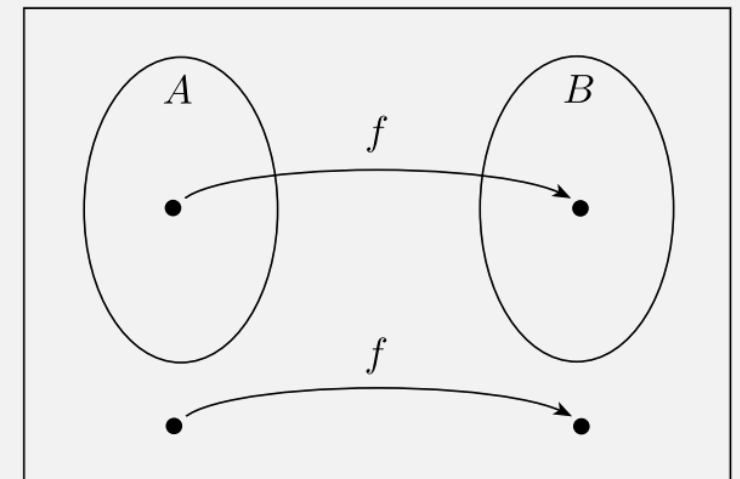


**UMB CS 420**

# Mapping Reducibility

Wednesday, April 29, 2024



# *Announcements*

- HW 10 out
  - Due Wed 5/1 12pm noon

Also:

- 5/1: HW 11 out
- 5/8: HW 11 in, HW 12 out
- 5/8: last lecture
- 5/15: HW 12 in (no exceptions)

Lecture participation question 4/29 (in gradescope)

- Mapping reducibility is a relation between two ...?

# Flashback: “Reduced”

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

known



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

unknown

Thm:  $HALT_{TM}$  is undecidable

Proof, by contradiction:

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

Essentially, we convert decidability of an  $A_{TM}$  string ...

$S =$  “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*.”

(Use  $R$  to) First: check if  $M$  will loop on  $w$

Then: run  $M$  on  $w$ , knowing it won't loop!

... into decidability of a  $HALT_{TM}$  string

A potential *problem*: could the

- Contradiction conversion itself go into an infinite loop? **no decider!**

Let's formalize this conversion, i.e., **mapping reducibility**

## Flashback: $A_{\text{NFA}}$ is a decidable language

$$A_{\text{NFA}} = \{ \langle B, w \rangle \mid B \text{ is an NFA that accepts input string } w \}$$

Decider for  $A_{\text{NFA}}$  :

$N =$  “On input  $\langle B, w \rangle$ , where  $B$  is an NFA and  $w$  is a string:

1. Convert NFA  $B$  to an equivalent DFA  $C$ , using the procedure **NFA→DFA**
2. Run TM  $M$  on input  $\langle C, w \rangle$ .
3. If  $M$  accepts, *accept*; otherwise, *reject*.”

We said this NFA→DFA algorithm is a decider TM, but it doesn't **accept/reject**?

More generally, our analogy has been:  
“**programs ~ TMs**”,  
but programs do more than **accept/reject**?

# *Definition:* Computable Functions

A function  $f: \Sigma^* \rightarrow \Sigma^*$  is a *computable function* if some Turing machine  $M$ , on every input  $w$ , halts with just  $f(w)$  on its tape.

- A **computable function** is represented with a TM that, instead of accept/reject, “outputs” its final tape contents
- Example 1: **All arithmetic operations**
- Example 2: **Converting between machines, like DFA→NFA**
  - E.g., adding states, changing transitions, wrapping TM in TM, etc.

