CS622

More Undecidability

Monday, October 25, 2021

```
DEFINE DOESITHALT(PROGRAM):
{
    RETURN TRUE;
}
```

*THE BIG PICTURE SOLUTION TO THE HALTING PROBLEM*
Announcements

• Hw5 in

• Hw6 out
  • Due Sunday 10/24 11:59pm EST

• Hw4 grades returned
Last Time: The Limits of Algorithms

- $A_{DFA} = \{ \langle B, w \rangle | B \text{ is a DFA that accepts input string } w \}$ Decidable

- $E_{DFA} = \{ \langle A \rangle | A \text{ is a DFA and } L(A) = \emptyset \}$ Decidable

- $EQ_{DFA} = \{ \langle A, B \rangle | A \text{ and } B \text{ are DFAs and } L(A) = L(B) \}$ Decidable
Last Time: The Limits of Algorithms

- $A_{DFA} = \{ \langle B, w \rangle | B$ is a DFA that accepts input string $w \}$ Decidable
- $A_{CFG} = \{ \langle G, w \rangle | G$ is a CFG that generates string $w \}$ Decidable

- $E_{DFA} = \{ \langle A \rangle | A$ is a DFA and $L(A) = \emptyset \}$ Decidable
- $E_{CFG} = \{ \langle G \rangle | G$ is a CFG and $L(G) = \emptyset \}$ Decidable

- $EQ_{DFA} = \{ \langle A, B \rangle | A$ and $B$ are DFAs and $L(A) = L(B) \}$ Decidable
- $EQ_{CFG} = \{ \langle G, H \rangle | G$ and $H$ are CFGs and $L(G) = L(H) \}$ Undecidable
Last Time: The Limits of Algorithms

- \( A_{\text{DFA}} = \{ \langle B, w \rangle \mid B \text{ is a DFA that accepts input string } w \} \)\hspace{1cm} \text{Decidable}
- \( A_{\text{CFG}} = \{ \langle G, w \rangle \mid G \text{ is a CFG that generates string } w \} \)\hspace{1cm} \text{Decidable}
- \( A_{\text{TM}} = \{ \langle M, w \rangle \mid M \text{ is a TM and } M \text{ accepts } w \} \)\hspace{1cm} \text{Undecidable}
- \( E_{\text{DFA}} = \{ \langle A \rangle \mid A \text{ is a DFA and } L(A) = \emptyset \} \)\hspace{1cm} \text{Decidable}
- \( E_{\text{CFG}} = \{ \langle G \rangle \mid G \text{ is a CFG and } L(G) = \emptyset \} \)\hspace{1cm} \text{Decidable}
- \( E_{\text{TM}} = \{ \langle M \rangle \mid M \text{ is a TM and } L(M) = \emptyset \} \)\hspace{1cm} \text{Undecidable}
- \( EQ_{\text{DFA}} = \{ \langle A, B \rangle \mid A \text{ and } B \text{ are DFAs and } L(A) = L(B) \} \)\hspace{1cm} \text{Decidable}
- \( EQ_{\text{CFG}} = \{ \langle G, H \rangle \mid G \text{ and } H \text{ are CFGs and } L(G) = L(H) \} \)\hspace{1cm} \text{Undecidable}
- \( EQ_{\text{TM}} = \{ \langle M_1, M_2 \rangle \mid M_1 \text{ and } M_2 \text{ are TMs and } L(M_1) = L(M_2) \} \)\hspace{1cm} \text{Undecidable}

TBD
No Algorithms About Language of TMs

- $\text{REGULAR}_\text{TM} = \{ <M> \mid M \text{ is a TM and } L(M) \text{ is a regular language} \}$

- $\text{CONTEXTFREE}_\text{TM} = \{ <M> \mid M \text{ is a TM and } L(M) \text{ is a CFL} \}$

- $\text{DECIDABLE}_\text{TM} = \{ <M> \mid M \text{ is a TM and } L(M) \text{ is a decidable language} \}$

- $\text{FINITE}_\text{TM} = \{ <M> \mid M \text{ is a TM and } L(M) \text{ is a finite language} \}$
Rice’s Theorem: $ANYTHING_{TM}^{TM}$ is Undecidable

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

- “Anything”, more precisely:
  - For any $M_1, M_2$, if $L(M_1) = L(M_2)$ ...
  - ... then $M_1 \in ANYTHING_{TM}^{TM} \iff M_2 \in ANYTHING_{TM}^{TM}$

- Also, anything must be “non-trivial”:
  - $ANYTHING_{TM}^{TM} \neq \{\}$
  - $ANYTHING_{TM}^{TM} \neq \text{ set of all TMs}$
Rice’s Theorem: \( \text{ANYTHING}_{\text{TM}} \) is Undecidable

