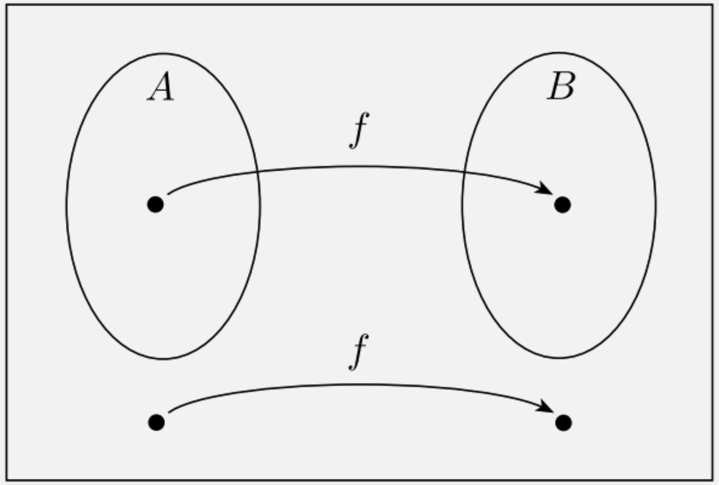
UMB CS622

Mapping Reducibility & Unrecognizability

Wednesday, October 27, 2021



Announcements

- HW6 due date extended
  - Due Wed 11/3 11:59pm

- New <u>required</u> reading:
  - Piazza posts about induction

Last Time: Undecidability By Checking TM Configs

$ALL_{\text{CFG}} = \{\langle G \rangle \mid G \text{ is a CFG and } L(G) = \Sigma^*\}$

Proof, by contradiction

- Assume ALL_{CFG} has a decider R. Use it to create decider for A_{TM}:

On input <M, w>:

- Construct a PDA P that rejects sequences of M configs that accept w
- Convert P to a CFG G (prev class)
- Give G to R:
  - If R accepts, then M has no accepting config sequences for w, so reject
  - If R rejects, then M has an accepting config sequence for w, so accept

Any machine that can validate TM config sequences could be used to prove undecidability?

Last Time: Algorithms For CFLs

- $A_{\mathsf{CFG}} = \{\langle G, w\rangle \mid G \text{ is a CFG that generates string } w\}$ — Decidable
- $E_{\mathsf{CFG}} = \{\langle G\rangle \mid G \text{ is a CFG and } L(G) = \emptyset\}$ ← Why is this **decidable?** — Decidable
- $ALL_{\mathsf{CFG}} = \{\langle G\rangle \mid G \text{ is a CFG and } L(G) = \Sigma^*\}$ ← But this is **undecidable?** — **Undecidable**

Last time: Exploring the Limits of CFLs

- This is a CFL: $\{w_1\#w_2 \mid w_1 \neq w_2\}$
  - PDA nondeterministically checks matching positions in 1st/2nd parts
  - And rejects if **any** pair of chars are not the same
  - I.e., Each branch is "context free"

This is like the TM-config-rejecting PDA used to prove ALL_{CFG} undecidable

- This is not a CFL: $\{w_1\#w_2 \mid w_1 = w_2\}$
  - Can nondeterministically check matching positions
  - But needs to accept only if **all** branches match
  - I.e., each branch is not "context free"

There's no TM-config-accepting PDA because this language is not a CFL! So it's ok that E_{CFG} is decidable

This is similar to the *ww* language (not pumpable)

Summary: CFLs cannot do (stack-based) nondet. computation where a branch depends on other branch results

(This is also why **union is closed for CFLs** but **intersection is not**)

Last time: Algorithms For CFLs

- $A_{\text{CFG}} = \{\langle G, w\rangle \mid G \text{ is a CFG that generates string } w\}$ Decidable
- $E_{\text{CFG}} = \{\langle G\rangle \mid G \text{ is a CFG and } L(G) = \emptyset\}$ Decidable
- $ALL_{\text{CFG}} = \{\langle G\rangle \mid G \text{ is a CFG and } L(G) = \Sigma^*\}$ **Undecidable**
- $EQ_{\text{CFG}} = \{\langle G, H\rangle \mid G \text{ and } H \text{ are CFGs and } L(G) = L(H)\}$ **Undecidable?**

(Still need to prove this is undecidable)

Theorem: EQ_{CFG} is undecidable

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

Proof by contradiction: Assume EQ_{CFG} has a decider R

- Use R to create a decider for ALL_{CFG}:

On input <G>:

- Construct a CFG G_{ALL} which generates all possible strings
- Run R (EQ_{CFG}'s decider) on <G, G_{ALL}>
- Accept G if R accepts, else reject

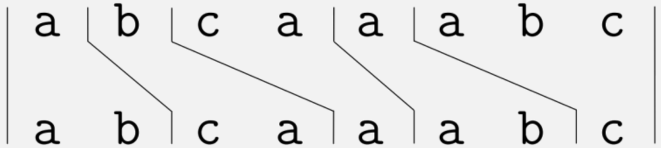
The Post Correspondence Problem (PCP)

A Non-Formal Languages Undecidable Problem: *PCP*

- Let P be a set of "**dominos**" $\left\{ \left[\frac{t_1}{b_1}\right], \left[\frac{t_2}{b_2}\right], \ldots, \left[\frac{t_k}{b_k}\right] \right\}$
  - Where each t_i and b_i are strings
  - E.g., $P = \left\{ \left[\frac{\mathtt{b}}{\mathtt{ca}}\right], \left[\frac{\mathtt{a}}{\mathtt{ab}}\right], \left[\frac{\mathtt{ca}}{\mathtt{a}}\right], \left[\frac{\mathtt{abc}}{\mathtt{c}}\right] \right\}$

- A **match** is:
  - A sequence of dominos with the same top and bottom strings

    Repeats allowed
  - E.g., $\left[\frac{\mathtt{a}}{\mathtt{ab}}\right]\left[\frac{\mathtt{b}}{\mathtt{ca}}\right]\left[\frac{\mathtt{ca}}{\mathtt{a}}\right]\left[\frac{\mathtt{a}}{\mathtt{ab}}\right]\left[\frac{\mathtt{abc}}{\mathtt{c}}\right]$ ➡



