CS 154
Unrecognizability, Undecidability, Diagonalization
“There are more problems to solve than there are programs to solve them.”

Languages over \{0,1\}

Turing Machines
Let $L$ be any set and $2^L$ be the power set of $L$

**Theorem:** There is no onto function from $L$ to $2^L$

No function from $L$ to $2^L$ can “cover” all the elements in $2^L$

No matter what the set $L$ is, the power set $2^L$ always has strictly larger cardinality than $L$
Suppose every language is recognizable.

Then for every language $L'$ over $\{0,1\}$ there is a TM $M$ such that $L(M) = L'$.

This means that the function $f(M) = L(M)$ from $\{\text{Turing Machines}\}$ to $\{\text{Languages}\}$ is \textit{onto}:

For every $L'$ in $\{\text{Languages}\}$, there is an $M$ in $\{\text{Turing Machines}\}$ such that $f(M) = L'$
Thm: There are *unrecognizable* languages

Assuming every language is recog., there’s an onto function

\[ f: \{\text{Turing Machines}\} \rightarrow \{\text{Languages}\} \]

\{Turing Machines\} \hspace{1cm} \{\text{Languages over \{0,1\}}\}

\{0,1\}^* \hspace{1cm} \{\text{Sets of strings of 0s and 1s}\}

Set \( S \) \hspace{1cm} \text{Set of all subsets of M: } 2^S

Since \( f \) is onto, there is also an onto \( g \) from \( S \) to \( 2^S \).

But there is *no* onto function from \( S \) to \( 2^S \). Contradiction!

This is an extremely generic argument!
In the early 1900’s, logicians were trying to define consistent foundations for mathematics.  

Suppose $X =$ “Universe of all possible sets”  

Frege’s Axiom:  Let $f : X \rightarrow \{0,1\}$  
Then $\{S \in X \mid f(S) = 1\}$ is a set.  

Define $F = \{S \in X \mid S \notin S\}$  

Suppose $F \in F$. Then by definition, $F \notin F$.  
So $F \notin F$ and by definition $F \in F$.  

*This logical system is inconsistent!*
A Concrete Undecidable Problem: The Acceptance Problem for TMs

\[ A_{TM} = \{ (M, w) \mid M \text{ is a TM that accepts string } w \} \]

**Theorem [Turing’30s]**

\( A_{TM} \) is recognizable but **NOT** decidable
$A_{TM} = \{ (M,w) \mid M \text{ is a TM that accepts string } w \}$

$A_{TM}$ is undecidable: (proof by contradiction)

Suppose $H$ is a machine that decides $A_{TM}$

\[
H((M,w)) = \begin{cases} 
\text{Accept} & \text{if } M \text{ accepts } w \\
\text{Reject} & \text{if } M \text{ does not accept } w 
\end{cases}
\]

Define a new TM $D$ as follows:

$D(M)$: Run $H$ on $(M,M)$ and output the opposite of $H$.

\[
D(D) = \begin{cases} 
\text{Reject} & \text{if } D \text{ accepts } D \\
\text{Accept} & \text{if } D \text{ does not accept } D 
\end{cases}
\]

Set $M = D$?
The table of outputs of $H(x,y)$

<table>
<thead>
<tr>
<th>$x$</th>
<th>$y$</th>
<th>$w_1$</th>
<th>$w_2$</th>
<th>$w_3$</th>
<th>$w_4$</th>
<th>...</th>
<th>$D$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$M_1$</td>
<td>accept</td>
<td>accept</td>
<td>accept</td>
<td>reject</td>
<td>accept</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$M_2$</td>
<td>reject</td>
<td>accept</td>
<td>reject</td>
<td>reject</td>
<td>reject</td>
<td></td>
<td>reject</td>
</tr>
<tr>
<td>$M_3$</td>
<td>accept</td>
<td>reject</td>
<td>reject</td>
<td>accept</td>
<td>accept</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$M_4$</td>
<td>accept</td>
<td>reject</td>
<td>reject</td>
<td>reject</td>
<td>accept</td>
<td></td>
<td></td>
</tr>
<tr>
<td>:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$D$</td>
<td>reject</td>
<td>reject</td>
<td>accept</td>
<td>accept</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
The behavior of $D(x)$ is a *diagonal* on this table.

