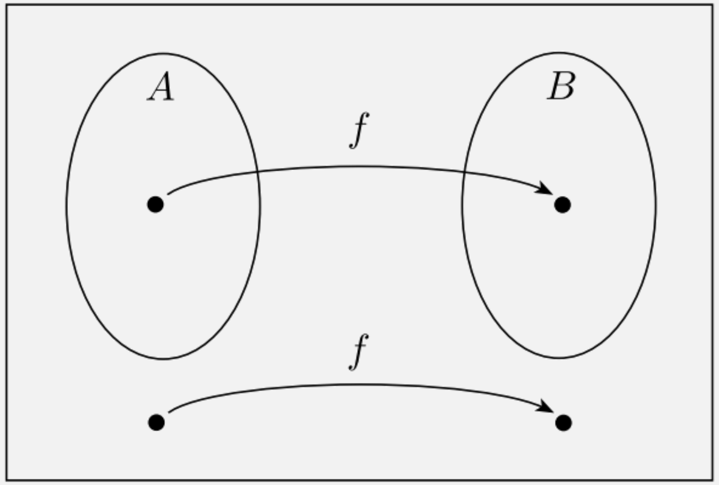
UMB CS 420

Mapping Reducibility

Thursday, November 17, 2022



Announcements

- Current hw: HW 9
  - Due Mon 11/21 11:59pm EST

- Next hw: HW 10
  - Out: Tue 11/22
  - Due: Mon 12/5
  - 2 weeks due to Thanksgiving break

Last time: Undecidable …

Seems like no algorithm can compute **anything** about Turing Machines, i.e., about programs …

- $REGULAR_{\mathrm{TM}}$ = {<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}
- …

Rice's Theorem

- $ANYTHING_{\mathrm{TM}}$ = {<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: $ANYTHING_{\text{TM}}$ is Undecidable

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

Proof by contradiction

- Assume some language satisfying $ANYTHING_{\text{TM}}$ has a decider R.
  - Since $ANYTHING_{\text{TM}}$ is non-trivial, then there exists $M_{ANY} \in ANYTHING_{\text{TM}}$
  - 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: $M_w = M_{\text{ANY}}$
If M doesn't accept w: M_w accepts nothing

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

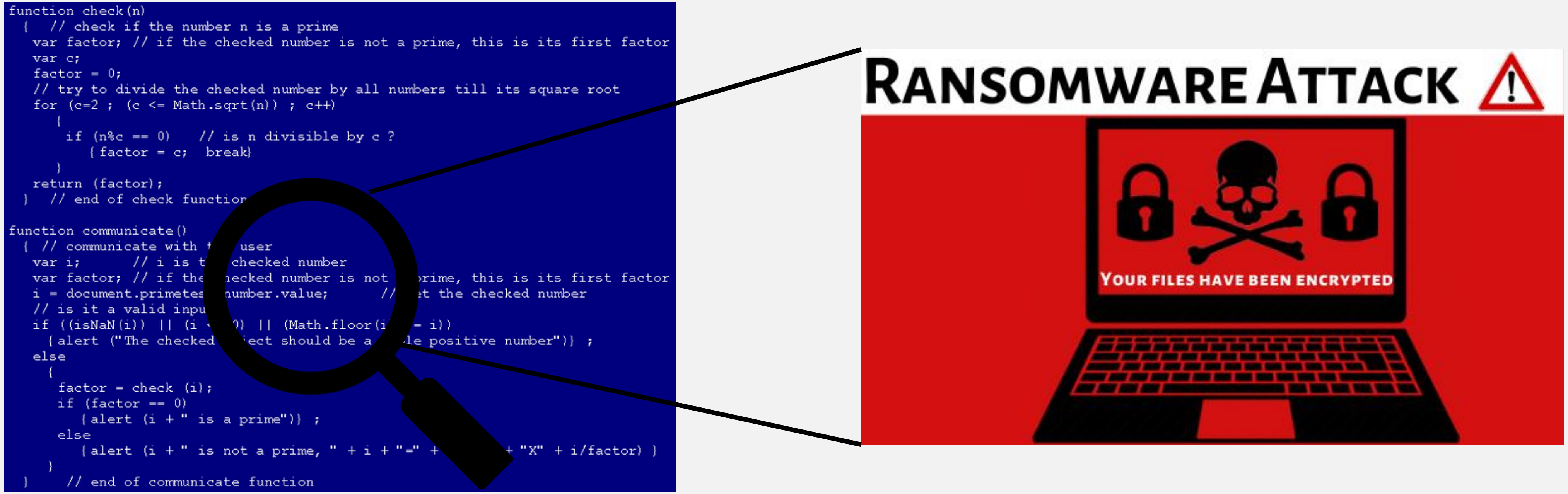
- Run R on M_w
  - If it accepts, then $M_w = M_{ANY}$, so M accepts w, so accept
  - Else reject

