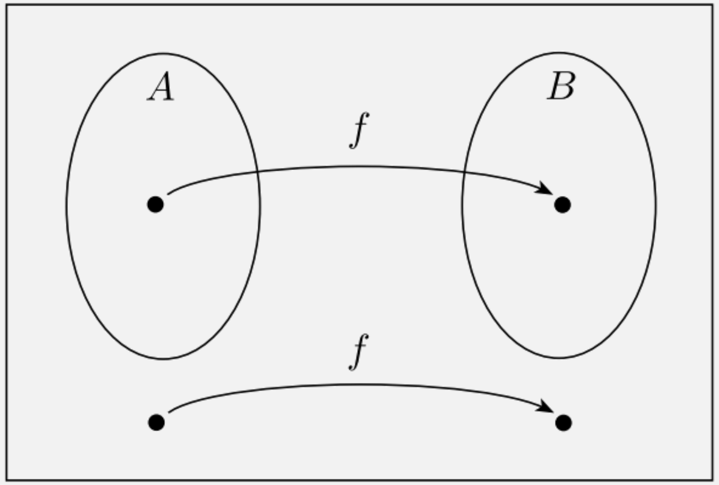
CS 420 / CS 620

Mapping Reducibility

Monday, November 24, 2025

UMass Boston Computer Science



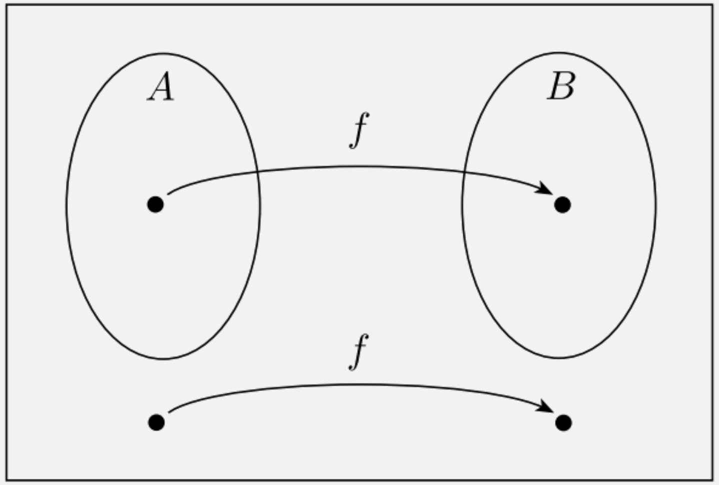
Announcements

- HW 11
  - ~~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)**

Last HW

- 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)



Flashback: "Reduced"

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

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

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

Proof, by **contradiction**:

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

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 $HALT_{\mathsf{TM}}$ **string**

A **potential** ***problem***: could the **conversion** itself go into an infinite loop?

- Contradicti... **no decider!**

Today: **formalize** this conversion, i.e., **mapping reducibilty**

Flashback: A_{NFA} is a decidable language

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

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

- 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

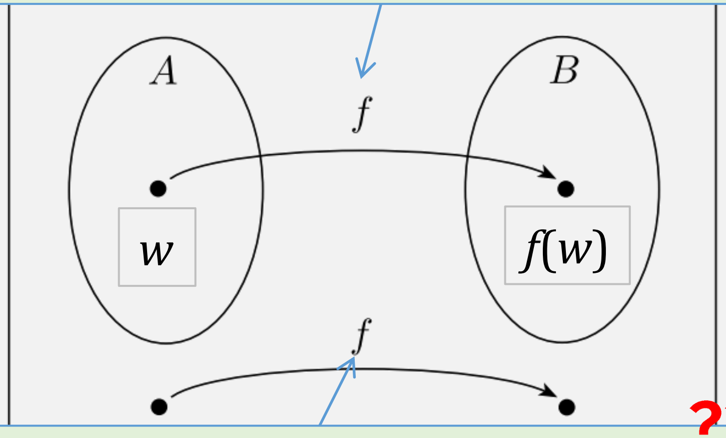
Language A is ***mapping reducible*** to language B, written $\boxed{A \leq_{\mathrm{m}} B}$, if there is a $\boxed{\text{computable function } f\colon \Sigma^* \longrightarrow \Sigma^*}$, where for every w,

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

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

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

"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

"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

$\boxed{\text{“If } X \text{ then } Y\text{”}}$ **is equivalent to ... ?**

× “If Y then X” (converse)

- **No!**

× “If **not** X then **not** Y” (inverse)

- **No!**

✓ $\boxed{\text{“If } \textbf{not } Y \text{ then } \textbf{not } X\text{”}}$ (contrapositive)

- **Yes!**

Definition: Mapping Reducibility

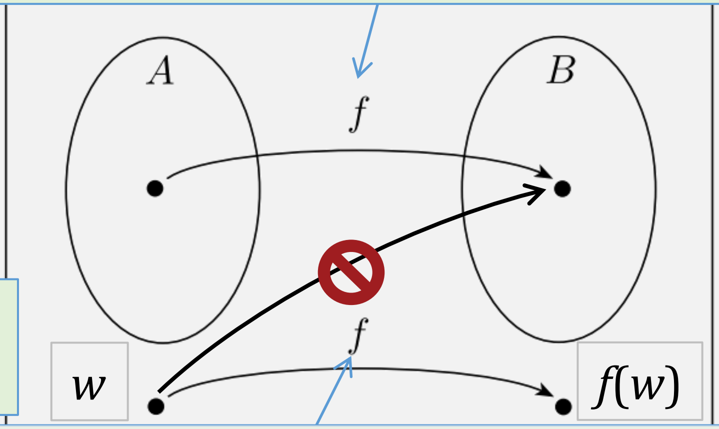
Language A is ***mapping reducible*** to language B, written $A \leq_{\mathrm{m}} B$, if there is a computable function $f \colon \Sigma^* \longrightarrow \Sigma^*$, where for every w,

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

"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 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 $f: \Sigma^* \longrightarrow \Sigma^*$, where for every w,

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

"if and only if"

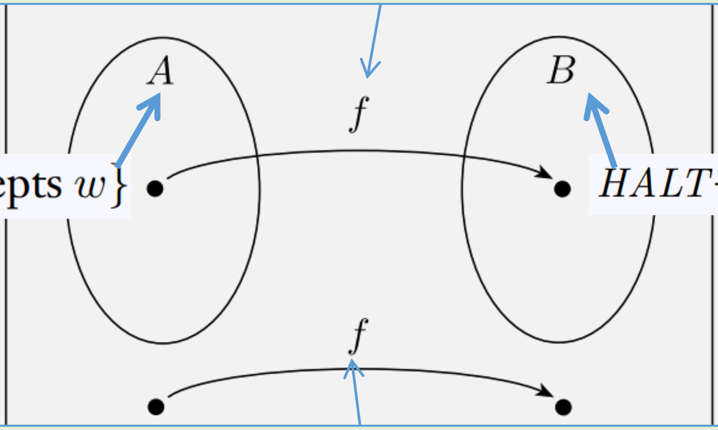
The function f is called the ***reduction*** from A to B.