<table>
<thead>
<tr>
<th></th>
<th>$w_1$</th>
<th>$w_2$</th>
<th>$w_3$</th>
<th>$w_4$</th>
<th>...</th>
<th>$D$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$M_1$</td>
<td>reject</td>
<td>accept</td>
<td>accept</td>
<td>reject</td>
<td>accept</td>
<td></td>
</tr>
<tr>
<td>$M_2$</td>
<td>reject</td>
<td>reject</td>
<td>reject</td>
<td>reject</td>
<td>reject</td>
<td></td>
</tr>
<tr>
<td>$M_3$</td>
<td>accept</td>
<td>reject</td>
<td>accept</td>
<td>accept</td>
<td>accept</td>
<td></td>
</tr>
<tr>
<td>$M_4$</td>
<td>accept</td>
<td>reject</td>
<td>reject</td>
<td>accept</td>
<td>accept</td>
<td></td>
</tr>
<tr>
<td>$\vdots$</td>
<td>$\vdots$</td>
<td>$\vdots$</td>
<td>$\vdots$</td>
<td>$\vdots$</td>
<td>$\vdots$</td>
<td>$\vdots$</td>
</tr>
<tr>
<td>$D$</td>
<td>reject</td>
<td>reject</td>
<td>accept</td>
<td>accept</td>
<td>$?$</td>
<td></td>
</tr>
</tbody>
</table>

$D(x)$ outputs the *opposite* of $H(x,x)$

$D(D)$ outputs the *opposite* of $H(D,D)=D(D)$
$A_{TM} = \{ (M,w) \mid M \text{ is a TM that accepts string } w \}$

$A_{TM}$ is undecidable:  (a constructive proof)

Let $U$ be a machine that recognizes $A_{TM}$

$$U( (M,w) ) = \begin{cases} 
\text{Accept} & \text{if } M \text{ accepts } w \\
\text{Rejects or loops} & \text{otherwise}
\end{cases}$$

Define a new TM $D_U$ as follows:

$D_U(M)$: Run $U$ on $(M,M)$ until the simulation halts
Output the opposite answer
$D_U( D_U ) = \begin{cases} 
\text{Reject if } D_U \text{ accepts } D_U \\
\text{(i.e. if } H( D_U , D_U ) = \text{Accept}) \\
\text{Accept if } D_U \text{ rejects } D_U \\
\text{(i.e. if } H( D_U , D_U ) = \text{Reject}) \\
\text{Loops if } D_U \text{ loops on } D_U \\
\text{(i.e. if } H( D_U , D_U ) \text{ loops}) 
\end{cases}$

**Note:** There is no contradiction here!

$D_U$ must loop on $D_U$

We have an input $(D_U, D_U)$ which is not in $A_{TM}$ but $U$ infinitely loops on $(D_U, D_U)$!
In summary:

Given the code of any \textbf{machine U} that \textbf{recognizes} \(A_{TM}\) (i.e. a Universal Turing Machine) we can \textbf{effectively} construct an input \((D_U, D_U)\), where:

1. \((D_U, D_U)\) does not belong to \(A_{TM}\)

2. \textbf{U runs forever} on the input \((D_U, D_U)\)

3. So \textbf{U cannot decide} \(A_{TM}\)

Given any program that recognizes the \textbf{Acceptance Problem}, we can efficiently construct an input where the program hangs!
Theorem: $A_{TM}$ is recognizable but NOT decidable

Corollary: $\neg A_{TM}$ is not recognizable

Proof: Suppose $\neg A_{TM}$ is recognizable. Then $\neg A_{TM}$ and $A_{TM}$ are both recognizable. But that would mean they’re both decidable… … this is a contradiction!
The Halting Problem

\[ \text{HALT}_{\text{TM}} = \{ (M, w) \mid M \text{ is a TM that halts on string } w \} \]

**Theorem:** \( \text{HALT}_{\text{TM}} \) is undecidable

**Proof:** Assume (for a contradiction) there is a TM \( H \) that decides \( \text{HALT}_{\text{TM}} \)

Idea: Use \( H \) to construct a TM \( M' \) that *decides* \( A_{\text{TM}} \)

\( M'(M, w): \) Run \( H(M, w) \)