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

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

Prove that the following is undecidable:

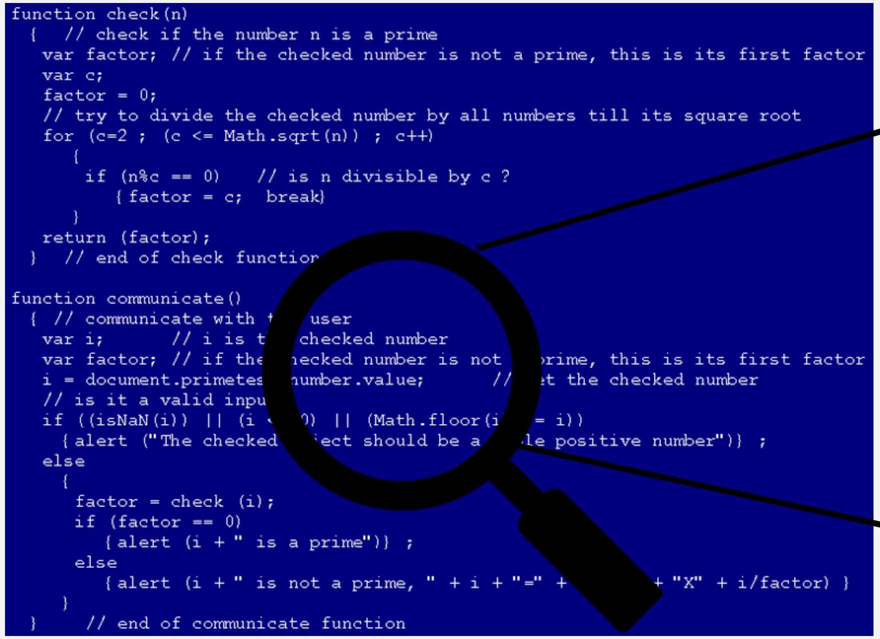
$\{<M> \mid M \text{ is a TM that installs malware}\}$



Rice's Theorem Implication

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

Undecidable!
(by Rice's Theorem)



Flashback: "Reduced"

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

$\Downarrow$

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

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

Proof, by contradiction:

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

... into 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

Essentially, **we** <u>convert</u> a hypothetical $HALT_{\mathsf{TM}}$ string ...

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

Let's <u>formalize</u> this conversion, i.e., **mapping reducibilty**

Flashback: A_{NFA} is a decidable language

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

Decider for A_{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 $f: \Sigma^* \longrightarrow \Sigma^*$ 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

