DFAs and } L(A) = L(B)\}$	Decidable
$EQ_{\textsf{CFG}} = \{\langle G, H\rangle \mid G \text{ and } H \text{ are CFGs and } L(G) = L(H)\}$	**Undecidable**
$EQ_{\textsf{TM}} = \{\langle M_1, M_2\rangle \mid M_1 \text{ and } M_2 \text{ are TMs and } L(M_1) = L(M_2)\}$	**Undecidable**

Also Undecidable …

next

- $REGULAR_{\mathsf{TM}}$ = {<M> | M is a TM and $L(M)$ is a regular language}

Undecidability Proof Technique #2:
Modify input TM M

Thm: $REGULAR_{\mathsf{TM}}$ is undecidable

$REGULAR_{\mathsf{TM}} = \{\langle M \rangle \mid M \text{ is a TM and } L(M) \text{ is a regular language}\}$

Proof, by **contradiction**:

- Assume: $REGULAR_{\mathsf{TM}}$ has *decider* R; use it to create *decider* for A_{TM}:

$S =$ "On input $\langle M, w \rangle$, an encoding of a TM M and a string w:

- First, construct M_2 (??)
- Run R on input $\langle M_2 \rangle$
- If R accepts, *accept*; if R rejects, *reject*

Want: $L(M_2)$ =
- **regular**, if M accepts w
- **nonregular**, if M does not accept w

<u>Thm</u>: $REGULAR_{\mathsf{TM}}$ is undecidable (continued)

$REGULAR_{\mathsf{TM}} = \{\langle M \rangle \mid M \text{ is a TM and } L(M) \text{ is a regular language}\}$

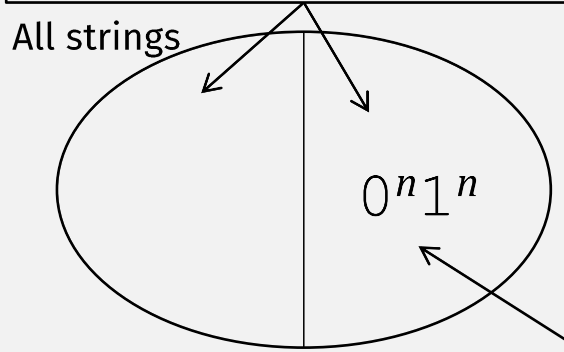
$M_2 =$ "On input x:

1. If x has the form 0^n1^n, *accept*.
2. If x does not have this form, run M on input w and *accept* if M accepts w."

Always accept strings 0^n1^n
$L(M_2)$ = **nonregular**, so far

If M accepts w,
accept everything else,
so $L(M_2) = \Sigma^* =$ **regular**

if M <u>does not</u> accept w, M_2 accepts all strings (**regular** lang)



if M accepts w, M_2 accepts this **nonregular** lang

<u>Want</u>: $L(M_2)$ =

- **regular**, if M accepts w
- **nonregular**, if M does not accept w

Also Undecidable …

- $REGULAR_{\mathrm{TM}}$ = {<M> | M is a TM and $L(M)$ is a regular language}
- $CONTEXTFREE_{\mathrm{TM}}$ = {<M> | M is a TM and $L(M)$ is a CFL}
- $DECIDABLE_{\mathrm{TM}}$ = {<M> | M is a TM and $L(M)$ is a decidable language}
- $FINITE_{\mathrm{TM}}$ = {<M> | M is a TM and $L(M)$ is a finite language}

Seems like:
no algorithm can compute anything about …
… the **language of a Turing Machine,**
i.e., about the **runtime behavior of programs** …

An Algorithm About Program Behavior?

```
main()
{
    printf("hello, world\n");
}
```

Write a program that,
given **another program** as its argument,
returns TRUE if that argument prints
“**`hello, world`**”

TRUE

Seems like:
no algorithm can compute anything about …
… the **language of a Turing Machine,**
i.e., about the **runtime behavior of programs** …

Fermat's Last Theorem
(unknown for ~350 years,
solved in 1990s)

```
main()
{
  If x^n + y^n = z^n, for any integer n > 2
    printf("hello, world\n");
}
```

Write a program that,
given another program as its argument,
returns TRUE if that argument prints
“`hello, world`”

?????