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

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 $f: \Sigma^* \longrightarrow \Sigma^*$ 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 $HALT_{\mathsf{TM}}$

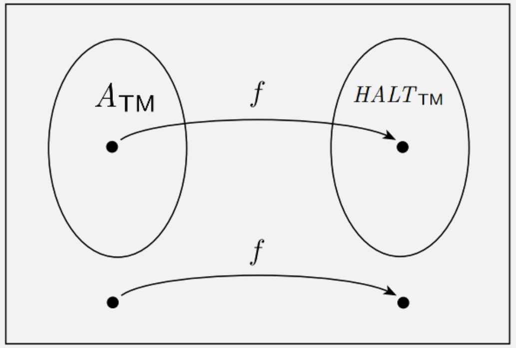
$A_{\mathsf{TM}} = \{\langle M, w\rangle \mid M \text{ is a TM and } M \text{ accepts } w\}$

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

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.

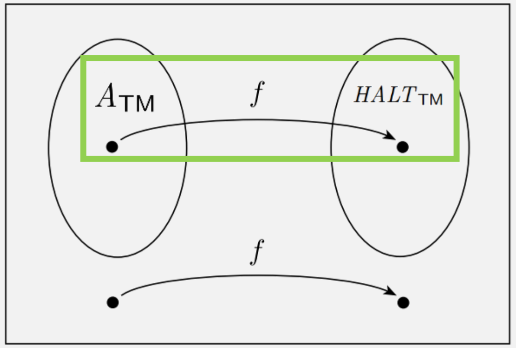
A function $f\colon \Sigma^* \longrightarrow \Sigma^*$ is a ***computable function*** if some Turing machine M, on every input w, halts with just $f(w)$ on its tape.

☑ ⇒ If M accepts w, then M' halts on w ← Expected output

assume

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

⇐ If M' halts on w, then M accepts w



The following machine F computes a reduction f.

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

1. Construct the following machine M'.
   $\boxed{M'}$ = "On input x:
   1. Run M on x. ← Assume M accepts w
   2. If M accepts, *accept*.
   3. If M rejects, enter a loop."
2. Output $\langle \boxed{M'}, w\rangle$."

then M' accepts w (and halts)

M on w	M' on w	expected M' on w
Accept	Accept	Accept/Reject (halt)

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

This step requires an Examples Table (for output-producing TMs)!

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

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

Check that: You can write this proof as **Statements / Justifications** …

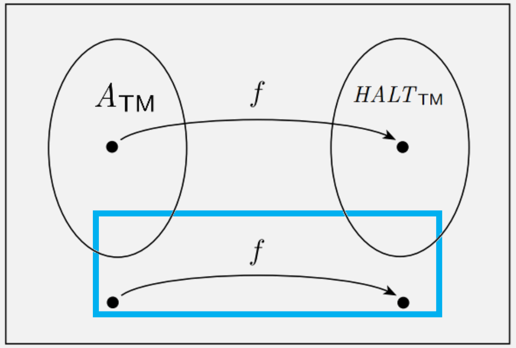
⇐ If M' halts on w, then M accepts w

assume

Expected output

☑ ⇐ (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 …

If M rejects …

M on w	M' on w	expected M' on w
Accept	Accept	Accept/Reject (halt)
Reject	Loop	Loop
Loop	Loop	Loop

… then M' loops!

… then M' loops

This step requires an Examples Table (for output-producing TMs)!

Previously

Hint: This is an **IF-THEN Statement** ...

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

Language A is ***mapping reducible*** to language B, written $A \leq_{\mathrm{m}} B$, $\boxed{\text{if}}$ 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.

Definition of **computable function**

IF a TM M computes f,
THEN f is a **computable function**

Definition of **mapping reducible**

IF there is a **computable function** f,
where $w \in A \Leftrightarrow f(w) \in B$,
THEN $A \leq_{\mathrm{m}} B$

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

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

Statements	**Justifications**
TODO	
1. TM F computes a function f	1. **Definition** of (output-producing) **TM**
2. f is a **computable function**	2. **Definition** of **computable function**
(iff forward) 3. $\langle M,w\rangle \in A_{\mathsf{TM}} \Rightarrow f(\langle M,w \rangle) \in HALT_{\mathsf{TM}}$	3. **Examples Table, row 1** TODO
(iff reverse, contrapositive) 4. $\langle M,w\rangle \notin A_{\mathsf{TM}} \Rightarrow f(\langle M,w \rangle) \notin HALT_{\mathsf{TM}}$	4. **Examples Table, row 2-3**
5. $\langle M,w\rangle \in A_{\mathsf{TM}} \Leftrightarrow f(\langle M,w \rangle) \in HALT_{\mathsf{TM}}$	5. **Stmts 4 and 5**
6. $A_{\mathsf{TM}} \leq_{\mathrm{m}} HALT_{\mathsf{TM}}$ (Statement to Prove)	6. **Definition of mapping reducible**

Definition of **computable function**

IF a TM M computes f,
THEN f is a **computable function**

Definition of **mapping reducible**

IF there is a **computable function** f,
where $w \in A \Leftrightarrow f(w) \in B$,
THEN $A \leq_{\mathrm{m}} B$

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.

f converts:
- $w \in A$ to $f(w) \in B$, and
- $w' \notin A$ to $f(w') \notin B$

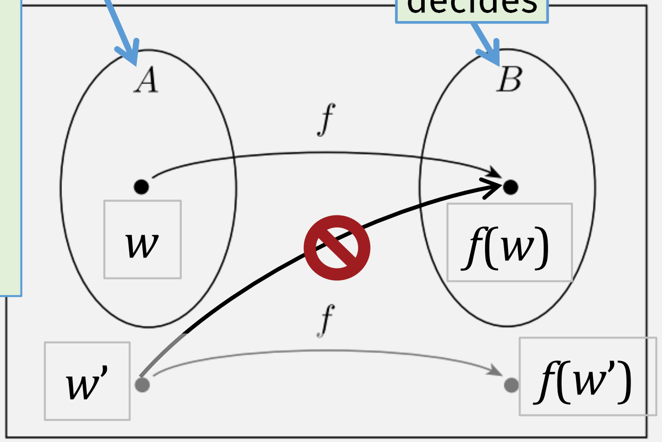
$N =$ "On input w:

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 $f\colon \Sigma^* \longrightarrow \Sigma^*$, where for every w,

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

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

Uses of Mapping Reducibility

☑ • To prove **Decidability**

??? • To prove **Undecidability**

Corollary: If $A \leq_{\mathrm{m}} B$ 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 $A \leq_{\mathrm{m}} B$ and B is decidable, then A is decidable.

