Lecture 10: Undecidability, Unrecognizability, and Reductions
Next Tuesday (3/19)

Your Midterm: 2:35-3:55pm, Room 3-270

No pset this week!

Just an optional (not graded) practice midterm

Solutions to practice midterm will come out during the weekend. All remaining HW solutions as well.

When you see the practice midterm...

DON’T PANIC!

Practice midterm will be harder than midterm
Your Midterm: 2:35-3:55pm, Room 3-270

No pset this week!
Just an optional (not graded) practice midterm

FAQ: What material is on the midterm?
Everything up to today (Lectures 1-10)
Thursday’s lecture may also help.
But we’ll focus more on earlier material

FAQ: Can I bring notes?
Yes, one single-sided sheet of notes, letter paper
Thm: There exist *unrecognizable* languages

Assuming the Church-Turing Thesis, this means there are problems that NO computing device can ever solve!

We prove there is no *onto* function from the set of all Turing Machines to the set of all languages over \{0,1\}. (But the proof will work for any finite \(\Sigma\))

Therefore, the function mapping every TM \(M\) to its language \(L(M)\), *fails to cover all possible languages*
“There are more problems to solve than there are programs to solve them.”

Languages over \( \{0,1\} \)

Turing Machines

\[ M \]

\[ L(M) \]
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 arbitrary

Define $S = \{ x \in L \mid x \not\in f(x) \} \in 2^L$

For all $x \in L$,

- If $x \in S$ then $x \not\in f(x)$ \[by \text{definition of } S\]
- If $x \not\in S$ then $x \in f(x)$

In either case, we have $f(x) \neq S$: the element $x$ is in *exactly one* of the sets!

Therefore $f$ is *not* onto!
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. Then we can make a list:

<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>

\[ \vdots \]

Define: \( r \in (0, 1) \)

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

\( f(n) \neq r \) for all \( n \) (Here, \( r = 0.11121... \) )

\( r \) is never output by \( f! \)
**Thm: There exist unrecognizable languages**

**Proof:** Suppose all languages are recognizable. Then for all $L$, there’s a TM $M$ that recognizes $L$.

Hence the function

$R: \{\text{Turing Machines}\} \rightarrow \{\text{Languages over \{0,1\}}\}$ defined by $M \mapsto L(M)$ is an onto function.

But we just showed there is *no* onto function from

$\{\text{Turing Machines}\} \subseteq \{0,1\}^*$ to the set of languages over \{0,1\}. *Contradiction!*
A Concrete Undecidable Problem: The Acceptance Problem for TMs

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

Given: code of a Turing machine M and an input w for that Turing machine, Decide: Does M accept w?

Theorem [Turing]: \( A_{TM} \) is recognizable but NOT decidable
\[ A_{TM} = \{ (M, w) \mid M \text{ is a TM that accepts string } w \} \]

**Thm.** \( A_{TM} \) is undecidable: (proof by contradiction)

Assume \( H \) 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 \( D \) with the following spec:

\[
D(M) : \text{Run } H \text{ on } (M, M) \text{ and output the opposite of } H
\]

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

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

<table>
<thead>
<tr>
<th></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>reject</td>
<td>accept</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>accept</td>
</tr>
<tr>
<td>$M_4$</td>
<td>accept</td>
<td>reject</td>
<td>reject</td>
<td>reject</td>
<td>reject</td>
<td>accept</td>
</tr>
<tr>
<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>accept</td>
<td>?</td>
</tr>
</tbody>
</table>

$M_1, M_2, \ldots$ and $w_1, w_2, \ldots$ are both ordered lists of all binary strings
### The table of outputs of $H(x,y)$

<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>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>accept</td>
</tr>
<tr>
<td>$M_4$</td>
<td>accept</td>
<td>reject</td>
<td>reject</td>
<td>reject</td>
<td>reject</td>
<td>accept</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>