- Then: *PCP* = { <*P*> | *P* is a set of dominos with a match }

Theorem: PCP is undecidable

Proof by contradiction:

Assume PCP has a decider R and use to create decider for A_{TM}:

On input <M, w>:
1. Construct a set of dominos P that has a match only when M accepts w
2. Run R with P as input
3. Accept if R accepts, else reject

P has M's TM configurations as its domino strings

A match is a sequence of configs showing M accepting w!

$$M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$$

PCP Dominos

- First domino: $\left[\frac{\#}{\# q_0 w_1 w_2 \cdots w_n \#}\right]$
- Key idea: add dominos representing valid TM steps:

$$\text{if } \delta(q, a) = (r, b, \text{R}), \text{ put } \left[\frac{qa}{br}\right] \text{ into } P$$
$$\text{if } \delta(q, a) = (r, b, \text{L}), \text{ put } \left[\frac{cqa}{rcb}\right] \text{ into } P$$

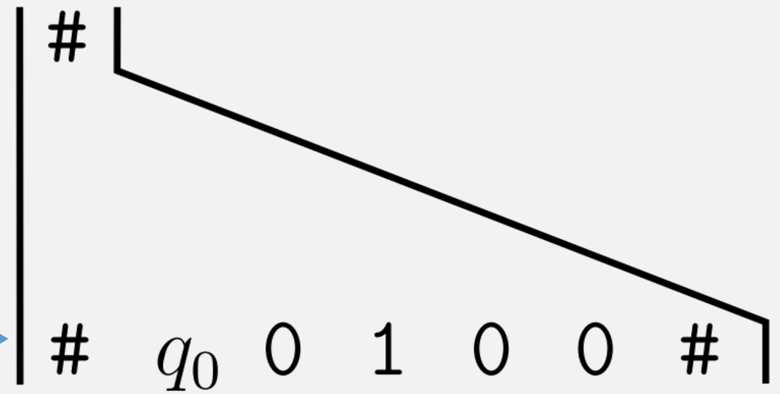
- For the tape cells that don't change: $\text{put } \left[\frac{a}{a}\right] \text{ into } P$
- Top can only "catch up" if there is an accepting config sequence

PCP Example

- Let w = `0100` and $\delta(q_0, 0) = (q_7, 2, \mathrm{R})$ so $\left[\dfrac{q_0 0}{2q_7}\right]$ in P

Top starts to catch up

Bottom is valid transition



Fill in with unchanged tape

PCP Dominos (accepting)

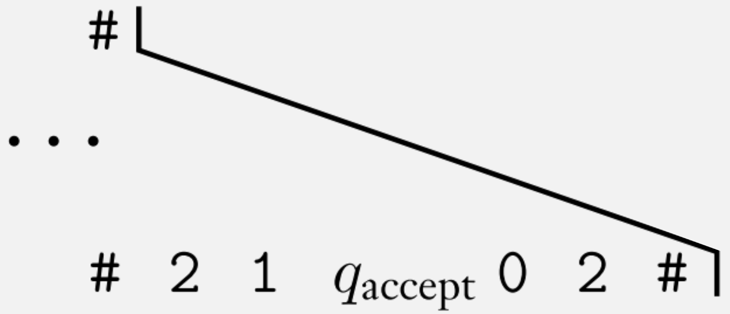
- When accept state reached, let top "catch" up:

For every $a \in \Gamma$,

$$\text{put } \left[\frac{a\, q_{\text{accept}}}{q_{\text{accept}}}\right] \text{ and } \left[\frac{q_{\text{accept}}\, a}{q_{\text{accept}}}\right] \text{ into } P$$

Bottom "eats" one char

Only possible match is accepting sequence of TM configs



"eat" one char

Mapping Reducibility

Flashback: “Reduced”

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

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

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

PROBLEM: What if it takes forever to create this decider?

Proof, by contradiction:

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

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

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

We need a formal definition of “reducibility”

Flashback: A_{NFA} is a decidable language

$A_{\mathsf{NFA}} = \{\langle B, w\rangle \mid B$ 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 TM, but it doesn't accept/reject?

More generally, we've been saying "**programs = TMs**", but programs do more than accept/reject?

Computable Functions

- A TM that, instead of accept/reject, "outputs" final tape contents

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.

- Example 1: All arithmetic operations

- Example 2: Converting between machines, like **DFA→NFA**
  - E.g., adding states, changing transitions, wrapping TM in TM, etc.

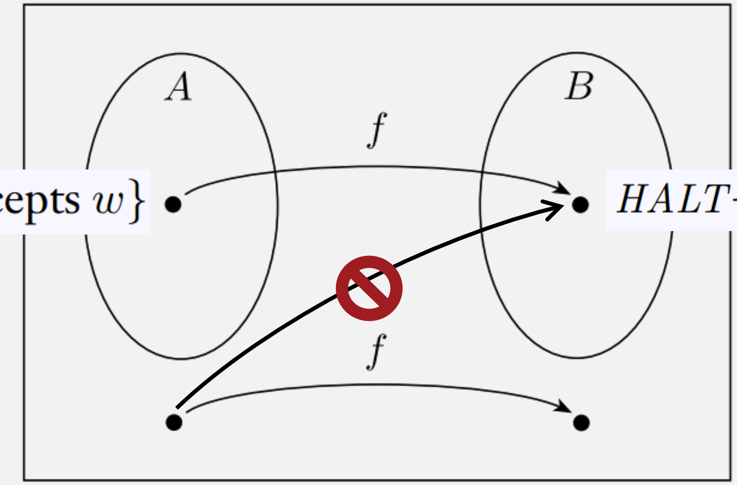
Mapping Reducibility

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.

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



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

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.