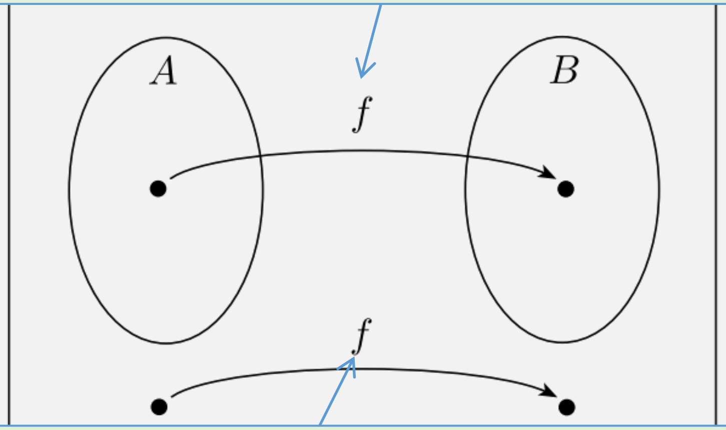
Language A is ***mapping reducible*** to language B, written $\boxed{A \leq_{\mathrm{m}} B}$, if there is a $\boxed{\text{computable function } f \colon \Sigma^* \longrightarrow \Sigma^*}$, where for every w,

$$\boxed{w \in A \iff f(w) \in B.}$$

"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 ($\Leftarrow$): if $f(w) \in B$ then $w \in A$

A function $f \colon \Sigma^* \longrightarrow \Sigma^*$ 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

$\boxed{\text{"If } X \text{ then } Y\text{"}}$ is equivalent to … ?

- "If Y then X" (converse)
  - No!
- "If not X then not Y" (inverse)
  - No!

✓ $\boxed{\text{"If not } Y \text{ then not } X\text{"}}$ (contrapositive)
  - **Yes!**

Definition: Mapping Reducibility

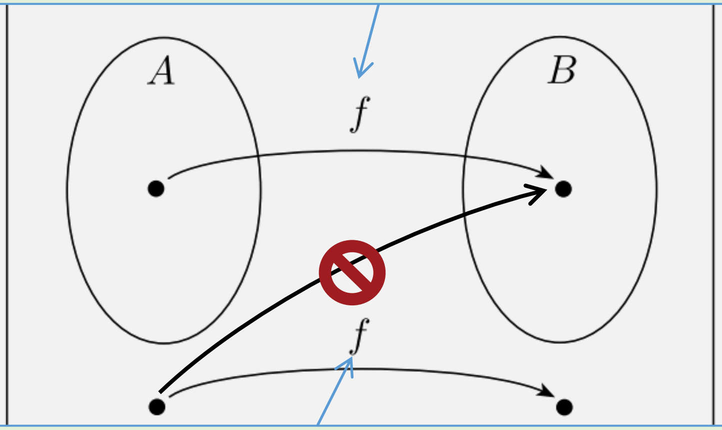
Language A is ***mapping reducible*** to language B, written $A \leq_{\mathrm{m}} B$, if there is a computable function $f\colon \Sigma^* \longrightarrow \Sigma^*$, where for every w,

$$\boxed{w \in A \iff f(w) \in B.}$$

"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 ($\Leftarrow$): if $f(w) \in B$ then $w \in A$

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

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 $A \leq_{\mathrm{m}} B$, if there is a computable function $f\colon \Sigma^* \longrightarrow \Sigma^*$, where for every w,

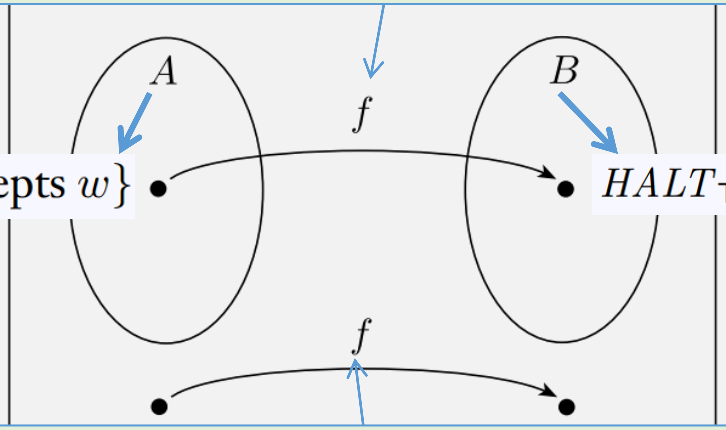
$$w \in A \iff f(w) \in B.$$

"if and only if"

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

Step 2:
Prove the iff is true

Step 2a: "forward" direction (⇒): if $w \in A$ then $f(w) \in B$



e.g.