# Definition: Mapping Reducibility

notation

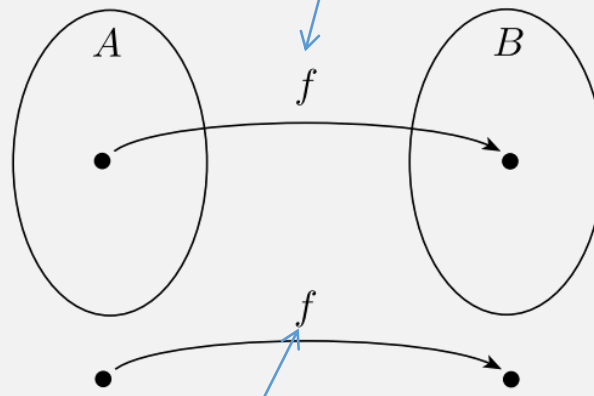
Language  $A$  is *mapping reducible* to language  $B$ , written  $A \leq_m B$ , if there is a **computable function  $f: \Sigma^* \rightarrow \Sigma^*$** , where for every  $w$ ,

$$w \in A \iff f(w) \in B.$$

$w \in A$   
“if and only if”  
 $f(w) \in B$

The function  $f$  is called the *reduction* from  $A$  to  $B$ .

“forward” direction ( $\Rightarrow$ ): if  $w \in A$  then  $f(w) \in B$



“reverse” direction ( $\Leftarrow$ ): if  $f(w) \in B$  then  $w \in A$

A function  $f: \Sigma^* \rightarrow \Sigma^*$  is a *computable function* if some Turing machine  $M$ , on every input  $w$ , halts with just  $f(w)$  on its tape.

# *Flashback:* Equivalence of Contrapositive

“If  $X$  then  $Y$ ” is equivalent to ... ?

1. “If  $Y$  then  $X$ ” (converse)
2. “If **not**  $X$  then **not**  $Y$ ” (inverse)
3. “If **not**  $Y$  then **not**  $X$ ” (contrapositive)

# *Flashback:* Equivalence of Contrapositive

**“If  $X$  then  $Y$ ”** is equivalent to ... ?

- × “If  $Y$  then  $X$ ” (converse)
  - **No!**
  
- × “If not  $X$  then not  $Y$ ” (inverse)
  - **No!**
  
- ✓ **“If not  $Y$  then not  $X$ ”** (contrapositive)
  - **Yes!**



# Definition: Mapping Reducibility

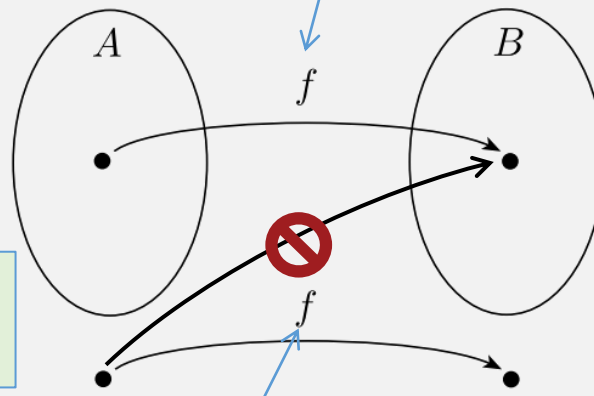
Language  $A$  is *mapping reducible* to language  $B$ , written  $A \leq_m B$ , if there is a computable function  $f: \Sigma^* \rightarrow \Sigma^*$ , where for every  $w$ ,

$$w \in A \iff f(w) \in B.$$

“if and only if”

The function  $f$  is called the *reduction* from  $A$  to  $B$ .

“forward” direction ( $\Rightarrow$ ): if  $w \in A$  then  $f(w) \in B$



Reverse direction just as important:  
“don’t convert non-As into Bs”

“reverse” direction ( $\Leftarrow$ ): if  $f(w) \in B$  then  $w \in A$

Equivalent (contrapositive): if  $w \notin A$  then  $f(w) \notin B$

Easier to prove

# Proving Mapping Reducibility: 2 Steps

Step 1:  
Show there is computable fn  $f$  ... by creating a TM

Language  $A$  is *mapping reducible* to language  $B$ , written  $A \leq_m B$ , if there is a **computable function  $f: \Sigma^* \rightarrow \Sigma^*$** , where for every  $w$ ,