$D(x)$ outputs the **opposite** of $H(x,x)$
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>reject</td>
</tr>
<tr>
<td>$M_3$</td>
<td>accept</td>
<td>reject</td>
<td>accept</td>
<td>accept</td>
<td>accept</td>
<td>accept</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>$D$</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 $H(D, D) = D(D)$
\[ \mathcal{A}_{TM} = \{ (M,w) \mid M \text{ is a TM that accepts string } w \} \]

Thm. \( \mathcal{A}_{TM} \) is undecidable. (a constructive proof)

Let \( U \) be a machine that recognizes \( \mathcal{A}_{TM} \)

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

Define a new TM \( D_U \) as follows:

\[
D_U(M): \text{Run } U \text{ on } (M,M) \text{ until the simulation halts} \\
\text{Output the opposite answer}
\]
\[ D_U( D_U ) = \]

- Reject if \( D_U \) accepts \( D_U \) (i.e. if \( H( D_U , D_U ) = \text{Accept} \))
- Accept if \( D_U \) rejects \( D_U \) (i.e. if \( H( D_U , D_U ) = \text{Reject} \))
- Loops if \( D_U \) loops on \( D_U \) (i.e. if \( H( D_U , D_U ) \) loops)

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

\( D_U \) must run forever 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 machine $U$ that recognizes $A_{\text{TM}}$ (i.e. a Universal Turing Machine) we can effectively construct an input $(D_U, D_U)$, where:

1. $(D_U, D_U) \notin A_{\text{TM}}$ ($D_U$ does not accept $D_U$)

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

Therefore $U$ cannot decide $A_{\text{TM}}$

Given any universal Turing machine, we can efficiently construct an input on which the program hangs!
A Concrete Unrecognizable Problem: The Non-Acceptance Problem for TMs

A TM $M$ *recognizes* a language $L$ if $M$ accepts exactly those strings in $L$ *(but could run forever on other strings)*

A TM $M$ *decides* a language $L$ if $M$ accepts all strings in $L$ and *rejects* all strings not in $L$

**Theorem:** $L$ is decidable $\iff L$ and $\neg L$ are recognizable
Recall: Given $L \subseteq \Sigma^*$, define $\neg L := \Sigma^* \setminus L$

Theorem: $L$ is decidable

$\iff$ $L$ and $\neg L$ are recognizable

$(\Leftarrow)$ Given: a TM $M_1$ that recognizes $L$ and a TM $M_2$ that recognizes $\neg L$, we want to build a new machine $M$ that decides $L$

How? Any ideas?

$Hint$: $M_1$ always accepts $x$, when $x$ is in $L$

$M_2$ always accepts $x$, when $x$ isn’t in $L$
Recall: Given $L \subseteq \Sigma^*$, define $\neg L := \Sigma^* \setminus L$

Theorem: $L$ is decidable

$\iff L$ and $\neg L$ are recognizable

$(\Leftarrow)$ Given: a TM $M_1$ that recognizes $L$ and a TM $M_2$ that recognizes $\neg L$, we want to build a new machine $M$ that decides $L$

$M(x)$: Run $M_1(x)$ and $M_2(x)$ on separate tapes. Alternate between simulating one step of $M_1$, and one step of $M_2$. If $M_1$ ever accepts, then accept. If $M_2$ ever accepts, then reject
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! Contradiction!
The Halting Problem [Turing]

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

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

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

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

\( M'(M,w) \): 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 \( \text{A}_{TM} \) \( \Rightarrow \) \( H \) does not exist!
Does $M$ halt on $w$?

- Yes: Output answer
- No: Output reject

$M'$ decides $A_{TM}$
THE HALTING PROBLEM IS EASY TO SOLVE. IF THE PROGRAM RUNS TOO LONG, I TAKE THIS STICK AND BEAT THE COMPUTER UNTIL IT STOPS.

What if Alan Turing had been an engineer?
The previous proof is one example of a MUCH more general phenomenon.