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

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

Step 2b: "reverse" direction (⇐): if $f(w) \in B$ then $w \in A$

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

A function $f\colon \Sigma^* \longrightarrow \Sigma^*$ 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 $HALT_{\mathsf{TM}}$

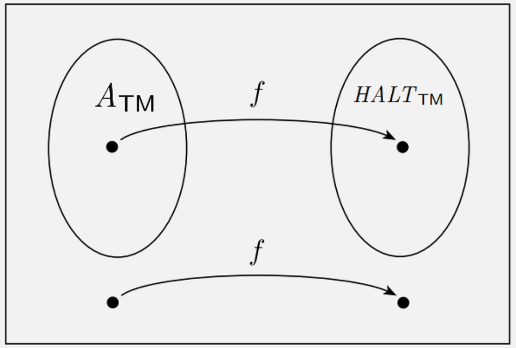
$A_{\mathsf{TM}} = \{\langle M, w\rangle \mid M \text{ is a TM and } M \text{ accepts } w\}$

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

To show: $A_{\mathsf{TM}} \leq_{\mathrm{m}} HALT_{\mathsf{TM}}$

Step 1: **create computable fn *f*: <*M*, *w*> → <*M'*, *w*>** where:

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



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 $A \leq_{\mathrm{m}} B$, if there is a computable function $f\colon \Sigma^* \longrightarrow \Sigma^*$, 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 $f\colon \Sigma^* \longrightarrow \Sigma^*$ is a ***computable function*** if some Turing machine M, on every input w, halts with just $f(w)$ on its tape.

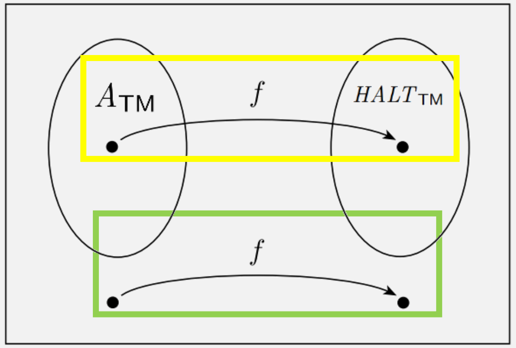
⇒ If M **accepts** w, then M' **halts on** w

- M' **accepts** (and thus **halts**) **if** M **accepts**

⇐ If M' **halts on** w, then M **accepts** w

⇐ (Alternatively) If M **doesn't accept** w, then M' **doesn't halt on** w (contrapositive)

- **Two possibilities for non-acceptance:**
  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 $\langle M', w\rangle$."

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

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

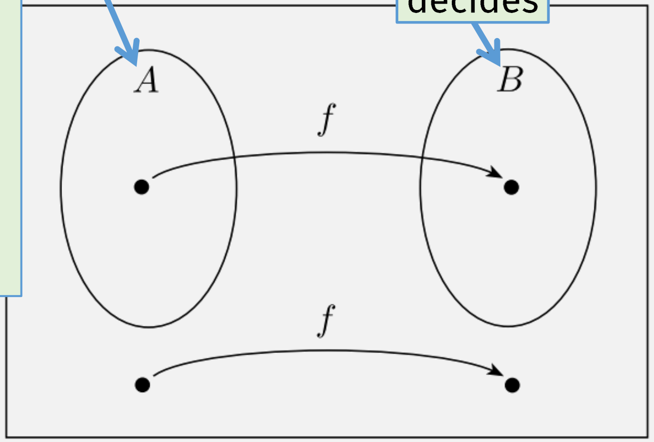
PROOF We let M be the decider for B and f be the reduction from A to B. We describe a decider N for A as follows.