$$w \in A \iff f(w) \in B. \quad \leftarrow \text{“if and only if”}$$

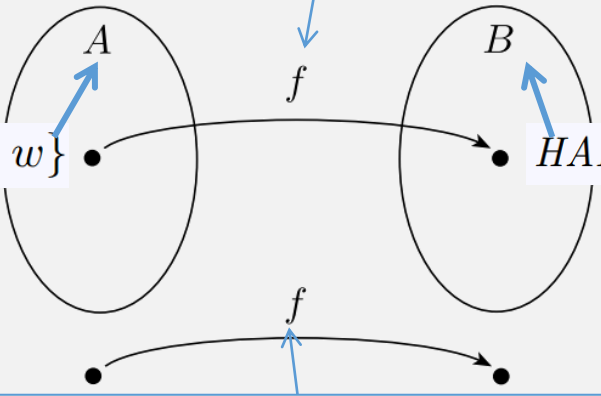
Step 2:  
Prove the iff is true for that computable fn TM

The function  $f$  is called the *reduction* from  $A$  to  $B$ .

Step 2a: “forward” direction ( $\Rightarrow$ ): if  $w \in A$  then  $f(w) \in B$

e.g.

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



Step 2b: “reverse” direction ( $\Leftarrow$ ): if  $f(w) \in B$  then  $w \in A$

Step 2b, alternate (contrapositive): if  $w \notin A$  then  $f(w) \notin B$

A function  $f: \Sigma^* \rightarrow \Sigma^*$  is a *computable function* if some Turing machine  $M$ , on every input  $w$ , halts with just  $f(w)$  on its tape.

# Thm: $A_{TM}$ is mapping reducible to $HALT_{TM}$

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

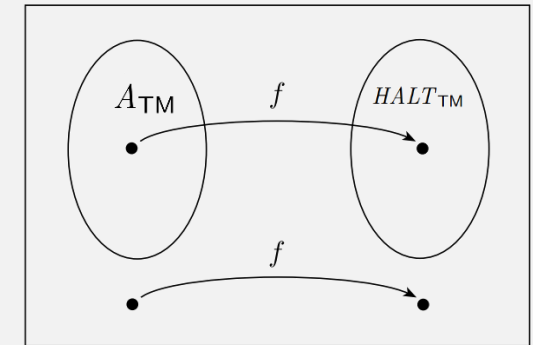


$$HALT_{TM} = \{ \langle M, w \rangle \mid M \text{ is a TM and } M \text{ halts on input } w \}$$

To show:  $A_{TM} \leq_m HALT_{TM}$

Step 1: create computable fn  $f: \langle M, w \rangle \rightarrow \langle M', w \rangle$  where:

Step 2: show  $\langle M, w \rangle \in A_{TM}$  if and only if  $\langle M', w \rangle \in HALT_{TM}$



The following machine  $F$  computes a reduction  $f$ .

$F =$  “On input  $\langle M, w \rangle$ :

1. Construct the following machine  $M'$

$M' =$  “On input  $x$ :

1. Run  $M$  on  $x$ .
2. If  $M$  accepts, *accept*.
3. If  $M$  rejects, enter a loop.”

2. Output  $\langle M', w \rangle$ .”

Converts  $M$  to  $M'$

Step 2:  
 $M$  accepts  $w$   
if and only if  
 $M'$  halts on  $w$

Output new  $M'$

$M'$  is like  $M$ , except it  
always loops when it  
doesn't accept

Language  $A$  is mapping reducible to language  $B$ , written  $A \leq_m B$ , if there is a computable function  $f: \Sigma^* \rightarrow \Sigma^*$ , where for every  $w$ ,

$$w \in A \iff f(w) \in B.$$

The function  $f$  is called the *reduction* from  $A$  to  $B$ .

A function  $f: \Sigma^* \rightarrow \Sigma^*$  is a computable function if some Turing machine  $M$ , on every input  $w$ , halts with just  $f(w)$  on its tape.

- ✓  $\Rightarrow$  If  $M$  accepts  $w$ , then  $M'$  halts on  $w$ 
  - $M'$  accepts (and thus halts) if  $M$  accepts
- $\Leftarrow$  If  $M'$  halts on  $w$ , then  $M$  accepts  $w$

The following machine  $F$  computes a reduction  $f$ .