Uses of Mapping Reducibility

☑ • To prove **Decidability**

☑ • To prove **Undecidability**

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,

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

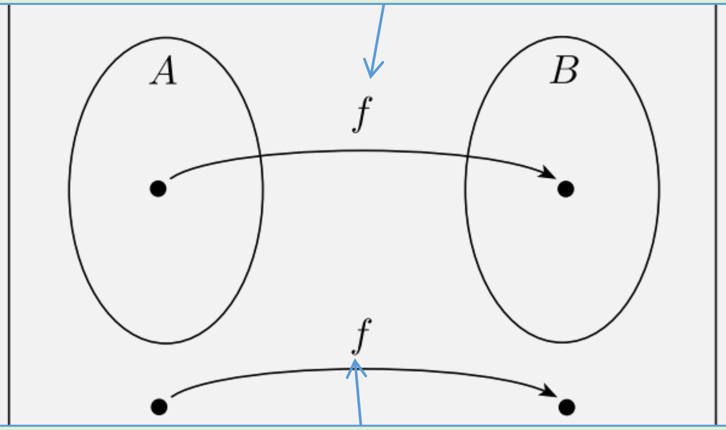
"if and only if"

Step 2:
Prove the iff is true

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

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

(using an Examples Table, for output-producing TMs)



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 $f\colon \Sigma^* \longrightarrow \Sigma^*$ 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. (Sipser 5.22)

Known

Unknown
(want to prove)

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

To prove undecidability …

- If $A \leq_{\mathrm{m}} B$ and A is undecidable, then B is undecidable. (Sipser 5.23)

Be careful with: the <u>direction</u> **of the reduction,** i.e., **what is known** and **what is unknown!**

As **S/J?**

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

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

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

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

As S/J

Alternate Proof: The Halting Problem

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

Previous proof of: $A_{\text{TM}} \leq_{\text{m}} HALT_{\text{TM}}$

Statements	Justifications
1. TM F computes a function f	1. Definition of (output-producing) TM
2. f is a computable function	2. Definition of computable function
3. $\langle M,w\rangle \in A_{\text{TM}} \Rightarrow f(\langle M,w \rangle) \in HALT_{\text{TM}}$	3. Examples Table, row 1
4. $\langle M,w\rangle \notin A_{\text{TM}} \Rightarrow f(\langle M,w \rangle) \notin HALT_{\text{TM}}$	4. Examples Table, row 2-3
5. $\langle M,w\rangle \in A_{\text{TM}} \Leftrightarrow f(\langle M,w \rangle) \in HALT_{\text{TM}}$	5. Stmts 4 and 5
6. $A_{\text{TM}} \leq_{\text{m}} HALT_{\text{TM}}$	6. Definition of mapping reducible
7. A_{TM} is undecidable	7. Sipser 4.11
8. $HALT_{\text{TM}}$ is undecidable	8. Sipser 5.23

If $A \leq_{\text{m}} B$ and A is undecidable, then B is undecidable. (Sipser 5.23)

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: $<M> \rightarrow <M_1, M_2>$, 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

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: $\langle M\rangle \rightarrow \langle M_1, M_2\rangle$, **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**

☑ $\Rightarrow$ If $\langle M\rangle \in E_{\mathsf{TM}}$, then $\langle M, M_1\rangle \in EQ_{\mathsf{TM}}$

☑ $\Leftarrow$ If $\langle M\rangle \notin E_{\mathsf{TM}}$, then $\langle M, M_1\rangle \notin EQ_{\mathsf{TM}}$

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

- So this only reduces A_{TM} to $\overline{E_{\mathsf{TM}}}$
- Maybe ok? Can still prove: E_{TM} is undecidable
  - If … undecidable langs are closed under **complement**

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

Language Complement

Complement (COP from hw9) of a language A, written $\overline{A}$ ...

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

Example:

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

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

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

Undecidable Langs Closed under Complement

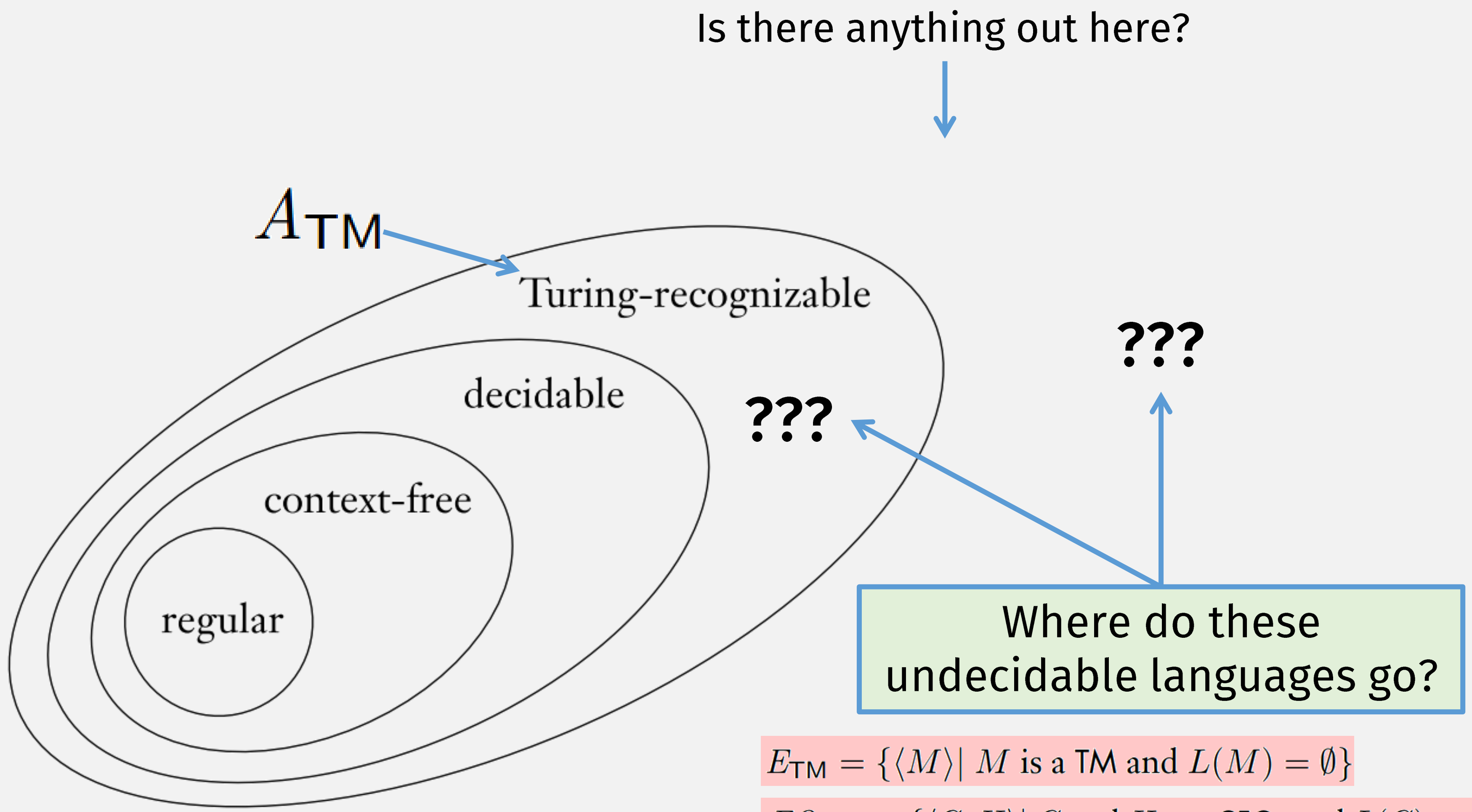
Proof by **contradiction**

- 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** (hw?)**!**

Next: Turing Unrecognizable?