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)



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

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

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

Coro: 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 $A \leq_{\mathrm{m}} B$, if there is a $\boxed{\text{computable function } f\colon \Sigma^* \longrightarrow \Sigma^*}$, where for every w,

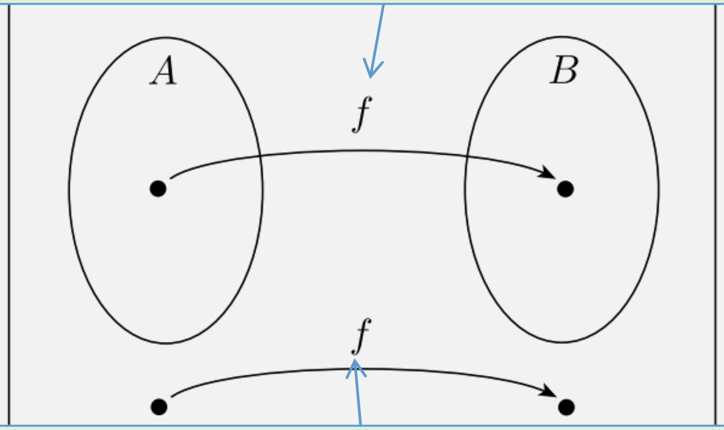
$$\boxed{w \in A \iff f(w) \in B.}$$

"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 (⇒): if $w \in A$ then $f(w) \in B$



Step 2b: "reverse" direction (⇐): if $f(w) \in B$ then $w \in A$

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

A function $f\colon \Sigma^* \longrightarrow \Sigma^*$ 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 $A \leq_{\mathrm{m}} B$ and B is decidable, then A is decidable.

Known

Unknown
(want to prove)

To prove undecidability …

- If $A \leq_{\mathrm{m}} B$ and A is undecidable, then B is undecidable.

Be careful with the **direction of the reduction!**

Alternate Proof: The Halting Problem

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

- If $A \leq_{\mathrm{m}} B$ and A is undecidable, then B is undecidable.

Must be known

- $A_{\mathsf{TM}} \leq_{\mathrm{m}} HALT_{\mathsf{TM}}$

- Since A_{TM} is undecidable,
- … and we showed mapping reducibility from A_{TM} to $HALT_{\mathsf{TM}}$,
- then $HALT_{\mathsf{TM}}$ is undecidable $\blacksquare$

Flashback: EQ_{TM} is undecidable

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

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

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

Show **mapping reducibility:** $E_{\mathsf{TM}} \leq_{\mathrm{m}} EQ_{\mathsf{TM}}$

Step 1: create computable fn f: <M> → <M_1, M_2>, 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** (exercise)

And use theorem …

If $A \leq_{\mathrm{m}} B$ and A is undecidable, then B is undecidable.

Flashback: E_{TM} is undecidable

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

Proof, by contradiction:

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

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

2. Run R on input $\langle M_1\rangle$.
3. If R accepts, *reject*; if R rejects, *accept*."

If M accepts w, M_1 not in E_{TM}!

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

Alternate Proof: E_{TM} is undecidable

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

Show **mapping reducibility??:** $A_{\mathsf{TM}} \leq_{\mathrm{m}} E_{\mathsf{TM}}$

Step 1: create computable fn *f*: <*M*, *w*> → <*M'*>, 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

$M_1 =$ "On input x:

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

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

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

3. ~~If R accepts, *reject*; if R rejects, *accept*.~~"

If M accepts w, M_1 not in E_{TM}!