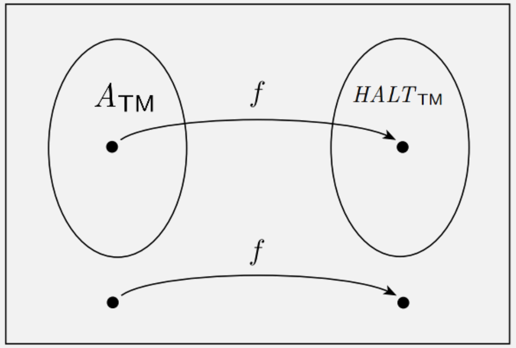
Thm: A_{TM} is mapping reducible to $HALT_{\mathsf{TM}}$

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

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

- To show: $A_{\mathsf{TM}} \leq_m HALT_{\mathsf{TM}}$
- Want: computable fn f : $\langle M, w\rangle \rightarrow \langle M', w'\rangle$ where:

$$\langle M, w\rangle \in A_{\mathsf{TM}} \text{ if and only if } \langle M', w'\rangle \in HALT_{\mathsf{TM}}$$



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'

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

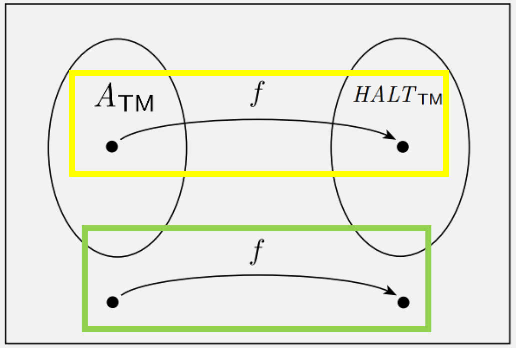
⇒ If M accepts w, then M' halts on w

- 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
  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 $\langle M', w\rangle$."

Use Mapping Reducibility to Prove …

- Decidability

- Undecidability

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

Has a decider

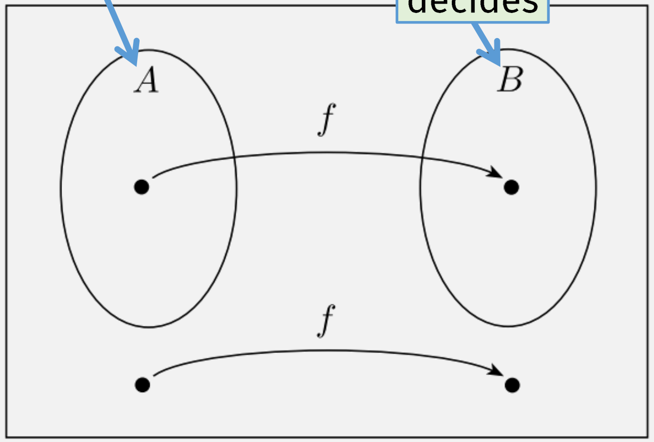
PROOF We let M be the decider for B and f be the reduction from A to B. We describe a decider N for A as follows.

N = "On input w:

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

decides

decides



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: Mapping Reducibility Theorems

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

Known

Unknown

- If $A \leq_m B$ 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.
- $A_{\mathsf{TM}} \leq_{\mathrm{m}} HALT_{\mathsf{TM}}$
- Since A_{TM} is undecidable, then $HALT_{\mathsf{TM}}$ is undecidable

Flashback: 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 to create E_{TM} *decider*:

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

> 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*."

Alternate proof: Show: $E_{\mathsf{TM}} \leq_{\mathrm{m}} EQ_{\mathsf{TM}}$

- Computable fn f : $\langle M\rangle$ → $\langle M, M_1\rangle$

Last step: show iff requirements of mapping reducibility (exercise)

Reducing to complement: E_{TM} is undecidable

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

Proof, by contradiction:

- Assume E_{TM} has *decider* R; use 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}!

Alternate proof: computable fn: $\langle M, w \rangle \rightarrow \langle M_1 \rangle$???

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

Last step: show iff requirements of mapping reducibility (exercise)

Undecidable Langs Closed under Complement

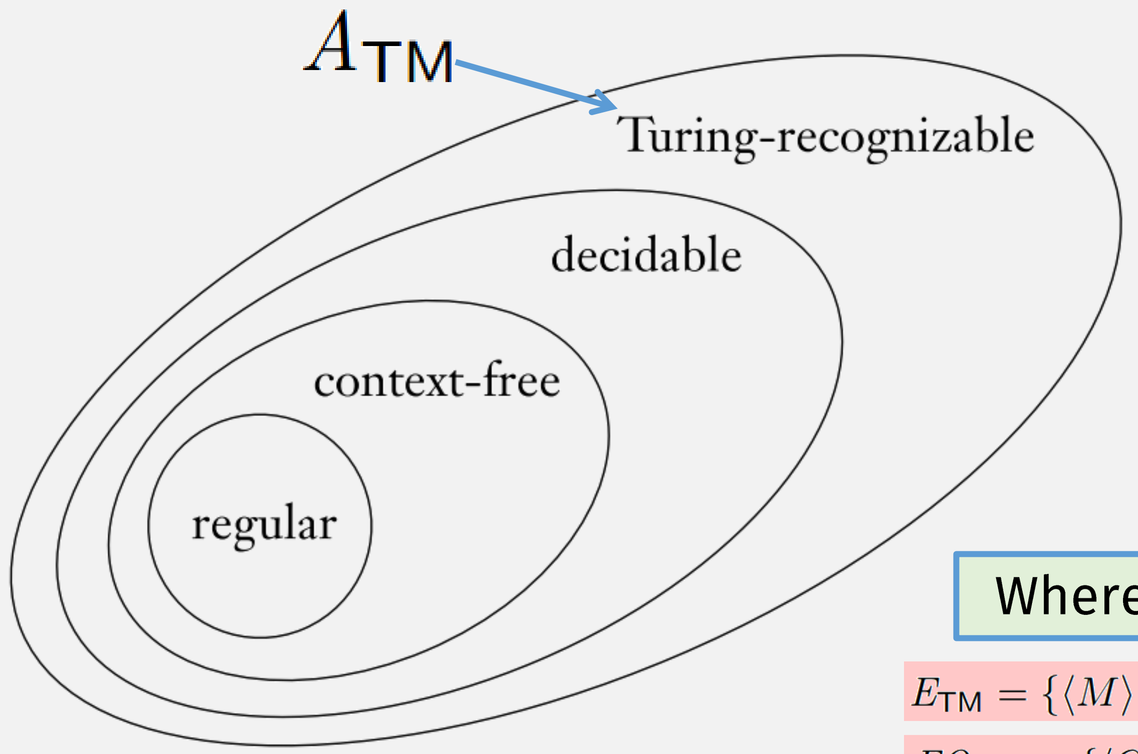
- E.g., if L is undecidable and $\overline{L}$ is decidable …
- … then we can create decider for L from decider for $\overline{L}$ …
- … which is a contradiction!