\( \text{ANYTHING}_{\text{TM}} = \{<M> \mid M \text{ is a TM and … anything … about } L(M)\} \)

Proof by contradiction

- Assume some lang satisfying \( \text{ANYTHING}_{\text{TM}} \) has a decider \( R \).
  - Since \( \text{ANYTHING}_{\text{TM}} \) is non-trivial, then there exists \( M_{\text{ANY}} \in \text{ANYTHING}_{\text{TM}} \)
  - Where \( R \) accepts \( M_{\text{ANY}} \)

- Use \( R \) to create decider for \( A_{\text{TM}} \):
  
  **On input \(<M, w>\):**

  - Create \( M_w \):
    
    \[
    M_w = \text{on input } x:
    \]
    
    - Run \( M \) on \( w \)
      
      - If \( M \) rejects \( w \): reject \( x \)
      - If \( M \) accepts \( w \):
        
        Run \( M_{\text{ANY}} \) 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

  - Run \( R \) on \( M_w \)
    
    - If it accepts, then \( M_w = M_{\text{ANY}} \), so \( M \) accepts \( w \), so accepts
    
    Else reject

Wait! What if the TM that accepts nothing is in \( \text{ANYTHING}_{\text{TM}} \)!

Proof still works! Just use the complement of \( \text{ANYTHING}_{\text{TM}} \) instead!
(see hw5: complement closed for decidable languages)
Rice’s Theorem Real-World Example

```c
main()
{
    printf("hello, world\n");
}
```

Write a program that, given another program as its argument, returns TRUE if the argument prints “Hello, World!”

TRUE
Rice’s Theorem Example

Write a program that, given another program as its argument, returns TRUE if the argument prints “Hello, World!”

main()
{
    If $x^n + y^n = z^n$, for any integer $n > 2$
    printf("hello, world\n");
}

Fermat’s Last Theorem
\{<M> \mid M \text{ is a TM that installs malware}\}

Undecidable!
(by Rice’s Theorem)

RANSOMWARE ATTACK

Your files have been encrypted

function check(n)
    // check if the number n is a prime
    var factor; // if the checked number is not a prime, this is its first factor
    var i = 2;
    // try to divide the checked number by all numbers till its square root
    for (i = 2 ; i <= Math.sqrt(n) ; i++)
    {
        if (n % i === 0) // is n divisible by i ?
        {
            factor = i; // break
        }
    }
    return factor;
    // end of check function

function communicate()
    // communicate with the user
    var i = 0; // checked number
    var factor; // if the checked number is not a prime, this is its first factor
    i = document.getElementById("inputValue").value; // get the checked number
    // is it a valid input?
    if (!isNaN(i)) || i >= 1 || Math.floor(i) == i)
    {
        alert("The checked number should be a positive number");
    } else
    {
        factor = check(i);
        if (factor == 0)
        {
            alert("1 + " + i + " is a prime");
        } else
        {
            alert("1 + "+ i + " is not a prime, " + i + " = " + factor + " x " + i/factor");
        }
    }
    // end of communicate function
\[ A_{\text{DFA}} = \{ \langle B, w \rangle \mid B \text{ is a DFA that accepts input string } w \} \]
\[ A_{\text{CFG}} = \{ \langle G, w \rangle \mid G \text{ is a CFG that generates string } w \} \]
\[ A_{\text{TM}} = \{ \langle M, w \rangle \mid M \text{ is a TM and } M \text{ accepts } w \} \]

- Decidable
- Decidable
- Undecidable

- In hindsight, of course a restricted TM (a decider) shouldn’t be able to simulate unrestricted TM (a recognizer)
- But could a restricted TM simulate an even more restricted TM?
A linear bounded automaton is a restricted type of Turing machine wherein the tape head isn’t permitted to move off the portion of the tape containing the input. If the machine tries to move its head off either end of the input, the head stays where it is—in the same way that the head will not move off the left-hand end of an ordinary Turing machine’s tape.
Context-Sensitive Languages

context-sensitive languages, recognized by linear bounded automata

What exactly does it mean to be **context-free** vs **context-sensitive**?

Chomsky Hierarchy
Theorem: $A_{LBA}$ is decidable

$A_{LBA} = \{(M, w) | M \text{ is an LBA that accepts string } w\}$
Flashback: TM Configuration = State + Head + Tape

10110111110

Textual representation of “configuration”

1st char after state is current head position
How Many Possible Configurations ...