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

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

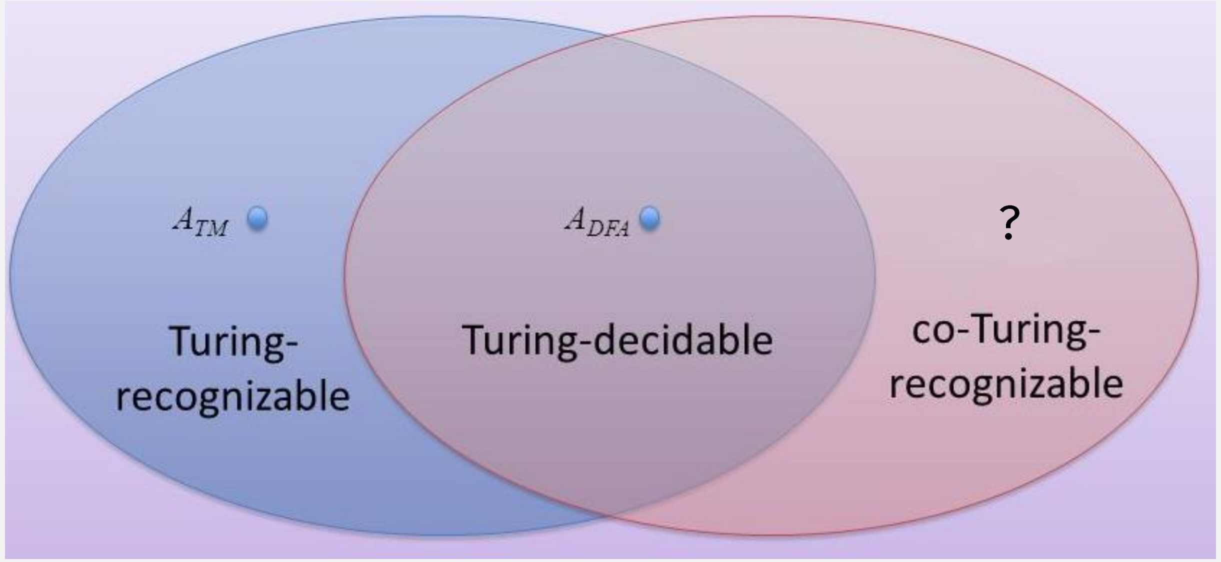
$EQ_{\mathsf{TM}} = \{\langle M_1, M_2 \rangle \mid M_1 \text{ and } M_2 \text{ are TMs and } L(M_1) = L(M_2)\}$

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

(complement)



A Turing-unrecognizable language

- We've proved:

  A_{TM} is Turing-recognizable

  A_{TM} is undecidable

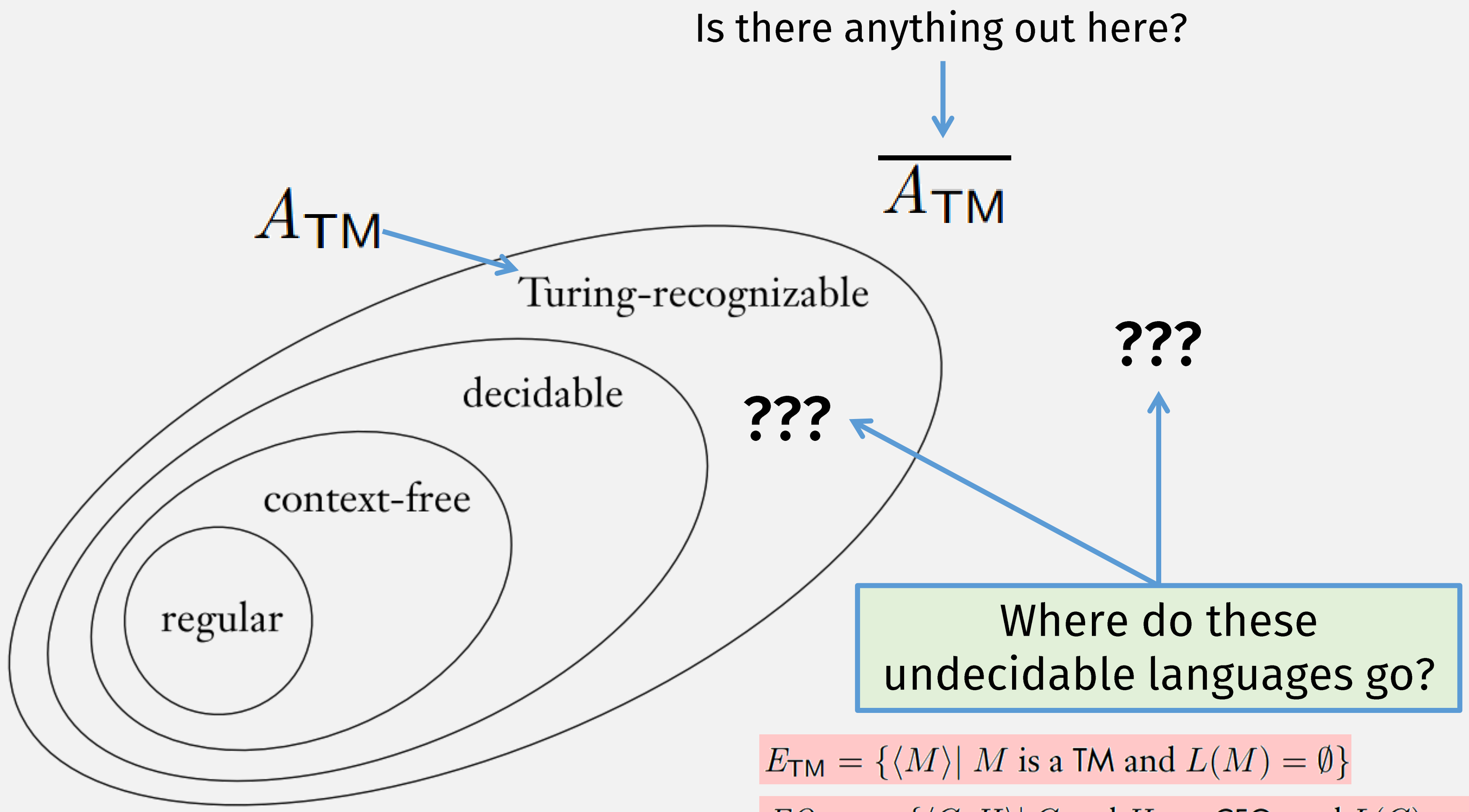
- So:

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

Unrecognizability
Proof Technique #1

- We know: **recognizable & co-recognizable ⇒ decidable**

  Contrapositive: undecidable ⇒ can't be both recognizable & co-recognizable



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

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

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

Thm: EQ_{CFG} is not Turing-recognizable

Unrecognizability
Proof Technique #1

$\boxed{\text{Recognizable \& co-recognizable} \Rightarrow \text{decidable}}$

Contrapositive: undecidable $\Rightarrow$ can't be both recognizable & co-recognizable

- We didn't prove this yet (but it is true and we will assume it here):
  EQ_{CFG} is undecidable
- We now prove:
  EQ_{CFG} is co-Turing recognizable
- And conclude that:
  - EQ_{CFG} is not Turing recognizable

Thm: EQ_{CFG} is co-Turing-recognizable

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

Recognizer for $\overline{EQ}_{\text{CFG}}$:

M = On input $\langle G, H\rangle$, where G and H are CFGs:
- For every possible string w:
  Accept if
  - $w \in L(G)$ and $w \notin L(H)$, or
  - $w \notin L(G)$ and $w \in L(G)$
- Else reject

How to compute this?