Seems like:
no algorithm can compute anything about …
… the **language of a Turing Machine,**
i.e., about the **runtime behavior of programs** …

Also Undecidable …

- $REGULAR_{\mathsf{TM}}$ = {<M> | M is a TM and $L(M)$ is a regular language}
- $CONTEXTFREE_{\mathsf{TM}}$ = {<M> | M is a TM and $L(M)$ is a CFL}
- $DECIDABLE_{\mathsf{TM}}$ = {<M> | M is a TM and $L(M)$ is a decidable language}
- $FINITE_{\mathsf{TM}}$ = {<M> | M is a TM and $L(M)$ is a finite language}
- …

Rice's Theorem

- $ANYTHING_{\mathsf{TM}}$ = {<M> | M is a TM and "… **anything** …" about $L(M)$}

Rice's Theorem: $ANYTHING_{\mathrm{TM}}$ is Undecidable

$ANYTHING_{\mathrm{TM}}$ = {<M> | M is a TM and … **anything** … about $L(M)$}

- "… **Anything** …", more precisely:

  For any M_1, M_2,
  - if $L(M_1) = L(M_2)$
  - then $M_1 \in ANYTHING_{\mathrm{TM}} \Leftrightarrow M_2 \in ANYTHING_{\mathrm{TM}}$

- Also, "… **Anything** …"must be "non-trivial":
  - $ANYTHING_{\mathrm{TM}}$!= {}
  - $ANYTHING_{\mathrm{TM}}$!= set of all TMs

Rice's Theorem: $ANYTHING_{\mathrm{TM}}$ is Undecidable

$ANYTHING_{\mathrm{TM}}$ = {<M> | M is a TM and … **anything** … about $L(M)$}

Proof by **contradiction**

- Assume **some language satisfying $ANYTHING_{\mathrm{TM}}$ has a decider R.**
  - Since $ANYTHING_{\mathrm{TM}}$ **is non-trivial,** then **there exists** $M_{ANY} \in ANYTHING_{\mathrm{TM}}$
  - Where R **accepts** M_{ANY}
- Use R **to create decider for** A_{TM}:

On input <M, w>:

- **Create** M_w:

  M_w = on input x:
  - Run M on w
    - If M rejects w: reject x
    - If M accepts w:
      Run M_{ANY} on x and accept if it accepts, else reject

  If M accepts w: M_w = M_{ANY}
  If M doesn't accept w: M_w accepts nothing

- Run R on M_w
  - If it accepts, then M_w = M_{ANY}, so M accepts w, so accept
  - Else reject

These two cases must be different, (so R can distinguish when M accepts w)

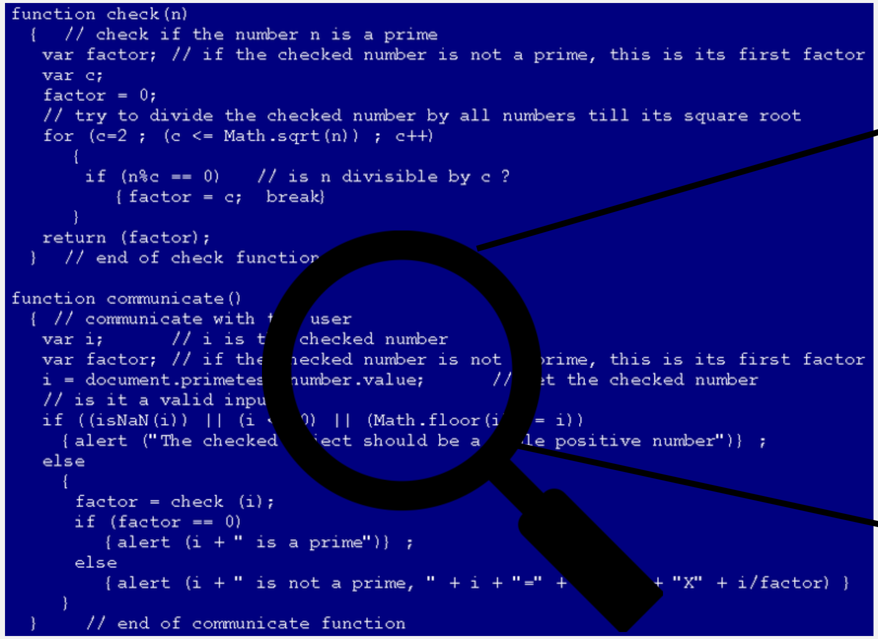
Wait! What if the TM that accepts nothing is in $ANYTHING_{\mathrm{TM}}$!

Proof still works! Just use the complement of $ANYTHING_{\mathrm{TM}}$ instead!