- So this only reduces A_{TM} to $\overline{E_{\mathsf{TM}}}$
- 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 (exercise)

Undecidable Langs Closed under Complement

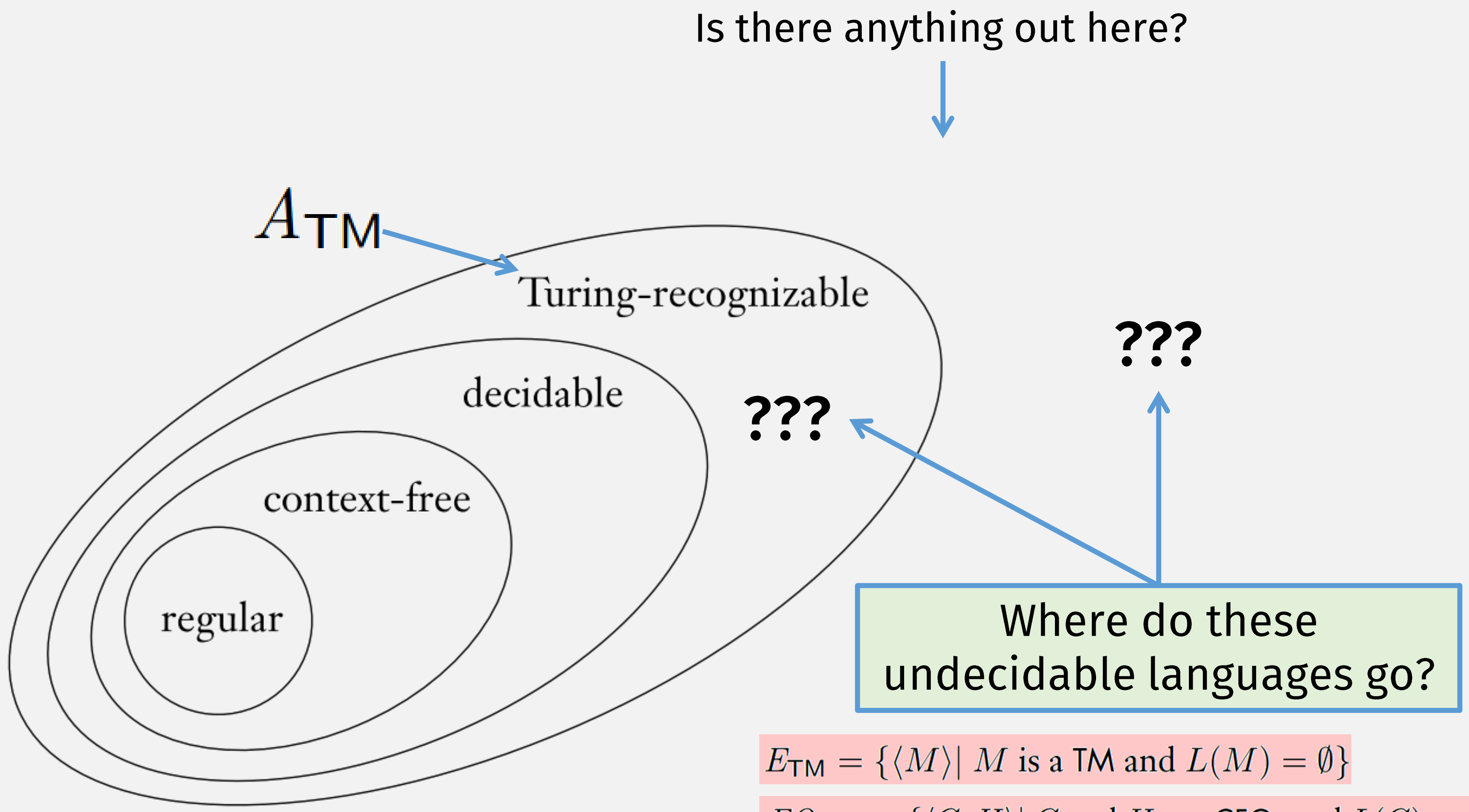
Proof by contradiction

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

Contradiction!

- … then we can create decider for L from decider for $\overline{L}$ …
  - Because decidable languages are closed under complement (hw10?)!

Next Time: Turing Unrecognizable?



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

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

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

Check-in Quiz 11/17

On gradescope