$F =$  "On input  $\langle M, w \rangle$ :

1. Construct the following machine  $M'$ .

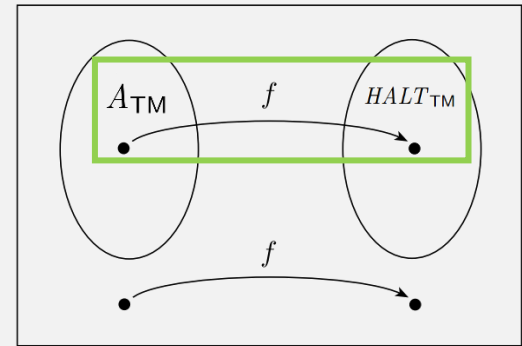
$M' =$  "On input  $x$ :

1. Run  $M$  on  $x$ .
2. If  $M$  accepts, *accept*.
3. If  $M$  rejects, enter a loop.

2. Output  $\langle M', w \rangle$ ."

If  $M$  accepts this string

Then  $M'$  accepts it (and halts)



Step 2:  
 $M$  accepts  $w$   
if and only if  
 $M'$  halts on  $w$

$M$ on (some) $w$	$M'$ on $w$
Accept	Accept
Reject	Loop!
Loop	Loop

Make an Examples Table!

- ✓  $\Rightarrow$  If  $M$  accepts  $w$ , then  $M'$  halts on  $w$ 
  - $M'$  accepts (and thus halts) if  $M$  accepts

$\Leftarrow$  If  $M'$  halts on  $w$ , then  $M$  accepts  $w$

- ✓  $\Leftarrow$  (Alternatively) If  $M$  doesn't accept  $w$ , then  $M'$  doesn't halt on  $w$  (contrapositive)

- Two possibilities for “doesn't accept”:

1.  $M$  loops:  $M'$  loops and doesn't halt

2.  $M$  rejects:  $M'$  loops and doesn't halt

The following machine  $F$  computes a reduction  $f$ .

$F =$  “On input  $\langle M, w \rangle$ :

1. Construct the following machine  $M'$ .

$M' =$  “On input  $x$ :

1. Run  $M$  on  $x$ .

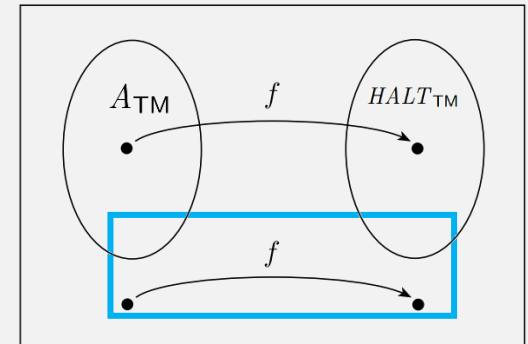
2. If  $M$  accepts, *accept*.

3. If  $M$  rejects, enter a loop.”

2. Output  $\langle M', w \rangle$ .”

If  $M$  loops, then  $M'$  loops

If  $M$  rejects, then  $M'$  loops!



Step 2:

$M$  accepts  $w$  if and only if  $M'$  halts on  $w$

$M$ on (some) $w$	$M'$ on $w$
Accept	Accept
Reject	Loop!
Loop	Loop

Now we know what **mapping reducibility** is, and how to prove it for two languages; but what is it used for?

Make an Examples Table!

# Uses of Mapping Reducibility

- To prove **Decidability**
- To prove **Undecidability**

Thm: If  $A \leq_m B$  and  $B$  is decidable, then  $A$  is decidable.