Rice’s Theorem Implication

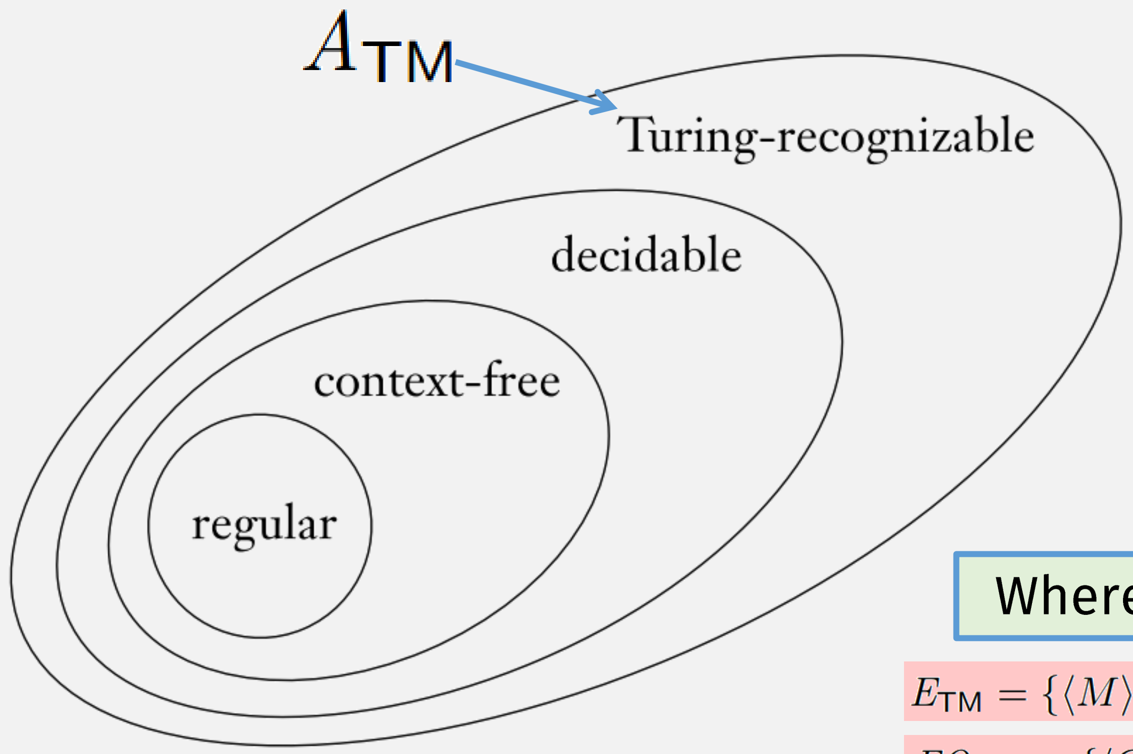
{<M> | M is a TM that installs malware}

Undecidable!
(by Rice’s Theorem)



Turing Unrecognizable?

Is there anything out here?



Where do these go?

$E_{\mathsf{TM}} = \{\langle M \rangle \mid M \text{ is a TM and } L(M) = \emptyset\}$

$EQ_{\mathsf{CFG}} = \{\langle G, H \rangle \mid G \text{ and } H \text{ are CFGs and } L(G) = L(H)\}$

$EQ_{\mathsf{TM}} = \{\langle M_1, M_2 \rangle \mid M_1 \text{ and } M_2 \text{ are TMs and } L(M_1) = L(M_2)\}$

Thm: Some langs are not Turing-recognizable

Proof: requires 2 lemmas

- Lemma 1: The **set of all languages** is *uncountable*
  - Proof: Show there is a bijection with another uncountable set …
    - … The set of all infinite binary sequences
- Lemma 2: The **set of all TMs** is *countable*
- Therefore, some language is not recognized by a TM

Mapping a Language to a Binary Sequence

All Possible Strings

$$\Sigma^* = \{\ \varepsilon,\quad 0,\quad 1,\quad 00,\quad 01,\quad 10,\quad 11,\quad 000,\ 001,\quad \cdots\ \}$$

Some Language (subset of above)

$$A = \{\quad 0,\quad 00,\quad 01,\quad 000,\ 001,\quad \cdots\ \}$$

Its (unique) Binary Sequence

$$\chi_A = \quad 0 \quad 1 \quad 0 \quad 1 \quad 1 \quad 0 \quad 0 \quad 1 \quad 1 \quad \cdots$$

