UMB CS622
Mapping Reducibility & Unrecognizability
Wednesday, October 27, 2021
Announcements

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

• New required 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 \( ALL_{\text{CFG}} \) has a decider \( R \). Use it to create decider for \( A_{\text{TM}} \):

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

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

Decidable
Decidable
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 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”

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

This is like the TM-config-rejecting PDA used to prove \( \text{ALL}_{\text{CFG}} \) undecidable

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

This is similar to the \( \text{ww} \) language (not pumpable)

(This is also why union is closed for CFLs but intersection is not)
Last time: Algorithms For CFLs

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

(Still need to prove this is undecidable)
Theorem: $EQ_{CFG}$ is undecidable

$EQ_{CFG} = \{ (G, H) | G$ and $H$ 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" $\left\{ \frac{t_1}{b_1}, \frac{t_2}{b_2}, \ldots, \frac{t_k}{b_k} \right\}$
  - Where each $t_i$ and $b_i$ are strings

- E.g., $P = \left\{ \frac{b}{ca}, \frac{a}{ab}, \frac{ca}{a}, \frac{abc}{c} \right\}$

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

- E.g., $\frac{a}{ab} \frac{b}{ca} \frac{ca}{a} \frac{a}{ab} \frac{abc}{c} \rightarrow \begin{array}{cccccccc}
a & b & c & a & a & b & c \\
\downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\
a & b & c & a & a & a & b & c \\
\end{array}$

- Then: $PCP = \{ <P> | P \text{ is a set of dominos with a match} \}$
**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 $w$!
PCP Dominos

• First domino: \[
\begin{array}{c}
\#
\\
\#q_0w_1w_2 \cdots w_n\#
\end{array}
\]

• Key idea: add dominos representing valid TM steps:
  
  if \( \delta(q, a) = (r, b, R) \), put \( \frac{qa}{br} \) into \( P \)

  if \( \delta(q, a) = (r, b, L) \), put \( \frac{cq a}{rc b} \) into \( P \)

• For the tape cells that don’t change: put \( \frac{a}{-a} \) into \( P \)

• Top can only “catch up” if there is an accepting config sequence
PCP Example

• Let $w = 0100$ and $\delta(q_0, 0) = (q_7, 2, R)$ so $\begin{bmatrix} q_0 & 0 \\ 2q_7 \end{bmatrix}$ in $P$
PCP Dominos (accepting)

• When accept state reached, let top “catch” up:

For every \( a \in \Gamma \),

\[
\text{put } \frac{aq_{\text{accept}}}{q_{\text{accept}}} \text{ and } \frac{q_{\text{accept}}a}{q_{\text{accept}}} \text{ into } P
\]

Only possible match is accepting sequence of TM configs

Bottom “eats” one char

“eat” one char
Mapping Reducibility
Flashback: “Reduced”

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

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

Thm: \( HALT_{TM} \) is undecidable

Proof, by contradiction:

• Assume \( HALT_{TM} \) has decider \( R \); use to create \( A_{TM} \) decider:

\[ S = \text{“On input } \langle M, w \rangle \text{, an encoding of a TM } M \text{ and a string } w:\]

\[ 1. \text{ Run TM } R \text{ on input } \langle M, w \rangle. \]
\[ 2. \text{ If } R \text{ rejects, reject.} \]
\[ 3. \text{ If } R \text{ accepts, simulate } M \text{ on } w \text{ until it halts.} \]
\[ 4. \text{ If } M \text{ has accepted, accept; if } M \text{ has rejected, reject.”} \]

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: \( A_{\text{NFA}} \) 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 \( \text{NFA} \rightarrow \text{DFA} \)
2. Run TM \( M \) on input \( \langle C, w \rangle \).
3. If \( M \) accepts, accept; otherwise, reject.”

More generally, we’ve been saying “programs = TMs”, but programs do more than accept/reject?

We said this \( \text{NFA} \rightarrow \text{DFA} \) algorithm is a TM, but it doesn’t accept/reject?
Computable Functions

• A TM that, instead of accept/reject, “outputs” final tape contents

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.