- Because decidable languages are closed under complement!

Unrecognizability

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 (pigeonhole principle)

Mapping a Language to a Binary Sequence

All Possible Strings

Some Language (subset of above)

Its (unique) Binary Sequence

$$
\begin{array}{lllllllllllll}
\Sigma^* & = \{ & \varepsilon, & 0, & 1, & 00, & 01, & 10, & 11, & 000, & 001, & \cdots & \} \\
A & = \{ & & 0, & & 00, & 01, & & & 000, & 001, & \cdots & \} \\
\chi_A & = & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & \cdots &
\end{array}
$$

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)

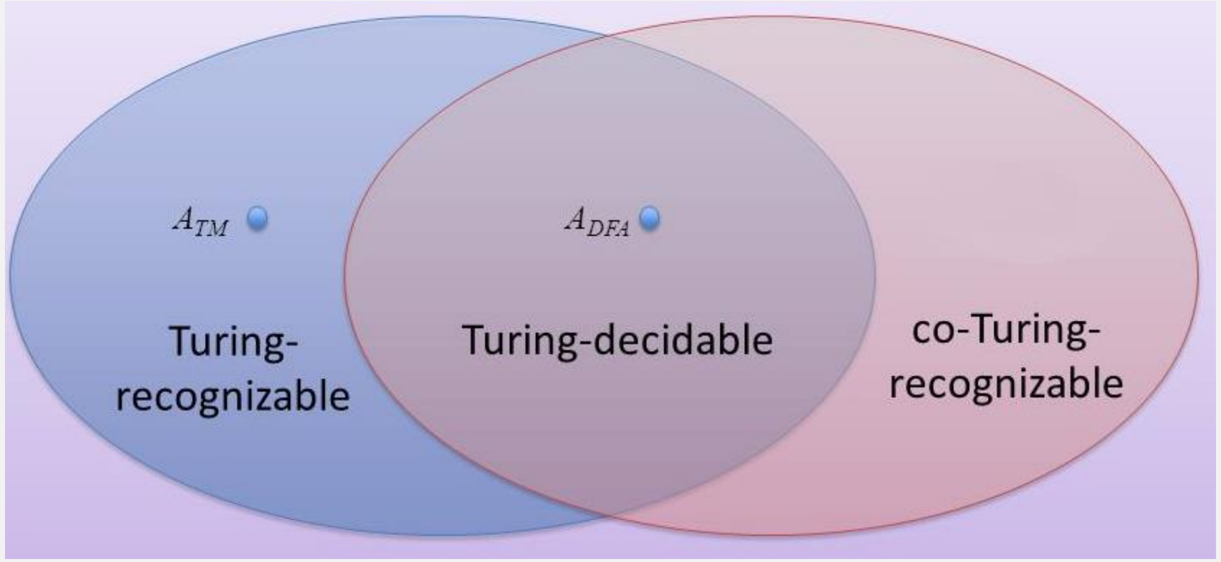
- Lemma 2: The **set of all TMs** is *countable*
  - Because every TM M can be encoded as a string $<M>$
  - And set of all strings is countable

- Therefore, some language is not recognized by a TM ∎

Co-Turing-Recognizability

- A language is **co-Turing-recognizable** if …
- … it is the complement 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 (hw5):
  - A decider is just a recognizer that halts
- 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 (hw5):
  - A decider is just a recognizer that halts
- 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

Recognizable & co-recognizable implies decidable

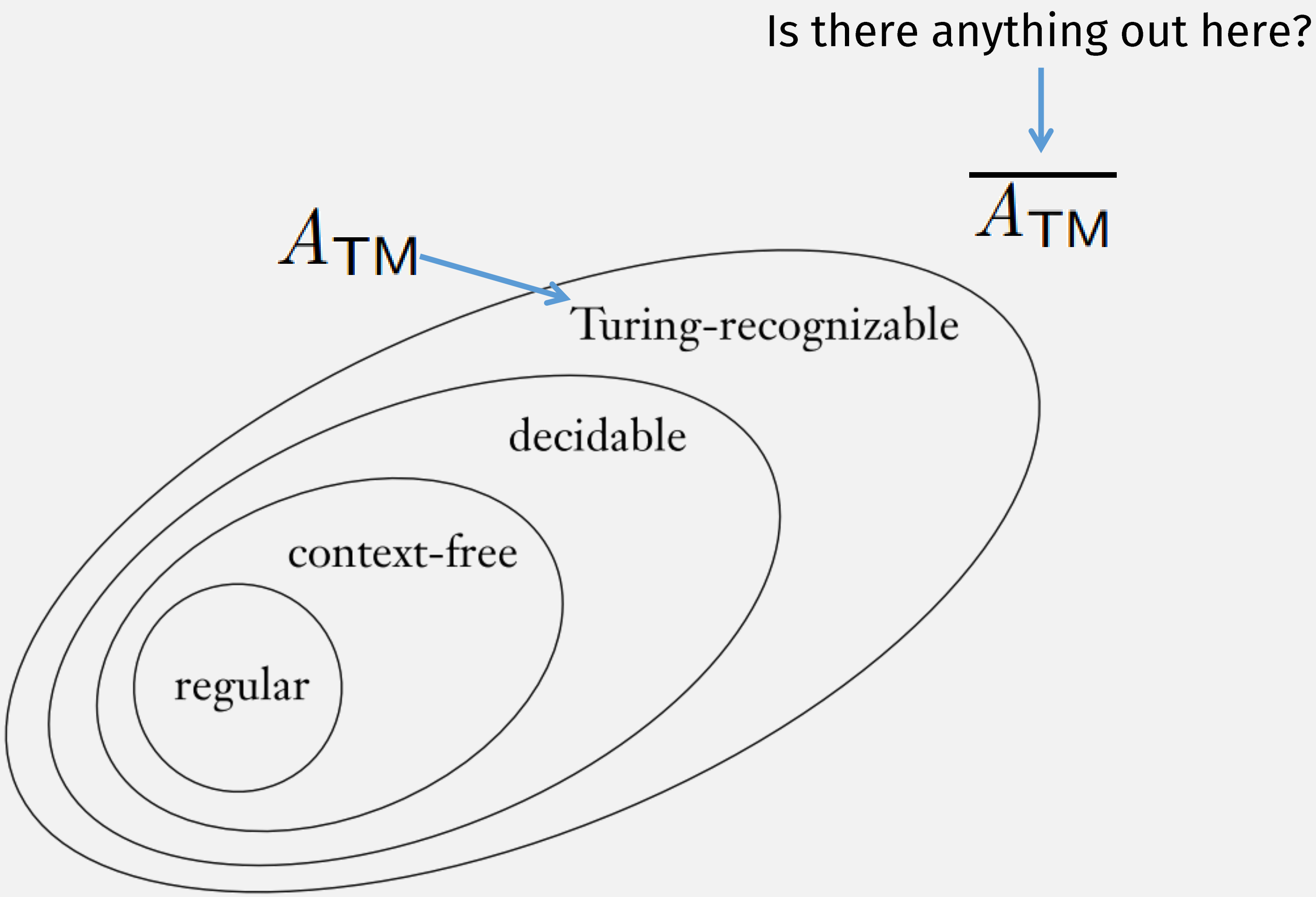
- We've proved:

  A_{TM} is Turing-recognizable

  A_{TM} is undecidable

- So:

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


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

Use Mapping Reducibility to Prove …

- Decidability
- Undecidability
- Recognizability
- Unrecognizability

More Helpful Theorems

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

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

- Same proofs as:

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

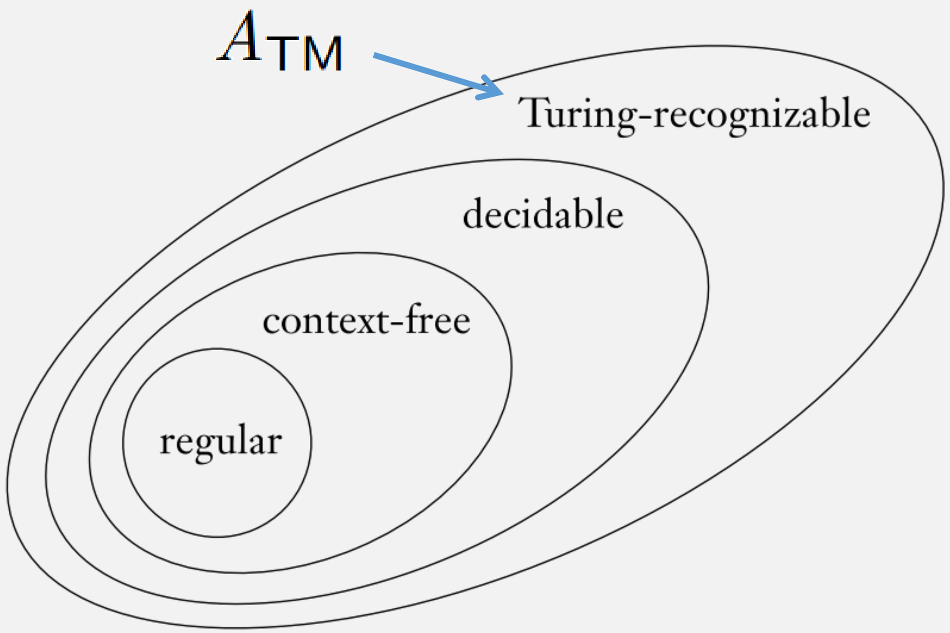
If $A \leq_m B$ and A is undecidable, then B is undecidable.

<u>Thm</u>: 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

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



$\overline{A_{\mathsf{TM}}} \leq_{\mathrm{m}} EQ_{\mathsf{TM}}$ A is not Turing-recognizable, th EQ_{TM} not Turing-recognizable.

Mapping Reducibility implies Mapping Red. of Complements

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,

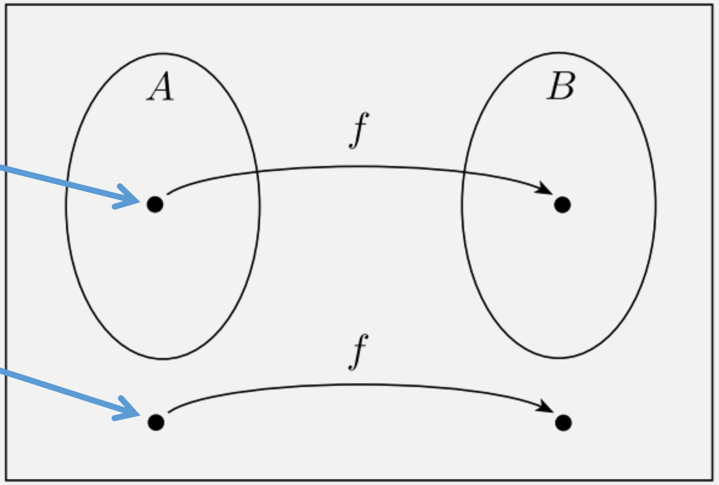
$$w \in A \iff f(w) \in B.$$

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

$A \leq_m B$

implies

$\overline{A} \leq_m \overline{B}$



<u>Thm</u>: 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

Two Choices:

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

Thm: 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 Computable fn: $A_{\mathsf{TM}} \rightarrow \overline{EQ_{\mathsf{TM}}}$
- $\langle M, w\rangle \rightarrow \langle M_1, M_2\rangle$ 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: ← Accepts nothing

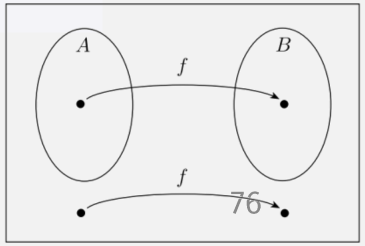
   1. *Reject*."