Each digit represents one possible string:
- 1 if lang has that string,
- 0 otherwise

Thm: Some langs are not Turing-recognizable

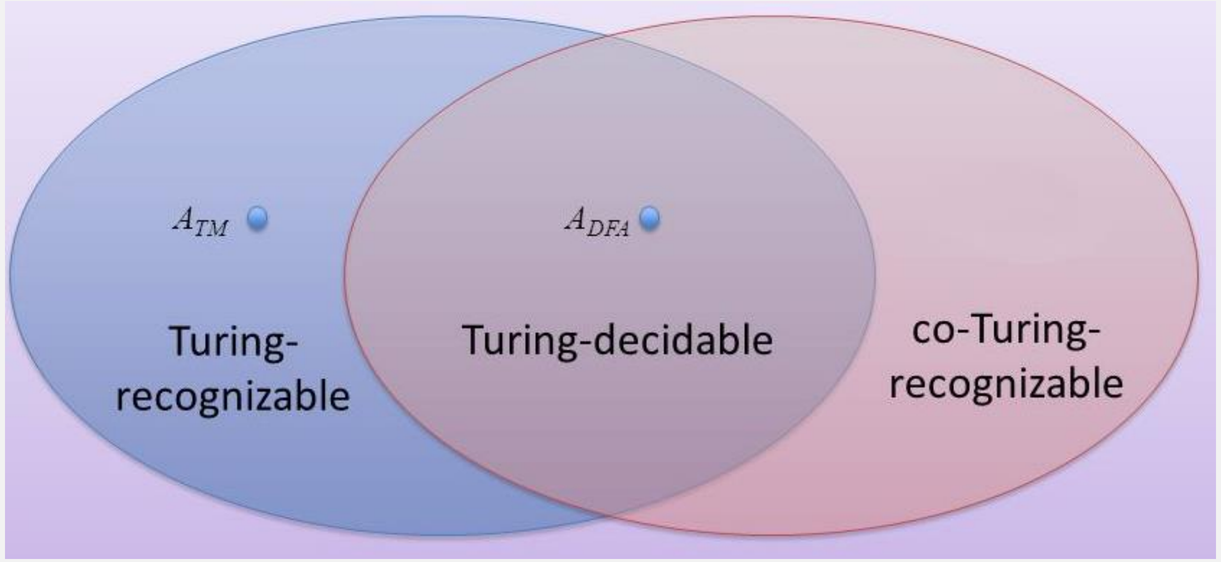
Proof: requires 2 lemmas

- Lemma 1: The **set of all languages** is *uncountable*
  - Proof: Show there is a bijection with another uncountable set …
    - … The set of all infinite binary sequences
    - ➢ Now just prove set of infinite binary sequences is uncountable (diagonalization)
- Lemma 2: The **set of all TMs** is *countable*
  - Because every TM M can be encoded as a string $<M>$
  - And set of all strings is countable
- Therefore, some language is not recognized by a TM ■

Co-Turing-Recognizability

- A language is **co-Turing-recognizable** if …
- … it is the <u>complement</u> of a Turing-recognizable language.

Thm: Decidable ⇔ Recognizable & co-Recognizable



<u>Thm</u>: Decidable ⇔ Recognizable & co-Recognizable

⇒ <u>If</u> a language is **decidable,** <u>then</u> it is **recognizable** and **co-recognizable**

- Decidable => Recognizable:
  - A decider is a recognizer, bc decidable langs are a subset of recognizable langs
- Decidable => Co-Recognizable:
  - To create co-decider from a decider … switch reject/accept of all inputs
  - A co-decider is a co-recognizer, for same reason as above

⇐ <u>If</u> a language is **recognizable** and **co-recognizable,** <u>then</u> it is **decidable**

Thm: Decidable ⇔ Recognizable & co-Recognizable

⇒ If a language is **decidable**, then it is **recognizable** and **co-recognizable**

- Decidable => Recognizable:
  - A decider is a recognizer, bc decidable langs are a subset of recognizable langs
- Decidable => Co-Recognizable:
  - To create co-decider from a decider … switch reject/accept of all inputs
  - A co-decider is a co-recognizer, for same reason as above

⇐ If a language is **recognizable** and **co-recognizable**, then it is **decidable**