Use decider for:

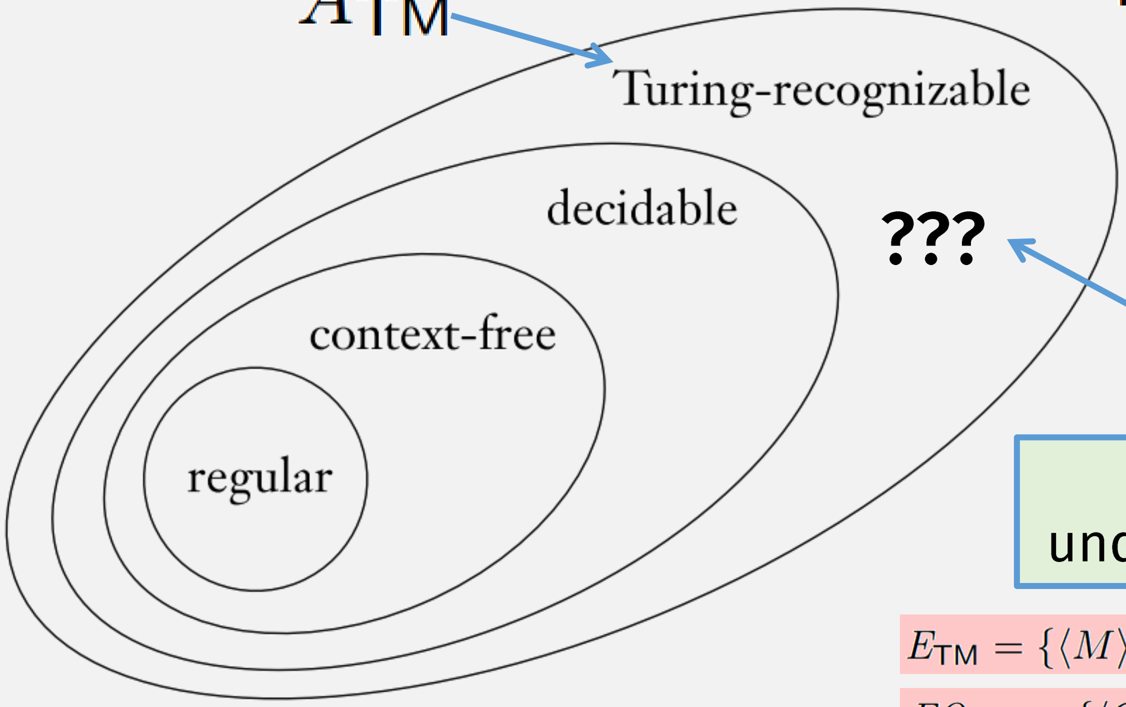
$A_{\text{CFG}} = \{\langle G, w\rangle \mid G \text{ is a CFG that generates string } w\}$

This is only a **recognizer** because it loops forever when $L(G) = L(H)$

Is there anything out here?

$\overline{A_{\mathsf{TM}}}$

A_{TM}



???

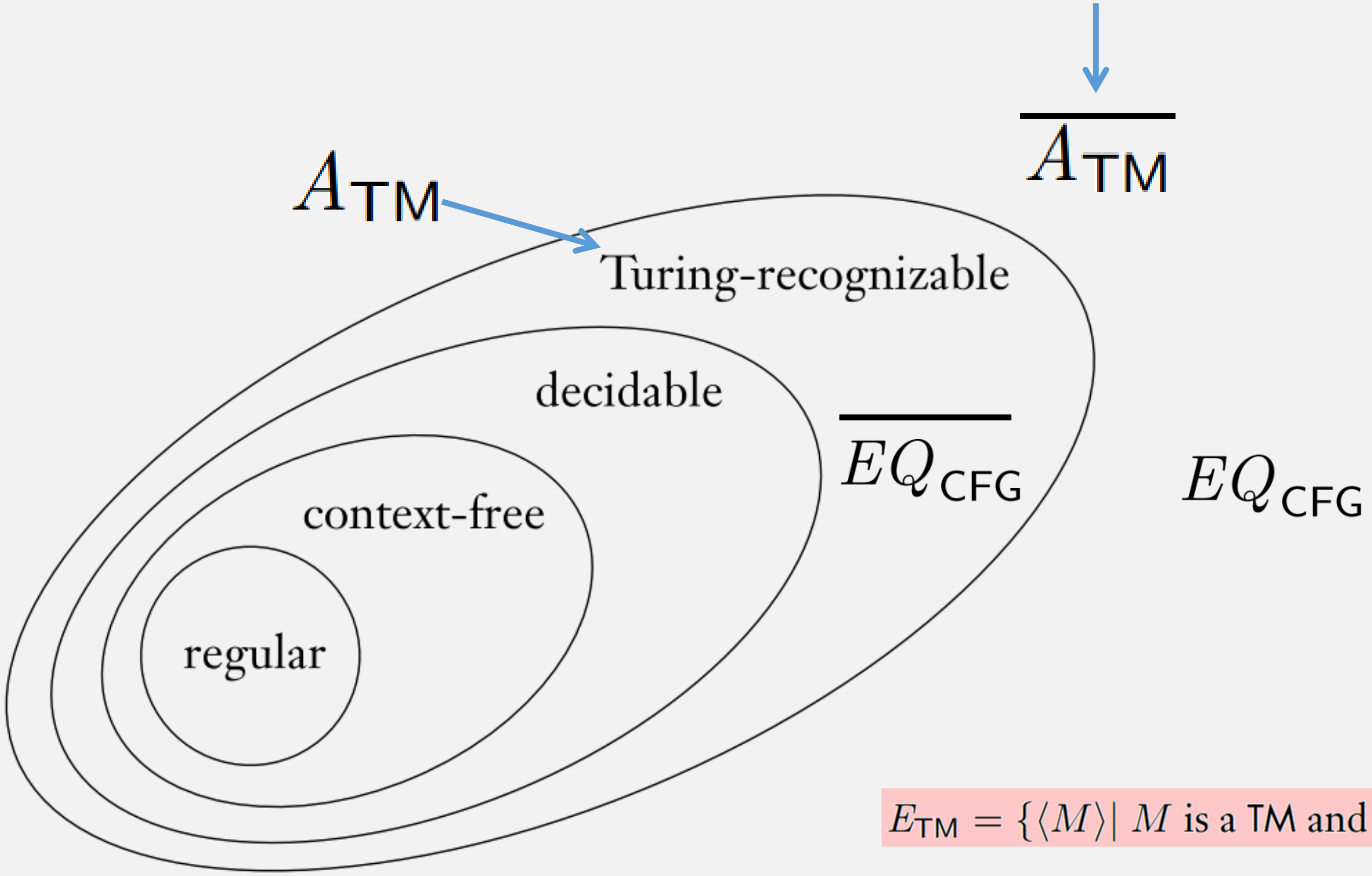
Where do these undecidable languages go?

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

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

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

Is there anything out here?