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

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

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

- If M accepts w, M_1 not equal to M_2
- If M does not accept w, M_1 equal to M_2



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

2. $\overline{EQ_{\mathsf{TM}}}$ is not ~~co~~-Turing-recognizable
   - (A lang is co-Turing-recog. if it is complement of Turing-recog. lang)

Prev: 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 Computable fn: A_{TM} → $\overline{EQ_{\mathsf{TM}}}$
- $\langle M, w\rangle$ → $\langle M_1, M_2\rangle$ 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 $\langle M_1, M_2\rangle$."

DONE!

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

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

- Create <u>Computable fn</u>: $A_{\mathsf{TM}} \rightarrow \overline{EQ}_{\mathsf{TM}}$
- $\langle M, w\rangle \rightarrow \langle M_1, M_2\rangle$ 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: ← Accepts ~~nothing~~ everything

1. *Accept*."

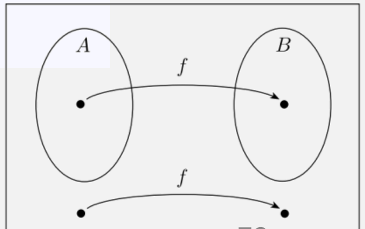
M_2 = "On any input: ← Accepts nothing or everything

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

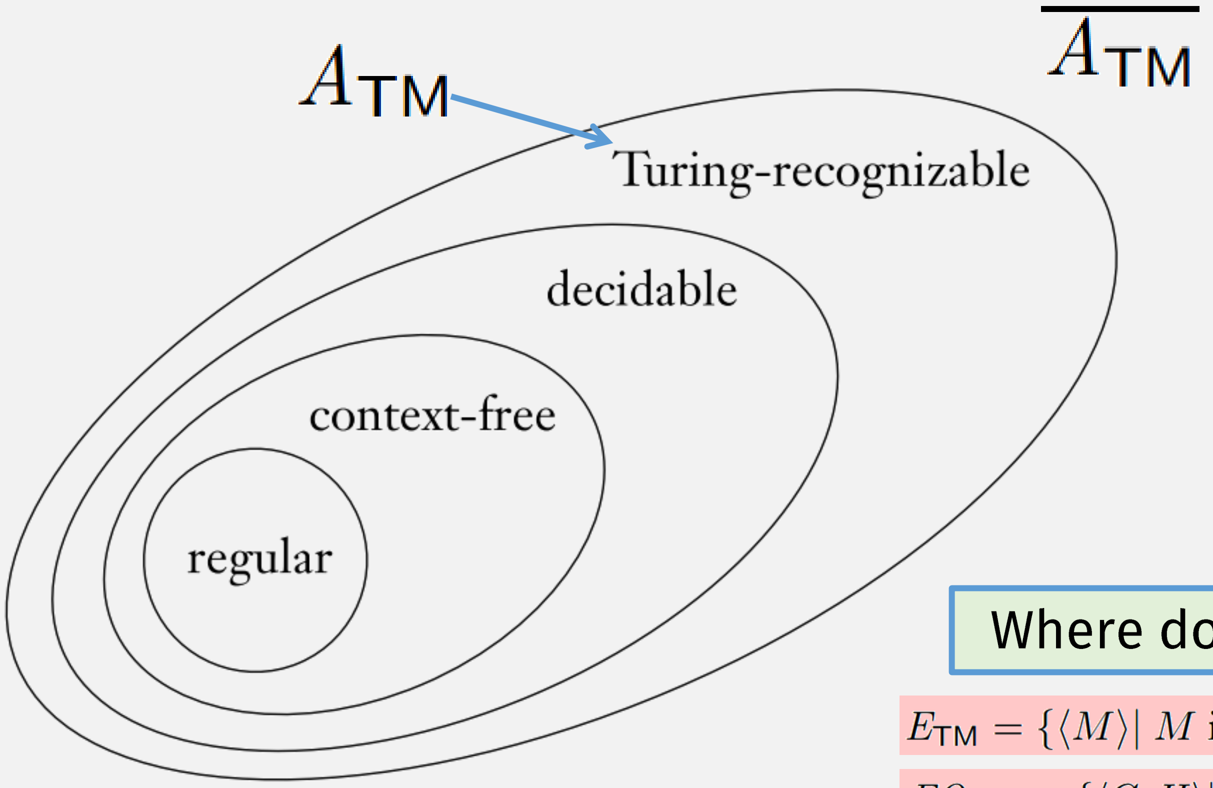
2. Output $\langle M_1, M_2\rangle$."

- If M accepts w, M_1 equals to M_2
- If M does not accept w, M_1 not equal to M_2

DONE!



Unrecognizable Languages?



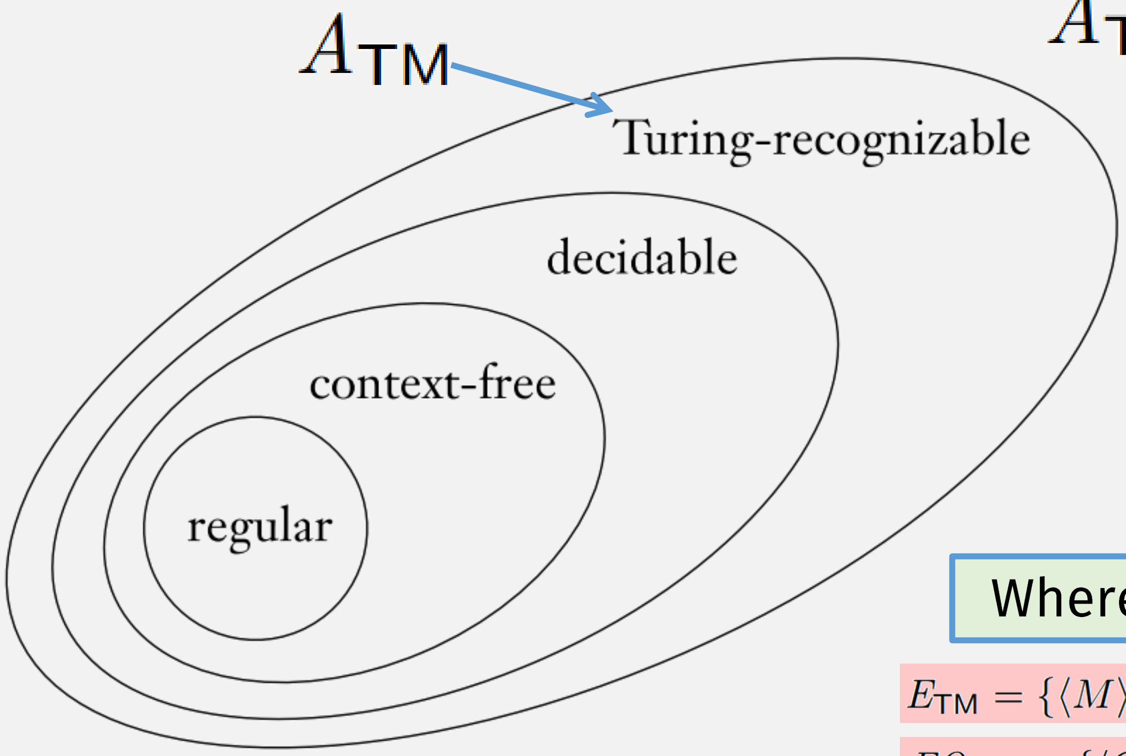
Where do these go?