- Let M_1 = recognizer for the language,
- and M_2 = recognizer for its complement
- Decider M:
  - Run 1 step on M_1,
  - Run 1 step on M_2,
  - Repeat, until one machine accepts. If it's M_1, accept. If it's M_2, reject

Termination Arg: Either M_1 or M_2 must accept and halt, so M halts and is a decider

A Turing-unrecognizable language

- We’ve proved:

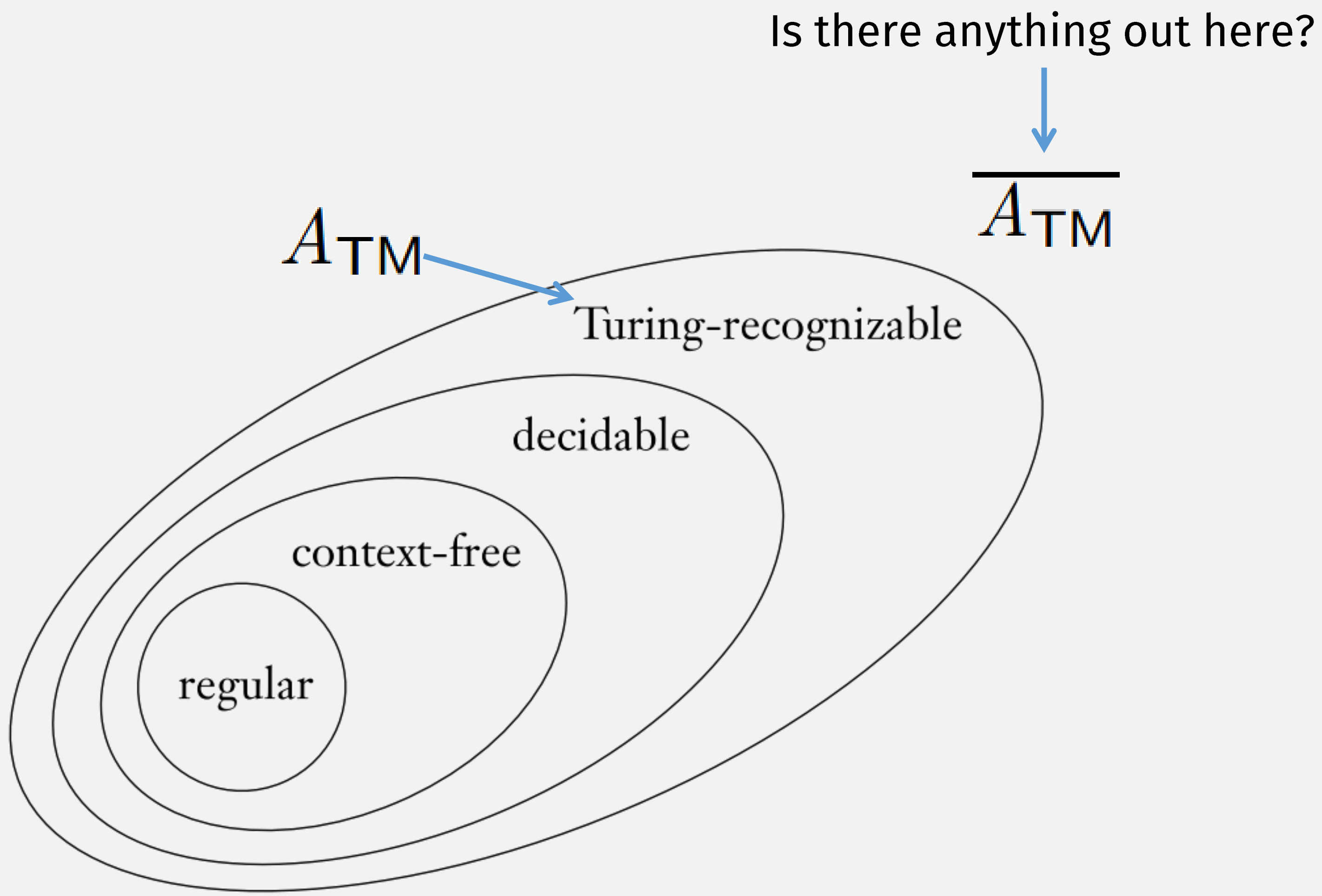
  A_{TM} is Turing-recognizable

  A_{TM} is undecidable

- So:

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

- Because: recognizable & co-recognizable implies decidable


Is there anything out here?
$\overline{A_{\mathsf{TM}}}$
A_{TM}
Turing-recognizable
decidable
context-free
regular