• **Example 1:** All arithmetic operations

• **Example 2:** Converting between machines, like DFA\( \rightarrow \)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** $f : \Sigma^* \rightarrow \Sigma^*$, 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.
Thm: $A_{TM}$ is mapping reducible to $HALT_{TM}$

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

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

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

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

Still need to show: $M$ accepts $w$ if and only if $M'$ halts on $w$

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

Output new $M'$

Conver in $M$ to $M'$

Language $A$ is mapping reducible to language $B$, written $A \leq_m B$, if there is a computable function $f : \Sigma^* \rightarrow \Sigma^*$ such that 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.
⇒ 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 $\langle M', w \rangle$."

49
Use Mapping Reducibility to Prove ...

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

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.”

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 $w$,

$$w \in A \iff f(w) \in B.$$
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
Summary: Mapping Reducibility Theorems

- 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.

Be careful with the direction of the reduction!
Alternate Proof: The Halting Problem

$HALT_{TM}$ is undecidable

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

2. $A_{TM} \leq_m HALT_{TM}$

3. Since $A_{TM}$ is undecidable, then $HALT_{TM}$ is undecidable
Flashback: $EQ_{TM}$ is undecidable

Proof by contradiction:
• Assume $EQ_{TM}$ has decider $R$; use to create $ETM$ decider:

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

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

$$S = \text{“On input } \langle M \rangle, \text{ where } M \text{ is a TM:
1. Run } R \text{ on input } \langle M, M_1 \rangle, \text{ where } M_1 \text{ is a TM that rejects all inputs.
2. If } R \text{ accepts, accept; if } R \text{ rejects, reject.”}$$

Alternate proof: Show: $ETM \leq_m EQ_{TM}$
• Computable fn $f: \langle M \rangle \rightarrow \langle M, M_1 \rangle$

Last step: show iff requirements of mapping reducibility (exercise)
Reducing to complement: \( E_{\text{TM}} \) is undecidable

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

Proof, by contradiction:

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

\[
S = "\text{On input } \langle M, w \rangle, \text{ an encoding of a TM } M \text{ and a string } w:\n1. \text{ Use the description of } M \text{ and } w \text{ to construct the TM } M_1\n\]
\[
M_1 = "\text{On input } x:\n1. \text{ If } x \neq w, \text{ reject.}\n2. \text{ If } x = w, \text{ run } M \text{ on input } w \text{ and accept if } M \text{ does.}"\n\]

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

Alternate proof: computable fn: \( \langle M, w \rangle \rightarrow \langle M_1 \rangle \)

- So this only reduces \( A_{\text{TM}} \) to \( E_{\text{TM}} \)
- It’s good enough! Still proves \( E_{\text{TM}} \) 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?

\[ A_{TM} \]

Turing-recognizable

decidable

context-free

regular

Where do these go?

\[ E_{TM} = \{ \{M\} \mid M \text{ is a TM and } L(M) = \emptyset \} \]

\[ EQ_{CFG} = \{ \{G, H\} \mid G \text{ and } H \text{ are CFGs and } L(G) = L(H) \} \]

\[ EQ_{TM} = \{ \{M_1, M_2\} \mid M_1 \text{ and } M_2 \text{ are TMs and } L(M_1) = L(M_2) \} \]
Theorem: Some languages 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

\[ \Sigma^* = \{ \epsilon, \ 0, \ 1, \ 00, \ 01, \ 10, \ 11, \ 000, \ 001, \ \ldots \} \]

\[ A = \{ 0, \ 00, \ 01, \ 000, \ 001, \ \ldots \} \]

\[ \chi_A = 0 \ 1 \ 0 \ 1 \ 1 \ 0 \ 0 \ 1 \ 1 \ \ldots \]

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 $\iff$ Recognizable & co-Recognizable
Thm: Decidable $\Leftrightarrow$ Recognizable & co-Recognizable

$\Rightarrow$ If a language is **decidable**, then it is **recognizable** and **co-recognizable**
- Decidable $\Rightarrow$ Recognizable (hw5):
  - A decider is just a recognizer that halts
- Decidable $\Rightarrow$ 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

$\Leftarrow$ If a language is **recognizable** and **co-recognizable**, then it is **decidable**
**Thm:** Decidable $\Leftrightarrow$ Recognizable & co-Recognizable

$\Rightarrow$ If a language is **decidable**, then it is **recognizable** and **co-recognizable**

- Decidable $\Rightarrow$ Recognizable (hw5):
  - A decider is just a recognizer that halts
- Decidable $\Rightarrow$ 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

$\Leftarrow$ If a language is **recognizable** and **co-recognizable**, then it is **decidable**