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

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

Thm: E_{TM} is not Turing-recognizable

Unrecognizability
Proof Technique #1

Recognizable & co-recognizable ⇒ decidable

Contrapositive: undecidable ⇒ can't be both recognizable & co-recognizable

- We've proved:
  - E_{TM} is undecidable
- We now prove:
  E_{TM} is co-Turing recognizable
- And then conclude that:
  - E_{TM} is not Turing recognizable

Thm: E_{TM} is co-Turing-recognizable

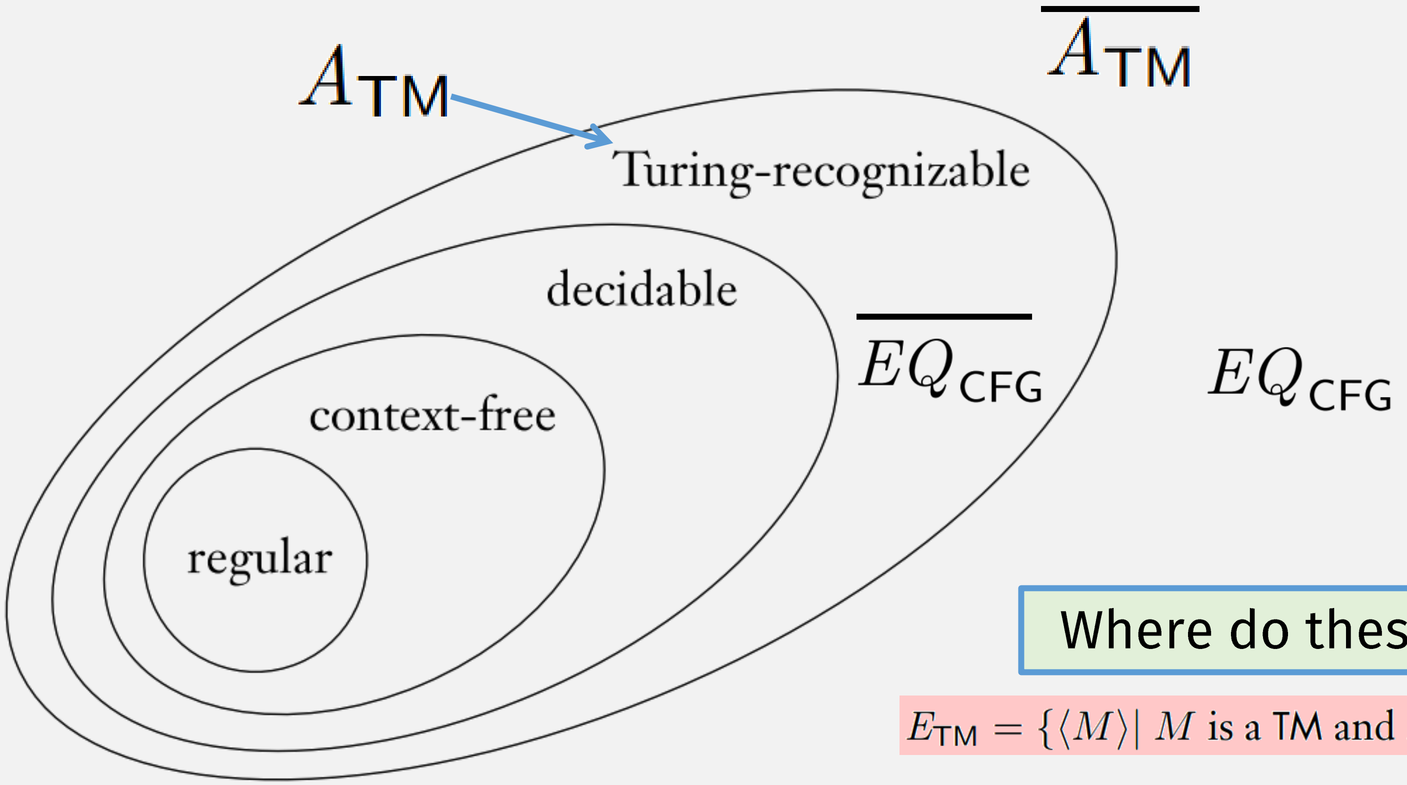
$E_{\mathsf{TM}} = \{\langle M\rangle \mid M \text{ is a TM and } L(M) = \emptyset\}$

Recognizer for $\overline{E_{\mathsf{TM}}}$: Let $s_1, s_2, \ldots$ be a list of all strings in Σ^*

"On input $\langle M\rangle$, where M is a TM:
1. Repeat the following for $\boxed{i} = 1, 2, 3, \ldots$.
2. Run M for $\boxed{i}$ steps on each input, $s_1, s_2, \ldots, \boxed{s_i}$.
3. If M has accepted any of these, *accept*. Otherwise, continue."

This is only a **recognizer** because it loops forever when $L(M)$ is empty

Unrecognizable Languages

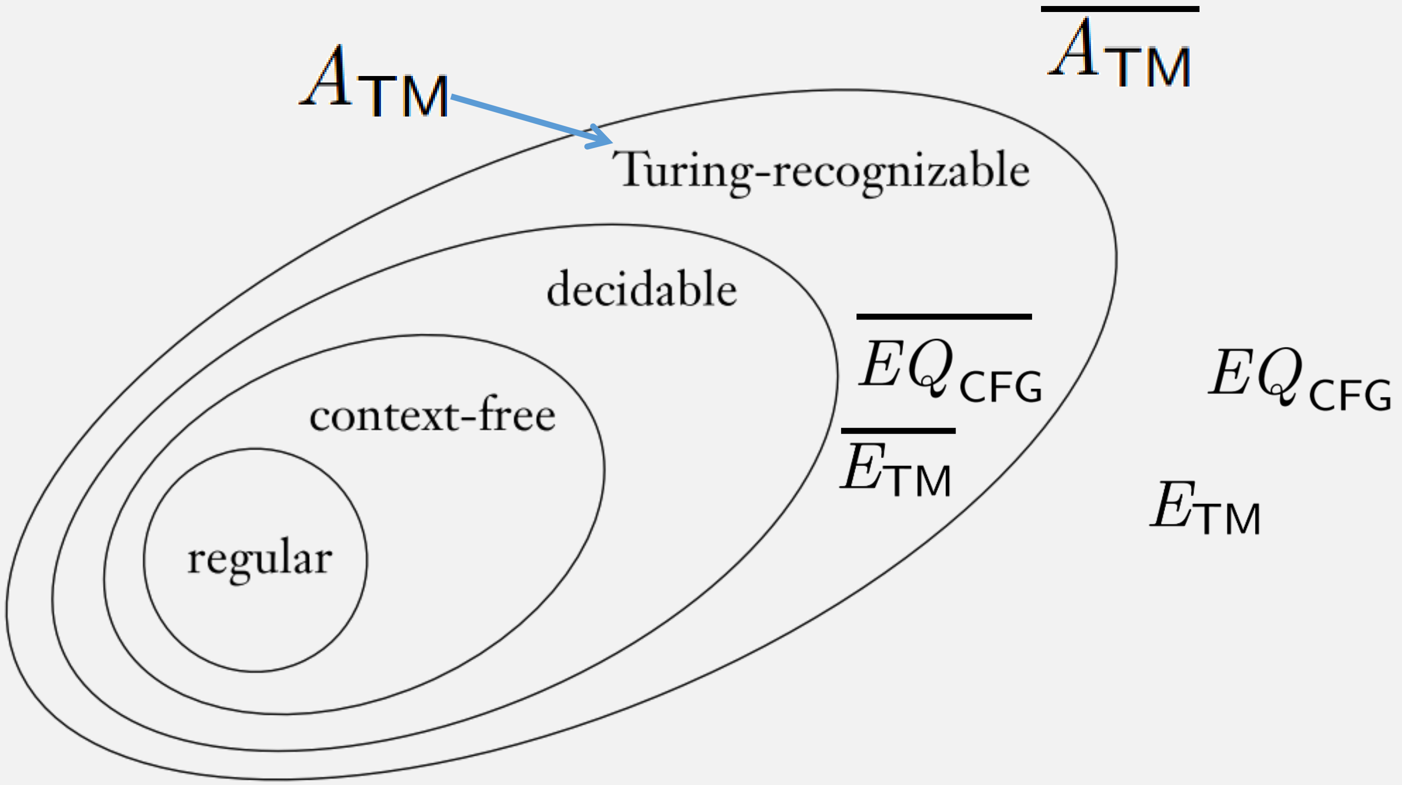


Where do these go?