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

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

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

Unrecognizable Languages



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

EQ_{TM}

$\overline{EQ_{\mathsf{TM}}}$

Where do these go?

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

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

Thm: EQ_{CFG} is not Turing-recognizable

Recognizable & co-recognizable implies decidable

- We've proved:

  EQ_{CFG} is undecidable

- We now prove:

  EQ_{CFG} is co-Turing recognizable

- So:
  - EQ_{CFG} is not Turing recognizable

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

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

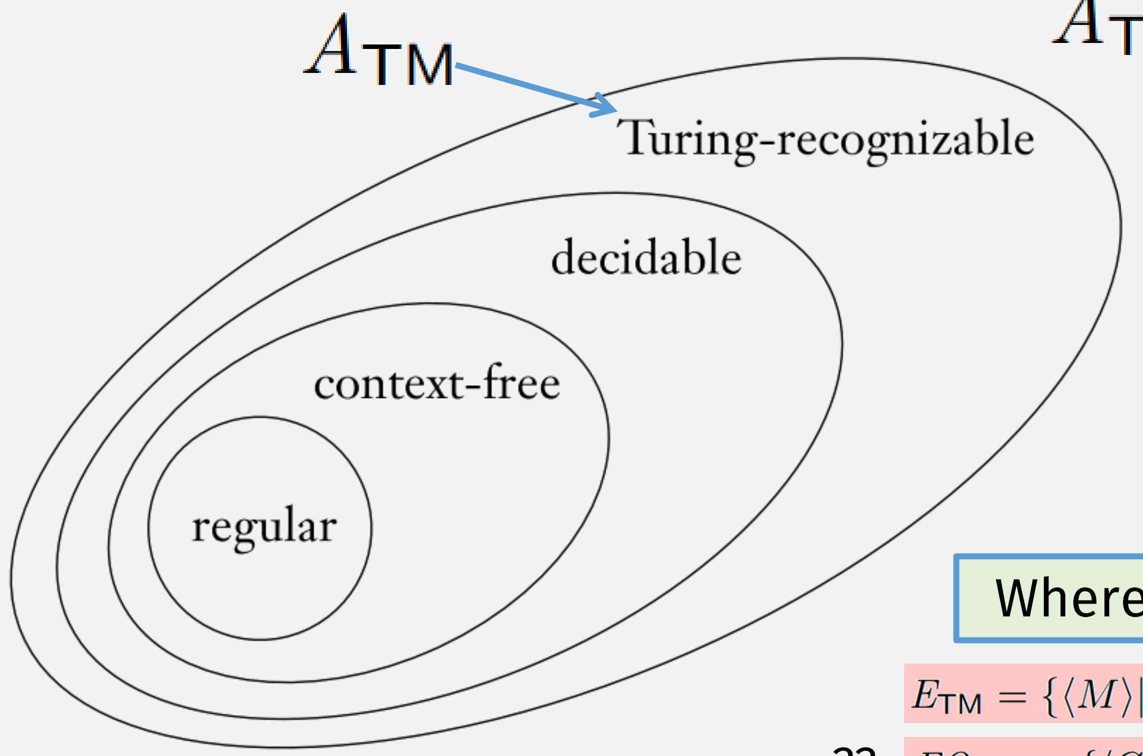
Recognizer for $\overline{EQ}_{\text{CFG}}$:

- On input <G, H>:
  - For every possible string w:
    - Accept if $w \in L(G)$ and $w \notin L(H)$
    - Or accept if $w \in L(H)$ and $w \notin L(G)$
  - Else reject

