CS 154
Lecture 9: Diagonalization, Undecidability, Unrecognizability
“There are more problems to solve than there are programs to solve them.”

Languages over \{0,1\}

Turing Machines
f : A → B is onto ⇔ (∀ b ∈ B)(∃ a ∈ A)[f(a) = b]

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

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

Proof: Assume, for a contradiction, there is an onto function $f : L → 2^L$

Define $S = \{ x ∈ L \mid x ∉ f(x) \} ∈ 2^L$

If f is onto, then there is a $y ∈ L$ with $f(y) = S$
Suppose $y ∈ S$. By definition of S, $y ∉ f(y) = S$.
Suppose $y ∉ S$. By definition of S, $y ∈ f(y) = S$.

Contradiction!
Let \( L \) be any set and \( 2^L \) be the power set of \( L \).

**Theorem:** There is no onto function from \( L \) to \( 2^L \).

**Proof:** Let \( f : L \rightarrow 2^L \) be an arbitrary function.

Define \( S = \{ x \in L \mid x \notin f(x) \} \in 2^L \).

For all \( x \in L \),
- If \( x \in S \) then \( x \notin f(x) \) \([by \ definition \ of \ S]\)
- If \( x \notin S \) then \( x \in f(x) \)

In either case, we have \( f(x) \neq S \). (Why?)

Therefore \( f \) is not onto!
What does this mean?

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 \( L \)
Thm: There are *unrecognizable* languages

Proof: If all languages were recognizable, then for all L, there’d be a Turing machine M for recognizing L. Hence there is an onto R: \{Turing Machines\} → \{Languages\}

Therefore, there is *no* onto function from \{Turing Machines\} ⊆ M to \{Languages\}. Contradiction!
Russell’s Paradox in Set Theory

In the early 1900’s, logicians were trying to define consistent foundations for mathematics.

Suppose \( X = \text{“Universe of all possible sets”} \)

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!*
Theorem: There is no onto function from the positive integers $\mathbb{Z}^+$ to the real numbers in $(0, 1)$.

Proof: Suppose $f$ is such a function:

<table>
<thead>
<tr>
<th>$n$</th>
<th>$f(n)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$0.28347279...$</td>
</tr>
<tr>
<td>2</td>
<td>$0.88388384...$</td>
</tr>
<tr>
<td>3</td>
<td>$0.77635284...$</td>
</tr>
<tr>
<td>4</td>
<td>$0.11111111...$</td>
</tr>
<tr>
<td>5</td>
<td>$0.12345678...$</td>
</tr>
</tbody>
</table>

Define: $r \in (0, 1)$ such that:

\[
[n\text{-th digit of } r] = \begin{cases} 
1 & \text{if } [n\text{-th digit of } f(n)] \neq 1 \\
2 & \text{otherwise}
\end{cases}
\]

Then $f(n) \neq r$ for all $n$. (Here, $r = 0.11121...$)
Let $Z^+ = \{1, 2, 3, 4, \ldots\}$

There is a bijection between $Z^+$ and $Z^+ \times Z^+$
A Concrete Undecidable Problem: The Acceptance Problem for TMs

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

Theorem: \( A_{\text{TM}} \) is recognizable but NOT decidable

Corollary: \( \neg A_{\text{TM}} \) is not recognizable
\[ A_{TM} = \{ (M, w) \mid M \text{ is a TM that accepts string } w \} \]

\( A_{TM} \) 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 \( H \)

\[ D( D ) = \begin{cases} 
  \text{Reject} & \text{if } D \text{ accepts } D \\
  \text{Accept} & \text{if } D \text{ does not accept } D
\end{cases} \]
The table of outputs of $H(x,y)$

<table>
<thead>
<tr>
<th></th>
<th>$M_1$</th>
<th>$M_2$</th>
<th>$M_3$</th>
<th>$M_4$</th>
<th>$\ldots$</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>
</tr>
<tr>
<td>$M_2$</td>
<td>reject</td>
<td>accept</td>
<td>reject</td>
<td>reject</td>
<td>reject</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>
</tr>
<tr>
<td>$M_4$</td>
<td>accept</td>
<td>reject</td>
<td>reject</td>
<td>reject</td>
<td>accept</td>
<td></td>
</tr>
<tr>
<td>$D$</td>
<td>reject</td>
<td>reject</td>
<td>accept</td>
<td>accept</td>
<td>accept</td>
<td>?</td>
</tr>
</tbody>
</table>
The outputs of $D(x)$