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

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

Unrecognizable Languages



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

Mapping Reducibility Can be Used to Prove …

- Decidability
- Undecidability
- Recognizability
- Unrecognizability

More Helpful Theorems

If $A \leq_m B$ and B is $\boxed{\text{Turing-recognizable,}}$ then A is Turing-recognizable.

If $A \leq_m B$ and A is not Turing-recognizable, then B is not Turing-recognizable.

Unrecognizability
Proof Technique #2:
Mapping reducibility
+ this theorem

- Same proofs as:

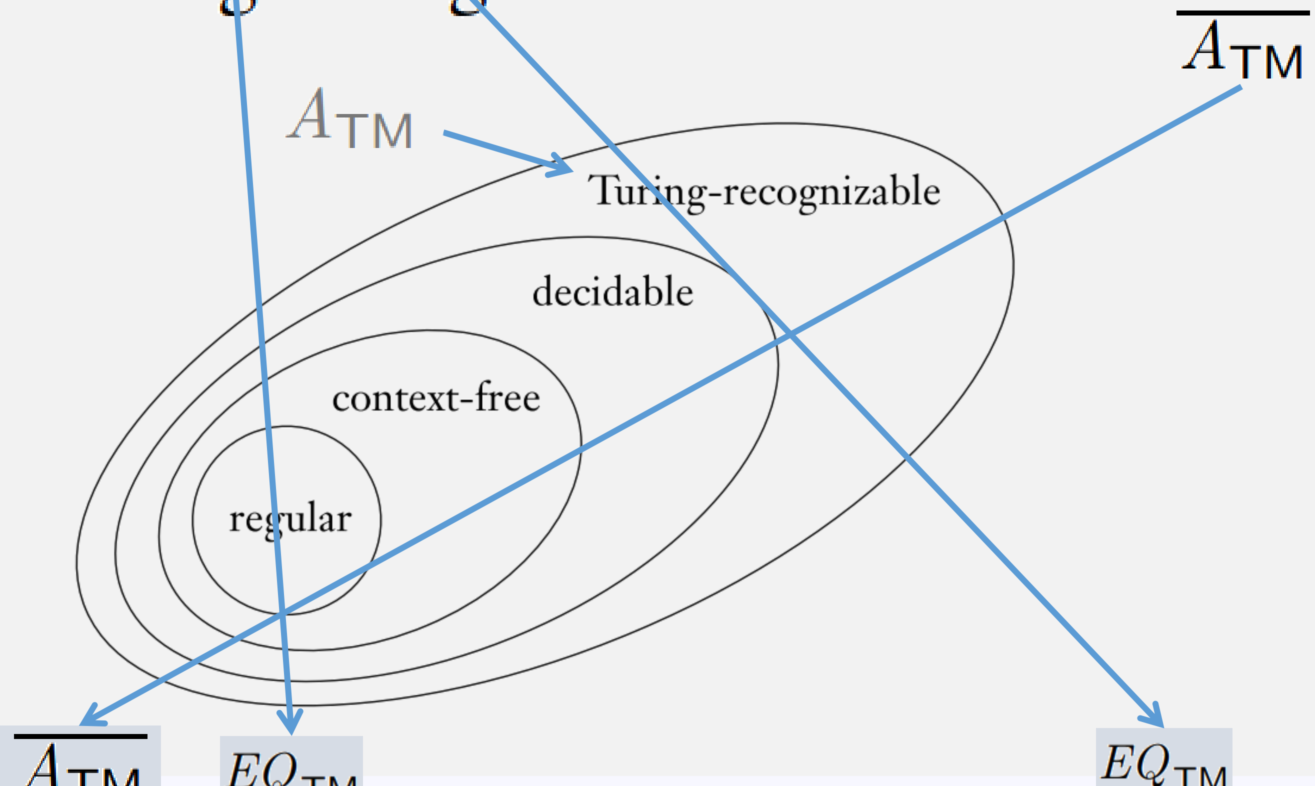
If $A \leq_m B$ and B is $\boxed{\text{decidable,}}$ then A is decidable.

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

Thm: EQ_{TM} is neither Turing-recognizable nor co-Turing-recognizable.

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

1. EQ_{TM} is not Turing-recognizable



Now just have to show this mapping reducibility

If $A \leq_{\mathrm{m}} B$ and A is not Turing-recognizable, then B is not Turing-recognizable.

Mapping Reducibility implies Mapping Red. of Complements

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,

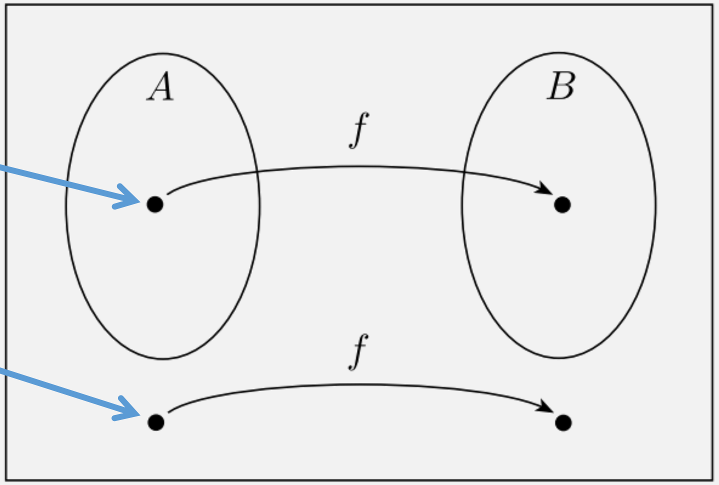
$$\boxed{w \in A \iff f(w) \in B.}$$

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

$\boxed{A \leq_{\mathrm{m}} B}$

implies

$\boxed{\overline{A} \leq_{\mathrm{m}} \overline{B}}$



Thm: EQ_{TM} is neither Turing-recognizable nor co-Turing-recognizable.

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

1. EQ_{TM} is not Turing-recognizable

Two Choices:

- Create Computable fn: $\overline{A_{\mathsf{TM}}} \rightarrow EQ_{\mathsf{TM}}$
- Or Computable fn: $A_{\mathsf{TM}} \rightarrow \overline{EQ_{\mathsf{TM}}}$

Because mapping reducibility implies mapping reducibility of complements

And use theorem ...

If $A \leq_{\mathrm{m}} B$ and A is not Turing-recognizable, then B is not Turing-recognizable.

Thm: EQ_{TM} is not Turing-recognizable

Step 1
Computable fn

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

- Create Computable fn: $A_{\text{TM}} \rightarrow \overline{EQ_{\text{TM}}}$
- $\langle M, w\rangle \rightarrow \langle M_1, M_2\rangle$ M_1 and M_2 are TMs and $L(M_1) \neq L(M_2)$

$F =$ "On input $\langle M, w\rangle$, where M is a TM and w a string:

1. Construct the following two machines, M_1 and M_2.

$M_1 =$ "On any input:

1. *Reject.*"

$M_2 =$ "On any input:

1. Run M on w. If it accepts, *accept*."

$\langle M_1, M_2\rangle$."