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

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

Unrecognizable Languages



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

EQ_{TM}

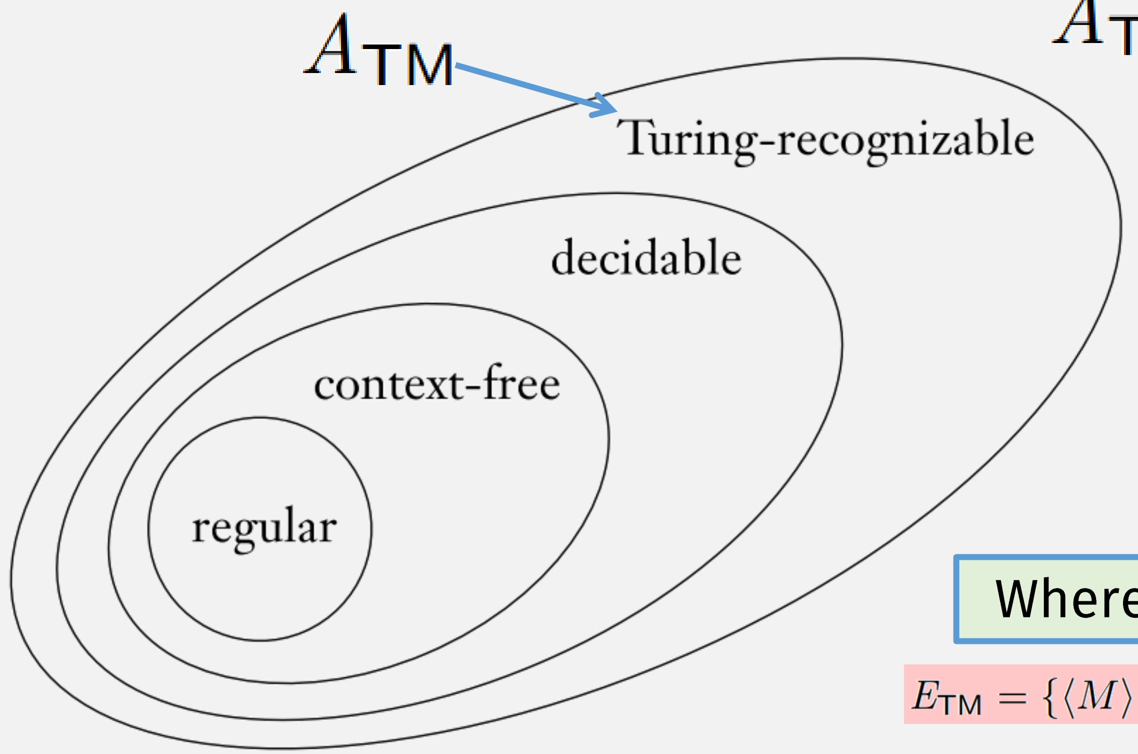
$\overline{EQ_{\mathsf{TM}}}$

Where do these go?

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

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

Unrecognizable Languages



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

EQ_{TM}

$\overline{EQ_{\mathsf{TM}}}$

EQ_{CFG}

Where do these go?

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

Thm: E_{TM} is not Turing-recognizable

Recognizable & co-recognizable implies decidable

- We've proved:
  - E_{TM} is undecidable
- We now prove:
  E_{TM} is co-Turing recognizable
- So:
  - E_{TM} is not Turing recognizable

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

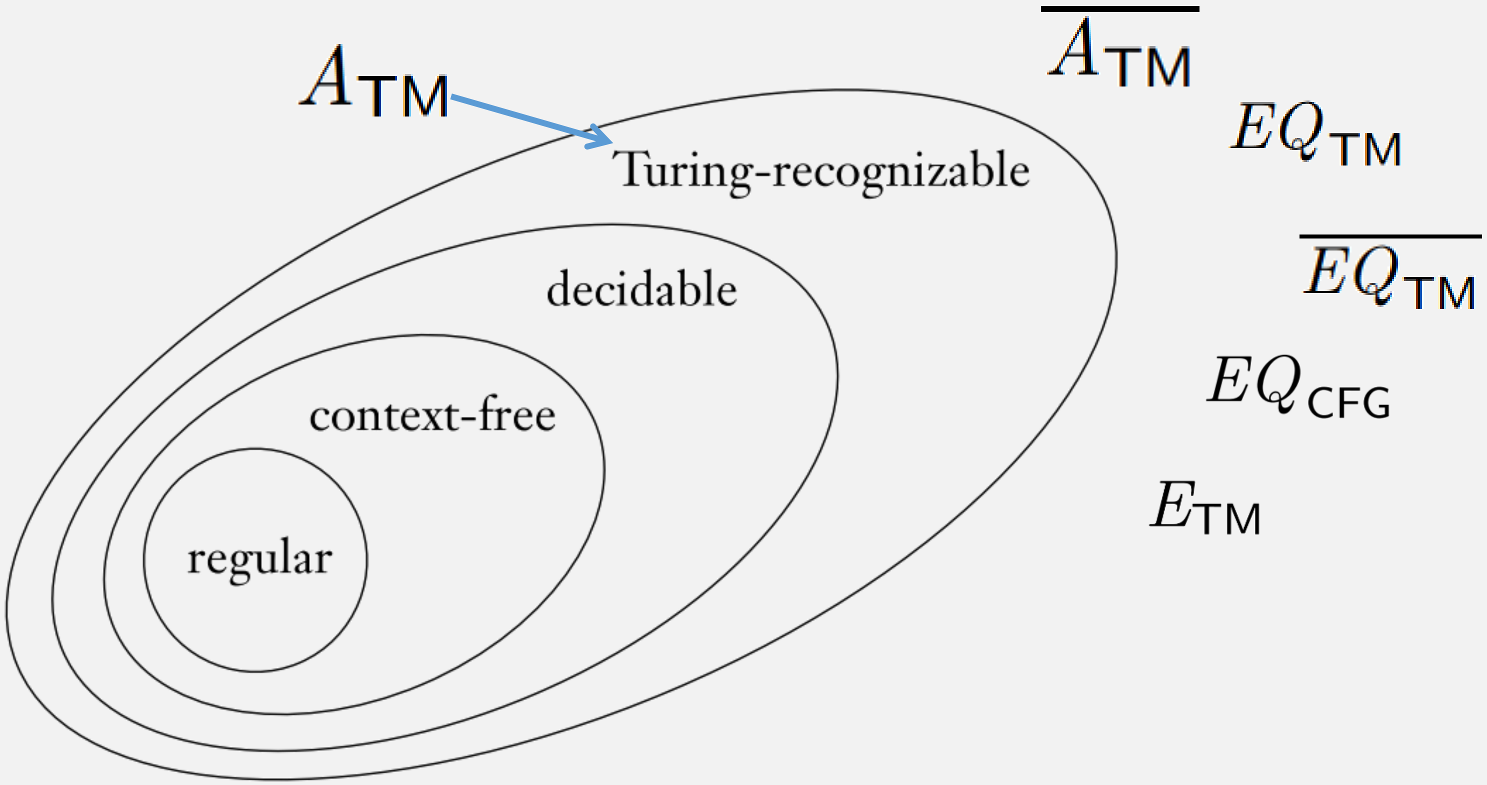
$E_{\mathsf{TM}} = \{\langle M \rangle \mid M \text{ is a TM and } L(M) = \emptyset\}$

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 $i = 1, 2, 3, \ldots$.
2. Run M for i steps on each input, $s_1, s_2, \ldots, s_i$.
3. If M has accepted any of these, *accept*. Otherwise, continue."

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

Unrecognizable Languages



Check-in Quiz 10/27

On gradescope