• Does an LBA have?
  • \( q \) states
  • \( g \) tape alphabet chars
  • tape of length \( n \)

• Possible Configurations = \( qng^n \)
  • \( g^n \) = number of possible tape configurations
  • \( qn \) = all the possible head positions
Theorem: $A_{LBA}$ is decidable

\[ A_{LBA} = \{ \langle M, w \rangle \mid M \text{ is an LBA that accepts string } w \} \]

Proof: Create decider for $A_{LBA}$

On input $\langle M, w \rangle$:

- Simulate $M$ on $w$.
- If $M$ accepts $w$, then accept.
- If $M$ runs > $qng^n$ steps then we are in a loop so halt and reject

Termination argument?
Theorem: $E_{\text{LBA}}$ is undecidable

$E_{\text{LBA}} = \{ \langle M \rangle | M \text{ is an LBA where } L(M) = \emptyset \}$
Flashback: TM Configuration Sequences

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

**Single-step**

(Right)

\[ \alpha q_1 a \beta \vdash \alpha x q_2 \beta \]

- if \( q_1, q_2 \in Q \)
- \( \delta(q_1, a) = (q_2, x, R) \)
- \( a, x \in \Gamma \)
- \( \alpha, \beta \in \Gamma^* \)

(Left)

\[ \alpha b q_1 a \beta \vdash \alpha q_2 b x \beta \]

- if \( \delta(q_1, a) = (q_2, x, L) \)

**Extended**

- Base Case
  \[ I \vdash^* I \text{ for any ID } I \]

- Recursive Case
  \[ I \vdash^* J \text{ if there exists some ID } K \text{ such that } I \vdash K \text{ and } K \vdash^* J \]
Theorem: $E_{\text{LBA}}$ is undecidable

$E_{\text{LBA}} = \{ \langle M \rangle \mid M \text{ is an LBA where } L(M) = \emptyset \}$

Proof, by contradiction:

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

- On input $\langle M, w \rangle$, where $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$:
  - Construct LBA $B$:
    - $B$ accepts sequences of $M$ configurations where $M$ accepts $w$, i.e.,
      - First configuration is $q_0 w_1 w_2 \cdots w_n$
      - Last configuration has state $q_{\text{accept}}$
      - Each pair of adjacent configs is valid according to $M$’s $\delta$

- Run $R$ with $B$ as input:
  - If $R$ accepts $B$, then $B$’s language is empty
    - So there’s no sequence of $M$ configs that accept $w$, so reject
  - If $R$ rejects $B$, then $B$’s language is not empty
    - So there’s a sequence of $M$ configs that accepts $w$, so accept

Wait! So any language that can be used to check computation histories must be undecidable
Theorem: $ALL_{CFG}$ is undecidable

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

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 $G$ (previous 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
A PDA That Rejects TM $M$ Config Sequences

On input $\# C_1 \# C_2 \# C_3 \# \cdots \# C_i \#$, nondeterministically:

- Reject if $C_1$ is not $q_0 w_1 w_2 \cdots w_n$
- Reject if $C_i$ does not have $q_{\text{accept}}$
- Reject if any $C_i$ and $C_{i+1}$ is invalid according to $\delta$:
  - Push $C_i$ onto the stack
  - Compare $C_i$ with $C_{i+1}$ (reversed):
    - Check that initial chars match
    - On first non-matching char, check that next 3 chars is valid according to $\delta$
      - Each possible $\delta$ can be hard-coded since $\delta$ is finite
      - Continue checking remaining chars
    - Reject whenever anything is invalid

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

Why reject accepting configuration sequences?

Could we create a PDA that accepts accepting configuration sequences?

But that would mean $E_{\text{CFG}}$ is undecidable??

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

We already proved this is decidable!
Algorithms For CFLs

- \( A_{CFG} = \{ \langle G, w \rangle \mid G \text{ is a CFG that generates string } w \} \)
  - Decidable

- \( E_{CFG} = \{ \langle G \rangle \mid G \text{ is a CFG and } L(G) = \emptyset \} \)
  - Already proved this is decidable

- \( ALL_{CFG} = \{ \langle G \rangle \mid G \text{ is a CFG and } L(G) = \Sigma^* \} \)
  - Just proved this is undecidable

- Undecidable
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 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 also why union is closed for CFLs but intersection is not)
Algorithms For CFLs

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

- $E_{CFG} = \{ \langle G \rangle | G \text{ is a CFG and } L(G) = \emptyset \}$  
  Decidable

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

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

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

$EQ_{CFG} = \{\langle G, H \rangle \mid G$ 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 $\langle G \rangle$:
|---|
| • Construct a CFG $G_{ALL}$ which generates all possible strings
| • Run $R$ with $G$ and $G_{ALL}$
| • Accept $G$ if $R$ accepts, else reject |
Turing Unrecognizable?

Is there anything out here?

Where do these go?