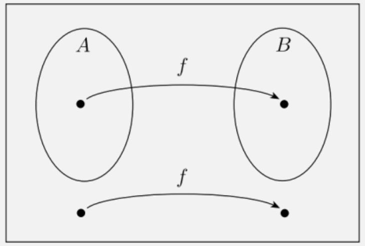
Accepts nothing

Accepts nothing or everything

Step 2, iff:

⇒ If M accepts w, then $M_1 \neq M_2$
- because M_1 accepts nothing
but M_2 accepts everything

⇐ If M does not accept w, then $M_1 = M_2$
- because M_1 accepts nothing
and M_2 accepts nothing



Thm: EQ_{TM} is neither Turing-recognizable nor co-Turing-recognizable.

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

1. EQ_{TM} is not Turing-recognizable

- Create Computable fn: $\overline{A_{\mathsf{TM}}} \rightarrow EQ_{\mathsf{TM}}$
- Or Computable fn: $A_{\mathsf{TM}} \rightarrow \overline{EQ_{\mathsf{TM}}}$
- **DONE!**

And use theorem …

If $A \leq_{\mathrm{m}} B$ and A is not Turing-recognizable, then B is not Turing-recognizable.

(Definition of co-Turing-recognizable)

2. $\overline{EQ_{\mathsf{TM}}}$ is not ~~co~~-Turing-recognizable

- (A lang is co-Turing-recog. if it is complement of Turing-recog. lang)

<u>Previous:</u> EQ_{TM} is not Turing-recognizable

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

- Create <u>Computable fn</u>: A_{TM} → $\overline{EQ_{\mathsf{TM}}}$

<u>Step 1</u>

- $\langle M, w\rangle$ → $\langle M_1, M_2\rangle$ M_1 and M_2 are TMs and $L(M_1) \neq L(M_2)$

F = "On input $\langle M, w\rangle$, where M is a TM and w a string:

1. Construct the following two machines, M_1 and M_2.

   M_1 = "On any input: ← Accepts nothing

   1. *Reject*."

   M_2 = "On any input: ← Accepts nothing or everything

   1. Run M on w. If it accepts, *accept*."

2. Output $\langle M_1, M_2\rangle$."

Now: $\overline{EQ}_{\mathsf{TM}}$ is not Turing-recognizable

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

- Create Computable fn: $A_{\mathsf{TM}} \rightarrow \overline{EQ}_{\mathsf{TM}}$

Step 1
- $\langle M, w\rangle \rightarrow \langle M_1, M_2\rangle$ M_1 and M_2 are TMs and $L(M_1) \neq L(M_2)$

$F =$ "On input $\langle M, w\rangle$, where M is a TM and w a string:

1. Construct the following two machines, M_1 and M_2.

   $M_1 =$ "On any input: ← Accepts ~~nothing~~ everything

   1. *Accept*."

   $M_2 =$ "On any input: ← Accepts nothing or everything

   1. Run M on w. If it accepts, *accept*."

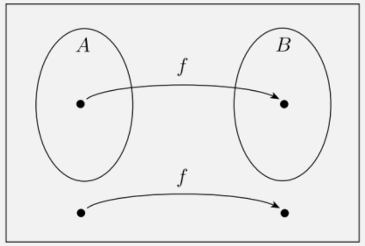
2. Output $\langle M_1, M_2\rangle$."

Step 2, iff:

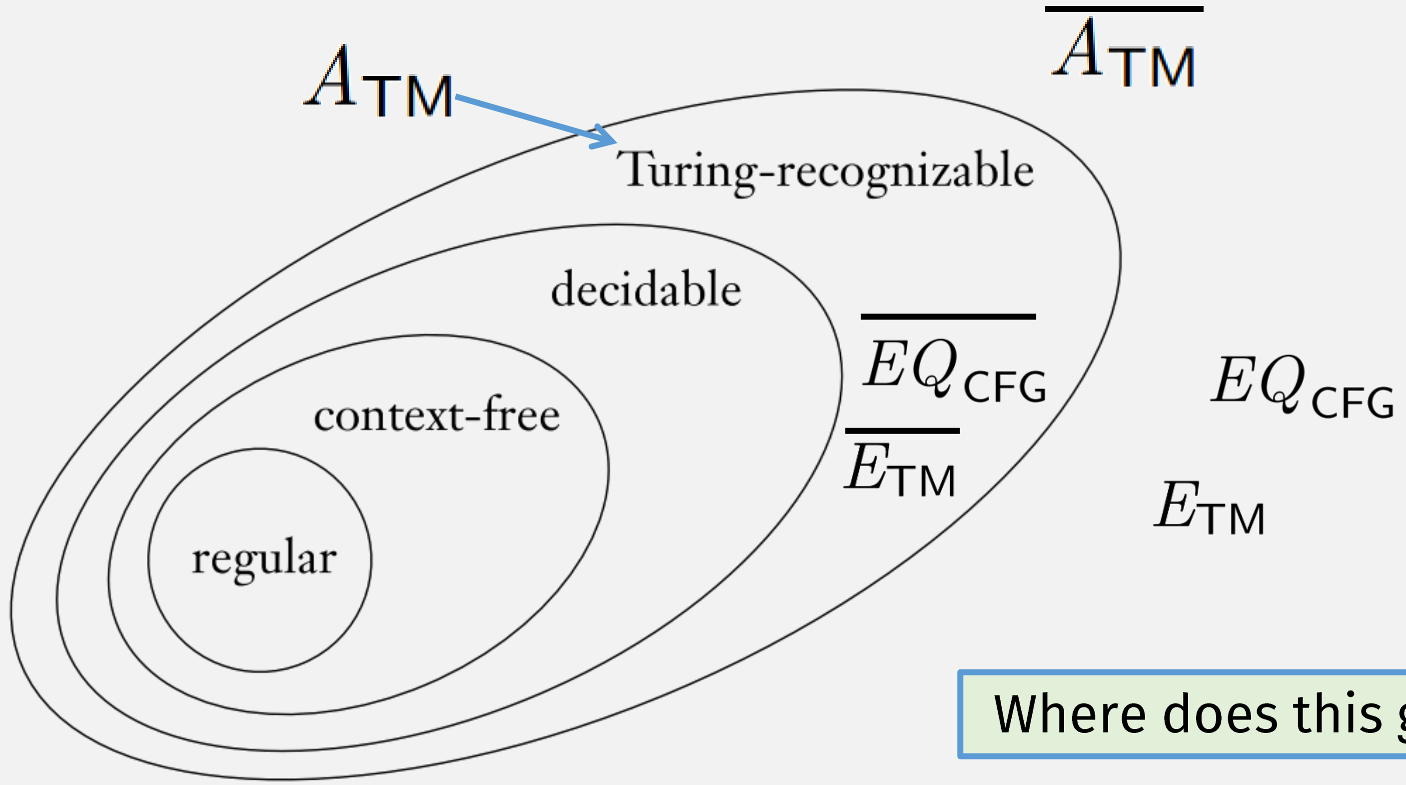
⇒ If M accepts w, then $M_1 \boxed{=} M_2$

⇐ If M does not accept w, then $M_1 \boxed{\neq} M_2$

DONE!



Unrecognizable Languages



Where does this go?

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

Unrecognizable Languages