- If \( H \) rejects then *reject*
- If \( H \) accepts, run \( M \) on \( w \) until it halts:
  - If \( M \) accepts, then *accept*
  - If \( M \) rejects, then *reject*

**Claim:** If \( H \) exists, then \( M' \) decides \( A_{\text{TM}} \)
If \( M \) doesn't halt:

\[
\text{reject}
\]

If \( M \) halts:

\[
\text{H}
\]

Does \( M \) halt on \( w \)?

\[
(M, w)
\]
Can often prove a language $L$ is undecidable by proving: “if $L$ is decidable, then so is $A_{TM}$”

We **reduce** $A_{TM}$ to the language $L$

$$A_{TM} \leq L$$

$L$ is “at least as difficult as” $A_{TM}$
Reducing from One Problem to Another

\[ f : \Sigma^* \rightarrow \Sigma^* \text{ is a computable function if} \]
\[ \text{there is a Turing machine } M \text{ that halts with just } f(w) \text{ written on its tape, for every input } w \]

A language A is **mapping reducible** to language B, written as \( A \leq_m B \), if there is a computable \( f : \Sigma^* \rightarrow \Sigma^* \) such that for every \( w \),

\[ w \in A \iff f(w) \in B \]

\( f \) is called a mapping reduction (or many-one reduction) from A to B
Let $f : \Sigma^* \rightarrow \Sigma^*$ be a computable function such that $w \in A \iff f(w) \in B$.

Say: “$A$ is mapping reducible to $B$”

Write: $A \leq_m B$
Theorem: If \( A \leq_m B \) and \( B \leq_m C \), then \( A \leq_m C \)

\[ w \in A \iff f(w) \in B \iff g(f(w)) \in C \]
Theorem: If $A \leq_m B$ and $B$ is decidable, then $A$ is decidable

Proof: Suppose TM $M$ decides $B$. Let $f$ be a mapping reduction from $A$ to $B$. We build a machine $M'$ for deciding $A$

$M'(w)$:

1. Compute $f(w)$
2. Run $M$ on $f(w)$, output its answer

$w \in A \iff f(w) \in B$ so $w \in A \Rightarrow M'$ accepts $w$

$w \notin A \Rightarrow M'$ rejects $w$
Theorem: If $A \leq_m B$ and $B$ is recognizable, then $A$ is recognizable.

Proof: Let $M$ recognize $B$. Let $f$ be a mapping reduction from $A$ to $B$.

To recognize $A$, we build a machine $M'$:

$M'(w)$:
1. Compute $f(w)$
2. Run $M$ on $f(w)$, output its answer if you ever receive one.
Theorem: If $A \leq_m B$ and $B$ is decidable, then $A$ is decidable

Corollary: If $A \leq_m B$ and $A$ is undecidable, then $B$ is undecidable

Theorem: If $A \leq_m B$ and $B$ is recognizable, then $A$ is recognizable

Corollary: If $A \leq_m B$ and $A$ is unrecognizable, then $B$ is unrecognizable
A mapping reduction from $A_{TM}$ to $HALT_{TM}$

**Theorem:** $A_{TM} \leq_m HALT_{TM}$

$f(z) :=$ Decode $z$ into a pair $(M, w)$

Construct a TM $M'$ with the specification:

"$M'(w) = $ Simulate $M$ on $w$. If $M(w)$ accepts then *accept* else *loop forever*"

Output $(M', w)$

We have $z \in A_{TM} \iff (M', w) \in HALT_{TM}$

**Corollary:** $HALT_{TM}$ is undecidable
Theorem: $A_{TM} \leq_m \text{HALT}_{TM}$

Corollary: $\neg A_{TM} \leq_m \neg \text{HALT}_{TM}$

Proof?

Corollary: $\neg \text{HALT}_{TM}$ is unrecognizable!

Proof: If $\neg \text{HALT}_{TM}$ were recognizable, then $\neg A_{TM}$ would be recognizable...
Theorem: $\text{HALT}_{TM} \leq_m A_{TM}$

Proof: Define the computable function

$$f(M, w) := \text{Construct } M' \text{ with the specification:}$$

"$M'(w) = \text{If } M(w) \text{ halts then accept else loop forever}"$

Output $(M', w)$

Observe $(M, w) \in \text{HALT}_{TM} \iff (M', w) \in A_{TM}$