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

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

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

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

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

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

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

$\Rightarrow$ If a language is **decidable**, then it is **recognizable** and **co-recognizable**
- Decidable $\Rightarrow$ Recognizable (hw5):
  - A decider is a recognizer, bc decidable langs are a subset of recognizable langs
- 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 ⇔ Recognizable & co-Recognizable**

⇒ If a language is **decidable**, then it is **recognizable** and **co-recognizable**
  - Decidable ⇒ Recognizable:
    - A decider is a recognizer, bc decidable langs are a subset of recognizable langs
  - 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

• We’ve proved:

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

• So:

\[ \overline{A_{TM}} \text{ is not Turing-recognizable} \]

• Because: recognizable & co-recognizable implies decidable
Is there anything out here?

$A_{TM}$

Turing-recognizable

decidable

context-free

regular
Mapping Reducibility
Last time: “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. 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.”

**Problem:** What if it takes forever to create this decider?

- **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 $A_{\text{NFA}}$:

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

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.”

We said this $\text{NFA} \rightarrow \text{DFA}$ 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 \( f : \Sigma^* \rightarrow \Sigma^* \) 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.
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 $f : \langle M, w \rangle \to \langle M', w' \rangle$ 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 $f$.

$F =$ “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 $\langle M', w \rangle$.”

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 $M'$

Converts $M$ to $M'$

Language $A$ is mapping reducible to language $B$, written $A \leq_m B$, if there is a computable function $f : \Sigma^* \to \Sigma^*$, 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 $f : \Sigma^* \to \Sigma^*$ 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 $(M, w)$:
  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 $(M', w)$.”
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.

$$N = \text{"On input } w:\"$$

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.
Alternate Proof: The Halting Problem

\( \text{HALT}_{\text{TM}} \text{ is undecidable} \)

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

- \( A_{\text{TM}} \leq_m \text{HALT}_{\text{TM}} \)

- Since \( A_{\text{TM}} \) is undecidable, then \( \text{HALT}_{\text{TM}} \) is undecidable
Flashback: $E_{Q_{TM}}$ is undecidable

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

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

\[ E_{Q_{TM}} = \{ \langle M_1, M_2 \rangle \mid M_1 \text{ and } M_2 \text{ are TMs and } L(M_1) = L(M_2) \} \]

$S =$ “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: $E_{TM} \leq_m E_{Q_{TM}}$

• Computable fn $f : \langle M \rangle \rightarrow \langle M, M_1 \rangle$

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

Proof, by contradiction:

• Assume $E_{TM}$ 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. Use the description of $M$ and $w$ to construct the TM $M_1$.

\[ M_1 = \text{"On input } x:\]

   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.”

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

• So this only reduces $A_{TM}$ to $E_{TM}$.

• 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!
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

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

1. \( EQ_{TM} \) is not Turing-recognizable

\[ \overline{A_{TM}} \]

\[
\overline{A_{TM}} \leq_m EQ_{TM} \quad A \text{ is not Turing-recognizable, th} \quad EQ_{TM} \text{ not Turing-recognizable.}
\]
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 showing reduction from $A$ to $B$.]
Thm: $EQ_{TM}$ is neither Turing-recognizable nor co-Turing-recognizable

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

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:
     1. Reject.”
   - $M_2 =$ “On any input:
     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:** $E_{TM}$ is neither Turing-recognizable nor co-Turing-recognizable

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

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

   - Create Computable fn: \( A_{TM} \rightarrow E_{TM} \)
   - Or Computable fn: \( A_{TM} \rightarrow \overline{E_{TM}} \)
   - **DONE!**

2. \( \overline{E_{TM}} \) 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

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

• Create Computable fn: \[ A_{TM} \rightarrow \overline{EQ}_{TM} \]

• \[ \langle M, w \rangle \rightarrow \langle M_1, M_2 \rangle \]
  \( M_1 \text{ and } M_2 \text{ are TMs and } L(M_1) \neq L(M_2) \)

\[ F = "\text{On input } \langle M, w \rangle, \text{ where } M \text{ is a TM and } w \text{ a string:}\]

1. Construct the following two machines, \( M_1 \) and \( M_2 \).
   \( M_1 = "\text{On any input:} \)
   \[ 1. \text{ Reject.} " \]
   \( M_2 = "\text{On any input:} \)
   \[ 1. \text{ Run } M \text{ on } w. \text{ If it accepts, accept.}" \]

2. Output \( \langle M_1, M_2 \rangle. " \)
Now: $\overline{\text{EQ}_{TM}}$ is not Turing-recognizable

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

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

- $\langle M, w \rangle \rightarrow \langle M_1, M_2 \rangle$ $M_1$ 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:
     1. Accept.”
   - $M_2 =$ “On any input:
     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 $M_2$
Unrecognizable Languages?

\[ A_{TM} \supseteq \text{Turing-recognizable} \supseteq \text{decidable} \supseteq \text{context-free} \supseteq \text{regular} \]

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

$A_{\text{TM}}$ - Turing-recognizable

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

$EQ_{\text{TM}}$

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

Where do these go?
Check-in Quiz 10/25

On gradescope