Has a decider

Must create decider

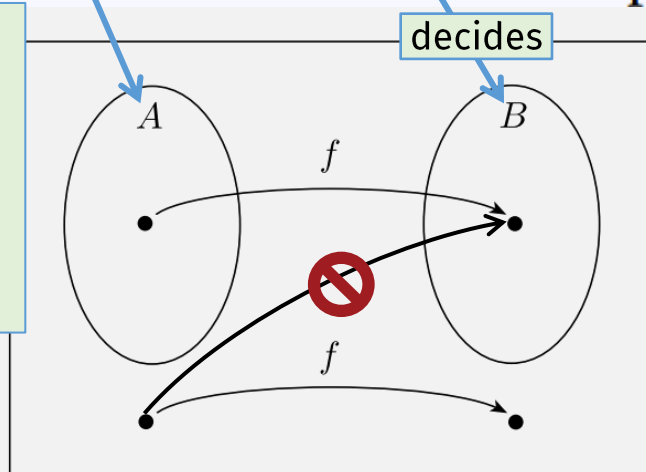
**PROOF** We let  $M$  be the decider for  $B$  and  $f$  be the reduction from  $A$  to  $B$ . We describe a decider  $N$  for  $A$  as follows.

$N =$  “On input  $w$ :

1. Compute  $f(w)$ .
2. Run  $M$  on input  $f(w)$  and output whatever  $M$  outputs.”

decides

decides



We know this is true bc of the iff (specifically the reverse direction)

Why is it true that:

If  $M$  accepts  $f(w)$  then  $N$  should accept  $w$  ??  
i.e.,  $f(w)$  in  $B$  guarantees that  $w$  in  $A$ ???

Language  $A$  is **mapping reducible** to language  $B$ , written  $A \leq_m B$ , if there is a computable function  $f: \Sigma^* \rightarrow \Sigma^*$ , where for every  $w$ ,

$$w \in A \iff f(w) \in B.$$

The function  $f$  is called the **reduction** from  $A$  to  $B$ .

Corollary: If  $A \leq_m B$  and  $A$  is undecidable, then  $B$  is undecidable.

- Proof by contradiction.
- Assume  $B$  is decidable.
- Then  $A$  is decidable (by the previous thm).
- Contradiction: we already said  $A$  is undecidable

If  $A \leq_m B$  and  $B$  is decidable, then  $A$  is decidable.



# Summary: Showing Mapping Reducibility

Step 1:  
Show there is computable  
fn  $f$  ... by creating a TM

Language  $A$  is *mapping reducible* to language  $B$ , written  $A \leq_m B$ ,  
if there is a **computable function  $f: \Sigma^* \rightarrow \Sigma^*$** , where for every  $w$ ,

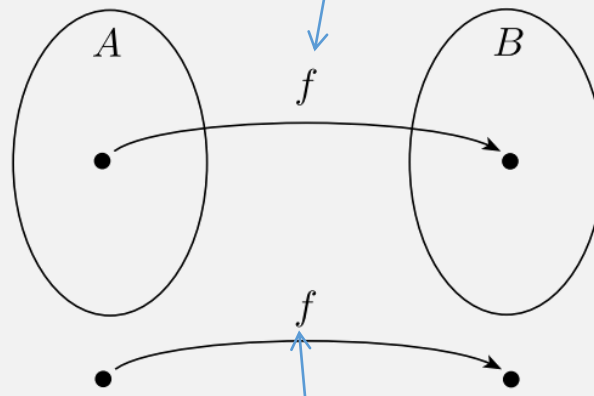
$$w \in A \iff f(w) \in B.$$

← “if and only if”

Step 2:  
Prove the iff is true

The function  $f$  is called the *reduction* from  $A$  to  $B$ .

Step 2a: “forward” direction ( $\Rightarrow$ ): if  $w \in A$  then  $f(w) \in B$



Step 2b: “reverse” direction ( $\Leftarrow$ ): if  $f(w) \in B$  then  $w \in A$

Step 2b, alternate (contrapositive): if  $w \notin A$  then  $f(w) \notin B$

A function  $f: \Sigma^* \rightarrow \Sigma^*$  is a *computable function* if some Turing machine  $M$ , on every input  $w$ , halts with just  $f(w)$  on its tape.

# Summary: Using Mapping Reducibility

To prove decidability ...

- If  $A \leq_m B$  and  $B$  is decidable, then  $A$  is decidable.

Known

Unknown  
(want to prove)

To prove undecidability ...