<table>
<thead>
<tr>
<th></th>
<th>$M_1$</th>
<th>$M_2$</th>
<th>$M_3$</th>
<th>$M_4$</th>
<th>$\ldots$</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></td>
<td>reject</td>
<td>reject</td>
<td>accept</td>
<td>accept</td>
<td>accept</td>
<td></td>
</tr>
</tbody>
</table>

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

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

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

Let $H$ be a machine that recognizes $A_{\text{TM}}$

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

Define a new TM $D_H$ as follows:

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

Note: There is no contradiction here!

\[ D_H \text{ must loop on } D_H \]

We have an instance \((D_H, D_H)\) which is not in \(A_{TM}\) but \(H\) fails to tell us that!

\[ H(D_H, D_H) \text{ runs forever} \]
That is:

Given the code of any machine $H$ that recognizes $A_{TM}$ we can effectively construct an instance $(D_H, D_H)$, where:

1. $(D_H, D_H)$ does not belong to $A_{TM}$
2. $H$ runs forever on the input $(D_H, D_H)$

So $H$ cannot decide $A_{TM}$

Given any program that recognizes the 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!
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 \( \text{HALT}_{\text{TM}} \)

We use \( H \) to construct a TM \( M' \) that decides \( A_{\text{TM}} \)

\[ M'(M,w): \text{ Run } H(M,w) \]

If \( H \) rejects then \textit{reject}

If \( H \) accepts, run \( M \) on \( w \) until it halts:

\begin{align*}
\text{If } M \text{ accepts, then } & \textit{accept} \\
\text{If } M \text{ rejects, then } & \textit{reject}
\end{align*}
If $M$ doesn't halt:
reject

If $M$ halts:
Does $M$ halt on $w$?

$(M, w)$

$M'$

$H$

$M$

$w$
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 $L$

$A_{TM} \leq L$
Mapping Reductions

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

A language \( A \) is \textit{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 $A \leq_m C$
Theorem: If $A \leq_m B$ and $B$ is decidable, then $A$ is decidable

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

To decide $A$, we build a machine $M'$

$M'(w)$:
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
The proof that the Halting Problem is undecidable can be seen as constructing a mapping reduction from $A_{TM}$ to $HALT_{TM}$.

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

$f(M, w) :=$ Construct $M'$ with the specification

“$M'(w) = \text{if } M(w) \text{ accepts then accept}$

else loop forever”

Output $(M', w)$

We have $(M, w) \in A_{TM} \iff (M', w) \in HALT_{TM}$
Another way of writing the reduction $f$:

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

$f(z) := \text{Decode } z \text{ into a pair } (M, w)$

Construct $M'$ with the specification:

"$M'(w) = \text{Simulate } M \text{ on } w.$

if $M(w)$ accepts then accept

else loop forever"

Output $(M', w)$

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

Corollary: $\neg A_{TM} \leq_m \neg HALT_{TM}$

Proof?

Corollary: $\neg HALT_{TM}$ is unrecognizable!

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

Proof: Define the computable function:

$f(z) := \text{Decode } z \text{ into a pair } (M, w)$

Construct $M'$ with the specification:

"$M'(w) = \text{Simulate } M \text{ on } w.$

If $M(w)$ halts then accept

else loop forever"

Output $(M', w)$

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

Yo, T.M.! I can give you the magical power to either compute the halting problem, or the acceptance problem. Which do you want?

Wow, hm, so hard to choose...

I can’t decide!
The Emptiness Problem

\[ \text{EMPTY}_{\text{DFA}} = \{ M \mid \text{M is a DFA such that } L(M) = \emptyset \} \]

Given a DFA, does it reject every input?

Theorem: \( \text{EMPTY}_{\text{DFA}} \) is decidable

Why?

\[ \text{EMPTY}_{\text{NFA}} = \{ M \mid \text{M is a NFA such that } L(M) = \emptyset \} \]

\[ \text{EMPTY}_{\text{REX}} = \{ R \mid \text{M is a regexp such that } L(M) = \emptyset \} \]
The Emptiness Problem for TMs

\[ \text{EMPTY}^\text{TM} = \{ M \mid M \text{ is a TM such that } L(M) = \emptyset \} \]

Given a program, does it reject every input?

Theorem: \( \text{EMPTY}^\text{TM} \) is not recognizable

Proof: Show that \( \neg A^\text{TM} \leq_m \text{EMPTY}^\text{TM} \)

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

Output a TM \( M' \) with the behavior:
“\( M'(x) := \) if \( (x = w) \) then run \( M(w) \), else reject”

\[ z \in A^\text{TM} \iff L(M') \neq \emptyset \iff M' \notin \text{EMPTY}^\text{TM} \iff f(z) \notin \text{EMPTY}^\text{TM} \]