Today's learning goals

- Define and explain core examples of computational problems, include $A^{**}$, $E^{**}$, $EQ^{**}$, $HALT_{TM}$ (for ** either DFA or TM)
- Explain what it means for one problem to reduce to another
- Define computable functions, and use them to give mapping reductions between computational problems.
<table>
<thead>
<tr>
<th>Decidable</th>
<th>Undecidable but recognizable</th>
<th>Undecidable and unrecognizable</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A_{DFA}$</td>
<td>$A_{TM}$</td>
<td>$A_{TM}^c$</td>
</tr>
<tr>
<td>$E_{DFA}$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$EQ_{DFA}$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Give algorithm!  
Diagonalization
Idea

If problem X is no harder than problem Y
   …and if Y is **decidable**
   …then X must also be **decidable**

If problem X is no harder than problem Y
   …and if X is **undecidable**
   …then Y must also be **undecidable**

“Problem X is no harder than problem Y” means
“Can convert questions about membership in X to questions about membership in Y”
Problem A is **mapping reducible** to problem B means there is a computable function $f: \Sigma^* \rightarrow \Sigma^*$ such that for all strings $x$ in $\Sigma^*$

\[
\text{x is in A iff } f(x) \text{ is in B}
\]

**Computable function?**
A function $f: \Sigma^* \rightarrow \Sigma^*$ is **computable** iff there is some Turing machine such that, for each $x$, on input $x$ halts with exactly $f(x)$ followed by all blanks on the tape
Which of the following functions are computable?

A. The string $x$ maps to the string $xx$.

B. The string $<M>$ (where $M$ is a TM) maps to $<M’>$ where $M’$ is the Turing machine that acts like $M$ does, except that if $M$ tries to reject, $M’$ goes into a loop; strings that are not the codes of TMs map to $\varepsilon$.

C. The string $x$ maps to $y$, where $x$ is the binary representation of the number $n$ and $y$ is the binary representation of the number $2^n$.

D. All of the above.

E. None of the above.
The halting problem!

\[ \text{HALT}_{\text{TM}} = \{ <M,w> | M \text{ is a TM and } M \text{ halts on input } w \} \]

\[ A_{\text{TM}} = \{ <M,w> | M \text{ is a TM and } w \text{ is in } L(M) \} \]

How is \( \text{HALT}_{\text{TM}} \) related to \( A_{\text{TM}} \)?

A. They're the same set.
B. \( \text{HALT}_{\text{TM}} \) is a subset of \( A_{\text{TM}} \)
C. \( A_{\text{TM}} \) is a subset of \( \text{HALT}_{\text{TM}} \)
D. They have the same type of elements but no other relation.
E. I don't know.
The halting problem!

\[ \text{HALT}_{\text{TM}} = \{ \langle M, w \rangle | M \text{ is a TM and } M \text{ halts on input } w \} \]

\[ A_{\text{TM}} = \{ \langle M, w \rangle | M \text{ is a TM and } w \text{ is in } L(M) \} \]

But subset inclusion doesn't determine difficulty!
The halting problem!

\[ \text{HALT}_\text{TM} = \{ <M,w> \mid \text{M is a TM and M halts on input } w \} \]

\[ \text{A}_{\text{TM}} = \{ <M,w> \mid \text{M is a TM and } w \text{ is in } L(M) \} \]

**Goal:** build function that \( f : \Sigma^* \rightarrow \Sigma^* \) such that for every string \( x \),

\[ x \text{ is in } \text{A}_{\text{TM}} \iff f(x) \text{ is in } \text{HALT}_{\text{TM}} \]
Reducing $A_{TM}$ to $HALT_{TM}$  

Desired function by cases:

- If $x = <M,w>$ and $w$ is in $L(M)$: map to $<M', w'>$ in $HALT_{TM}$

- If $x = <M,w>$ and $w$ is not in $L(M)$: map to $<M', w'>$ not in $HALT_{TM}$

- If $x \neq <M,w>$: map to some string not in $HALT_{TM}$
Reducing $A_{TM}$ to $HALT_{TM}$  

Sipser Example 5.24

Desired function by cases:

- If $x = <M,w>$ and $w$ is in $L(M)$: map to $<M', w'>$ in $HALT_{TM}$

- If $x = <M,w>$ and $w$ is not in $L(M)$: map to $<M', w'>$ not in $HALT_{TM}$

- If $x \neq <M,w>$: map to some string not in $HALT_{TM}$
  Pick some specific string constant not in $HALT_{TM}$
Reducing $A_{TM}$ to $HALT_{TM}$

Sipser Example 5.24

Define **computable** function:

$F =$ “On input $x$:
1. Type-check whether $x = <M,w>$ for some TM $M$, and string $w$. If not, output $const_{out}$.
2. ...
3. ...
4. .... ”

$F$ is defined by high-level description of TM: each step must be algorithmic!
Reducing $A_{\text{TM}}$ to $\text{HALT}_{\text{TM}}$  

Sipser Example 5.24

Define \textit{computable} function:

$F =$ “On input $x$:
1. Type-check whether $x = \langle M, w \rangle$ for some TM $M$, and string $w$. If not, output $\text{const}_{\text{out}}$.
2. Simulate $M$ on $w$.
3. If accepts, accept. If rejects, reject.
4. …. ”

$F$ is defined by high-level description of TM: each step must be algorithmic!
Reducing $A_{TM}$ to $HALT_{TM}$  

Sipser Example 5.24

Define *computable* function:

F = “On input x:
1. Type-check whether $x = <M,w>$ for some TM $M$, and string $w$. If not, output $\text{const}_{\text{out}}$.
2. 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.”
3. Output $<M', w>$”

F is defined by high-level description of TM: each step must be algorithmic!
Reducing $A_{TM}$ to $HALT_{TM}$  

Check how function behaves by cases:

- If $x = <M,w>$ and $w$ is in $L(M)$: map to $<M', w'>$ in $HALT_{TM}$?

- If $x = <M,w>$ and $w$ is not in $L(M)$: map to $<M', w'>$ not in $HALT_{TM}$?

- If $x \neq <M,w>$: map to some string not in $HALT_{TM}$?
Other direction?

Goal: build function that $f: \Sigma^* \rightarrow \Sigma^*$ such that for every string $x$,

$$x \text{ is in } \text{HALT}_{TM} \iff f(x) \text{ is in } \text{A}_{TM}$$

What function should be used for $f(x)$ in the reduction?
A. Use the function $F$ from previous reduction
B. Use the inverse of the function $F$ from previous reduction
C. Use a different computable function
D. Impossible to find a computable function that works!
Next time

Pre-class reading Example 5.24, Theorems 5.22