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 $L$:

$$A_{TM} \leq L$$

Intuition: $L$ is “at least as hard as” $A_{TM}$

Given the ability to solve problem $L$, we can solve $A_{TM}$
Reducing One Problem to Another

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

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

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

Say: “A is mapping reducible to B”

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

$$w \in A \iff f(w) \in B \iff g(f(w)) \in C$$
Some (Simple) Examples

\[ A_{\text{DFA}} = \{ (D, w) \mid D \text{ encodes a DFA over some } \Sigma, \]
and D accepts \( w \in \Sigma^* \} \]

\[ A_{\text{NFA}} = \{ (N, w) \mid N \text{ encodes an NFA, N accepts } w \} \]

Theorem: For every regular language \( L' \), \( L' \leq_m A_{\text{DFA}} \)

For every regular \( L' \), there’s a DFA \( D \) for \( L' \).
So here’s a mapping reduction \( f \) from \( L' \) to \( A_{\text{DFA}} \):

\[ f(w) := \text{Output } (D,w) \]

Then, \( w \in L' \iff D \text{ accepts } w \iff f(w) = (D,w) \in A_{\text{DFA}} \)

So \( f \) is a mapping reduction from \( L' \) to \( A_{\text{DFA}} \)
Some (Simple) Examples

\[ A_{\text{DFA}} = \{ (D, w) \mid D \text{ encodes a DFA over some } \Sigma, \]
\[ \text{and } D \text{ accepts } w \in \Sigma^* \} \]

\[ A_{\text{NFA}} = \{ (N, w) \mid N \text{ encodes an NFA, } N \text{ accepts } w \} \]

Theorem: \[ A_{\text{DFA}} \leq_m A_{\text{NFA}} \]

Every DFA can be trivially written as an NFA. Here’s a reduction \( f \) from \( A_{\text{DFA}} \) to \( A_{\text{NFA}} \):

\[ f(D,w) := \text{Write down NFA } N \text{ that’s equivalent to } D \]
\[ \quad \text{Output } (N,w) \]

Theorem: \[ A_{\text{NFA}} \leq_m A_{\text{DFA}} \]

\[ f(N,w) := \text{Use the subset construction to convert NFA } N \text{ into an equivalent DFA } D. \] Output \( (D,w) \)
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 $B$.

We build a machine $M'$ deciding $A$ as follows:

$M'(w)$:

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

Then: $w \in A \iff f(w) \in B$ [since $f$ reduces $A$ to $B$]

$\iff M$ accepts $f(w)$ [since $M$ decides $B$]

$\iff M'$ accepts $w$ [by def of $M'$]
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.
Corollary: If $A \leq_m B$ and $A$ is undecidable, then $B$ is undecidable

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

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

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 $(M, w)$. Write down the description of a TM $M'$ with the spec:

"$M'(w) =$ Run $M$ on $w$. If $M$ accepts, then accept; else loop forever"

Output $(M', w)$

Then, $z \in A_{TM} \iff M$ accepts $w$

\[ \iff M' \text{ halts on } w \iff (M', w) \in HALT_{TM} \]

**Corollary:** $HALT_{TM}$ is undecidable
Theorem: \( A_{TM} \leq_m HALT_{TM} \)

Corollary: \( \neg A_{TM} \leq_m \neg HALT_{TM} \) (why?)

Corollary: \( \neg HALT_{TM} \) is unrecognizable!

Proof: If \( \neg HALT_{TM} \) were recognizable, then \( \neg A_{TM} \) would also be recognizable, because \( \neg A_{TM} \leq_m \neg HALT_{TM} \). But it’s not!

Question: \( A_{TM} \leq_m \neg A_{TM} \)?
Theorem: $\text{HALT}_{TM} \leq_m A_{TM}$

Proof: Define the computable function $f$:

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

Construct a TM $M'$ with the specification:

"$M'(w) = \text{Run } M \text{ on } w$.
If $M$ halts, then accept"

Output $(M', w)$

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

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!