- If  $A \leq_m B$  and  $A$  is undecidable, then  $B$  is undecidable.

Undecidability Proof  
Technique #4:  
**Mapping Reducibility**  
+ this theorem

Be careful with the direction of the **reduction**,  
i.e., what is known and what is unknown!

# Alternate Proof: The Halting Problem

$HALT_{TM}$  is undecidable

- If  $A \leq_m B$  and  $A$  is undecidable, then  $B$  is undecidable.

Must be known

- $A_{TM} \leq_m HALT_{TM}$

**Undecidability** Proof  
Technique #4:  
**Mapping Reducibility**  
+ this theorem

- Since  $A_{TM}$  is undecidable,
- ... and we showed mapping reducibility from  $A_{TM}$  to  $HALT_{TM}$ ,
- then  $HALT_{TM}$  is undecidable ■

*Flashback:*  $EQ_{TM}$  is undecidable

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

Proof by contradiction:

- Assume  $EQ_{TM}$  has decider  $R$ ; use it to create  $E_{TM}$  decider:  
 $= \{\langle M \rangle \mid M \text{ is a TM and } L(M) = \emptyset\}$

$S =$  “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*.”

## Alternate Proof: $EQ_{TM}$ is undecidable

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

Show mapping reducibility:  $E_{TM} \leq_m EQ_{TM}$

Step 1: create computable fn  $f: \langle M \rangle \rightarrow \langle M_1, M_2 \rangle$ , computed by  $S$

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

1. Construct:  $\langle M, M_1 \rangle$ , where  $M_1$  is a TM that rejects all inputs.
2. Output:  $\langle M, M_1 \rangle$

Step 2: show iff requirements of mapping reducibility (hw exercise?)

And use theorem ...

**Undecidability Proof Technique #4:**  
**Mapping Reducibility + theorem** 

If  $A \leq_m B$  and  $A$  is undecidable, then  $B$  is undecidable.

Flashback:  $E_{\text{TM}}$  is undecidable

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

Proof, by contradiction:

- Assume  $E_{\text{TM}}$  has decider  $R$ ; use it to create  $A_{\text{TM}}$  decider:

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

1. Use the description of  $M$  and  $w$  to construct the TM  $M_1$

2. Run  $R$  on input  $\langle M_1 \rangle$ .

3. If  $R$  accepts, *reject*; if  $R$  rejects, *accept*.”

$M_1 =$  “On input  $x$ :

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

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

If  $M$  accepts  $w$ ,  
then  $M_1$  accepts  $w$ ,  
meaning  $M_1$  is not in  $E_{\text{TM}}$ !

- So this only reduces  $A_{\text{TM}}$  to  $\overline{E_{\text{TM}}}$

# Alternate Proof: $E_{TM}$ is undecidable

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

Show mapping reducibility??:  $A_{TM} \leq_m E_{TM}$

Step 1: create computable fn  $f: \langle M, w \rangle \rightarrow \langle M' \rangle$ , computed by  $S$

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

1. Use the description of  $M$  and  $w$  to construct the TM  $M_1$

2. Output:  $\langle M_1 \rangle$ .

3. ~~If  $R$  accepts, reject; if  $R$  rejects, accept.~~”

$M_1 =$  “On input  $x$ :

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

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

If  $M$  accepts  $w$ ,  
then  $M_1$  accepts  $w$ ,  
meaning  $M_1$  is not in  $E_{TM}$ !

- So this only reduces  $A_{TM}$  to  $\overline{E_{TM}}$
- It's good enough! Still proves  $E_{TM}$  is undecidable
  - If ... undecidable langs are closed under **complement**

Step 2: show iff requirements of mapping reducibility (hw exercise?)