- Let $M_1 = \text{recognizer for the language},$
- and $M_2 = \text{recognizer for its complement}$
- Decider $M$:
  - Run 1 step on $M_1$, Termination Arg: Either $M_1$ or $M_2$ must accept and halt, so $M$ halts and is a decider
  - Run 1 step on $M_2$,
  - Repeat, until one machine accepts. If it’s $M_1$, accept. If it’s $M_2$, reject
A Turing-unrecognizable language

Recognizable & co-recognizable implies decidable

• We’ve proved:

\[ A_{TM} \text{ is Turing-recognizable} \]

\[ A_{TM} \text{ is undecidable} \]

• So:

\[ \overline{A_{TM}} \text{ is not Turing-recognizable} \]
Is there anything out here?

$A_{TM}$

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

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

1. $EQ_{TM}$ is not Turing-recognizable

---

$A_{TM}$

---

$A_{TM} \leq_m EQ_{TM}$

$A_{TM}$ is not Turing-recognizable, thus $EQ_{TM}$ is 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 $f : \Sigma^* \rightarrow \Sigma^*$, where for every $w$,

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

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

Diagram:

$$A \leq_m B$$

implies

$$\overline{A} \leq_m \overline{B}$$
**Thm:** $EQ_{TM}$ is neither Turing-recognizable nor co-Turing-recognizable

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

1. $EQ_{TM}$ is not Turing-recognizable
   - Two Choices:
     - Create Computable fn: $A_{TM} \rightarrow EQ_{TM}$
     - Or Computable fn: $A_{TM} \rightarrow \overline{EQ_{TM}}$
**Thm:** $EQ_{TM}$ is not Turing-recognizable

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

- Create Computable fn: $A_{TM} \rightarrow \overline{EQ_{TM}}$
- $\langle M, w \rangle \rightarrow \langle M_1, M_2 \rangle$ $M_1$ 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: $\leftarrow$ Accepts nothing
     1. Reject.”

   - $M_2 =$ “On any input: $\leftarrow$ 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

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

1. \(EQ_{TM}\) is not Turing-recognizable
   
   • Create Computable fn: \(A_{TM} \rightarrow EQ_{TM}\)
   
   • Or Computable fn: \(A_{TM} \rightarrow \overline{EQ_{TM}}\)

   • DONE!

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

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

• Create Computable fn: $A_{TM} \rightarrow \overline{E_{TM}}$

• $\langle M, w \rangle \rightarrow \langle M_1, M_2 \rangle$ $M_1$ 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.”

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

DONE!
Now: $\overline{EQ_{TM}}$ is not Turing-recognizable

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

$$F = \text{“On input } \langle M, w \rangle, \text{ where } M \text{ is a TM and } w \text{ a string:}
\begin{align*}
1. \text{ Construct the following two machines, } M_1 \text{ and } M_2. \\
M_1 &= \text{“On any input:} \\
1. \text{ Accept.”} \\
M_2 &= \text{“On any input:} \\
1. \text{ Run } M \text{ on } w. \text{ If it accepts, accept.”} \\
2. \text{ Output } \langle M_1, M_2 \rangle."
\end{align*}
$$

- 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?

$A_{TM}$

Where do these go?

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

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

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

The figure illustrates the hierarchy of language classes: regular, context-free, decidable, Turing-recognizable, and $A_{TM}$. The complement of $A_{TM}$, $\overline{A_{TM}}$, is also shown. The set $E_{TM}$ is defined as $\{ \{M\} \mid M \text{ is a TM and } L(M) = \emptyset \}$ and $E_{Q_{TM}}$ as $\{ (G, H) \mid G \text{ and } H \text{ are CFGs and } L(G) = L(H) \}$. Where do these go?
**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:** \( \text{EQ}_{\text{CFG}} \) is co-Turing-recognizable

\[ \text{EQ}_{\text{CFG}} = \{ (G, H) | G \text{ and } H \text{ are CFGs and } L(G) = L(H) \} \]

Recognizer for \( \overline{\text{EQ}_{\text{CFG}}} \):

- 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

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

Where do these go?

\[ E_{TM} = \{ \{M\} | M \text{ is a TM and } L(M) = \emptyset \} \]

\[ EQ_{CFG} = \{ \{G, H\} | G \text{ and } H \text{ are CFGs and } L(G) = L(H) \} \]
Unrecognizable Languages

Where do these go?

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

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

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

```
“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