# Language Complement

**Complement** (NEG from hw3) of a language  $A$ , written  $\overline{A}$  ...

... is the set of all strings not in set  $A$

*Example:*

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

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

$$\cup \{ w \mid w \text{ is a string that is not a TM description} \}$$

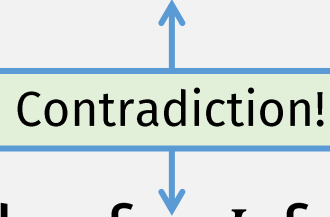


# Undecidable Langs Closed under Complement

Proof by contradiction

- Assume some lang  $L$  is undecidable and  $\bar{L}$  is decidable ...
  - Then  $\bar{L}$  has a decider

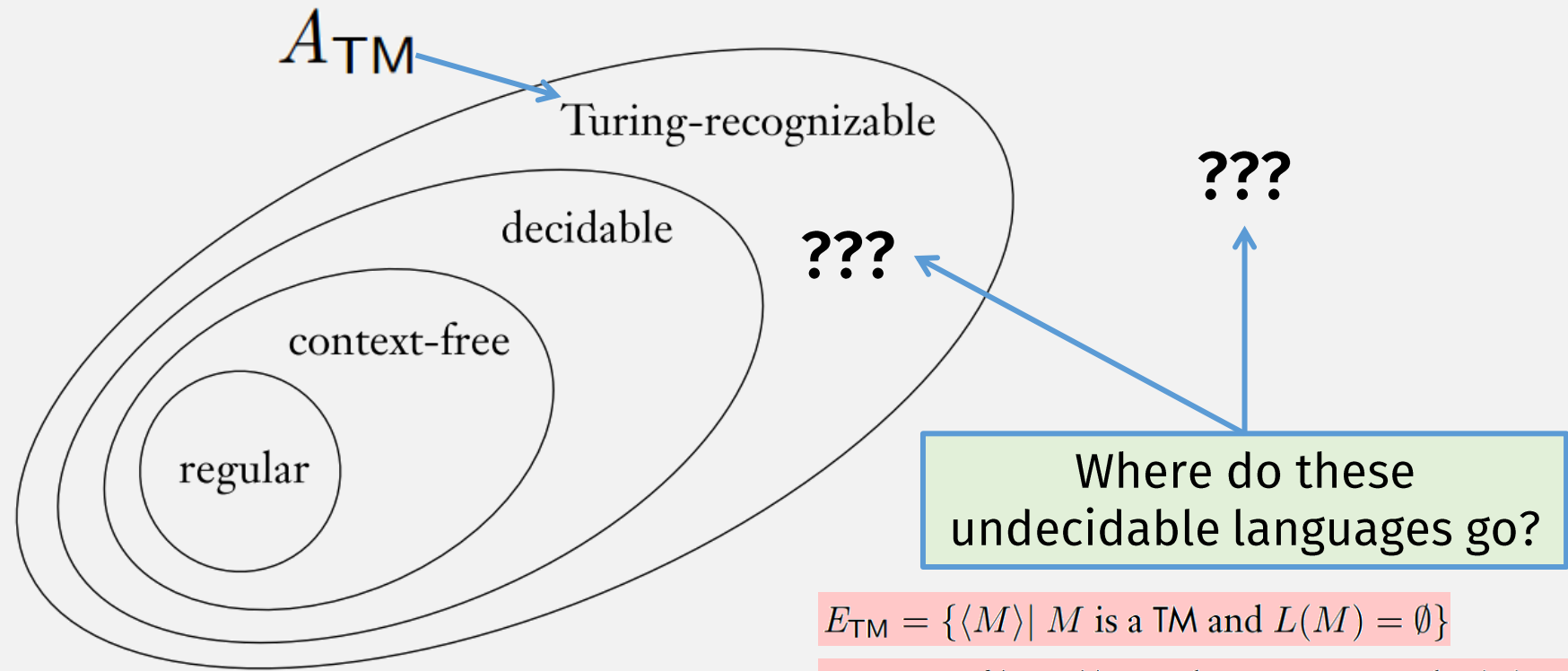
Contradiction!



- ... then we can create decider for  $L$  from decider for  $\bar{L}$  ...
  - Because decidable languages are closed under complement (hw?!)

# Next Time: Turing Unrecognizable?

Is there anything out here?



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

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

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

# **Class Participation Question 4/29**

On gradescope