Lecture 10: Undecidability, Unrecognizability, and Reductions
Your Midterm: 2:35-3:55pm, 32-144 + 155

No pset this week!
Just an optional (not graded) practice midterm
Solutions to practice midterm will come out with the practice midterm. Also all HW solutions.
When you see the practice midterm...
DON’T PANIC!
Practice midterm will be harder than midterm
Next Thursday (3/19)

Your Midterm: 2:35-3:55pm, 32-144 + 155
No pset this week!
Just an optional (not graded) practice midterm

FAQ: What material is on the midterm?
Everything up to Thursday (Lectures 1-11)

FAQ: Can I bring notes?
Yes, one single-sided sheet of notes, US letter paper
A TM $M$ \textit{recognizes} a language $L$ if $M$ \textit{accepts} exactly those strings in $L$.

A language $L$ is \textit{recognizable} (a.k.a. recursively enumerable) if some TM recognizes $L$.

A TM $M$ \textit{decides} a language $L$ if $M$ \textit{accepts} all strings in $L$ and \textit{rejects} all strings not in $L$.

A language $L$ is \textit{decidable} (a.k.a. recursive) if some TM decides $L$.

$L(M) :=$ set of strings $M$ accepts.
Thm: There are *unrecognizable* languages

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

The proof will be very NON-CONSTRUCTIVE: We will 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.”

Turing Machines

Languages over \( \{0,1\} \)
Theorem: There is no onto function from \( L \) to \( 2^L \)

Proof: Let \( f : L \rightarrow 2^L \) be arbitrary

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

Claim: For all \( x \in L, f(x) \neq S \)

For all \( x \in L, \) observe that

\( x \in S \) if and only if \( x \not\in f(x) \) \[\text{by definition of } S\]

Therefore \( f(x) \neq S: \)

the element \( x \) is in \textit{exactly one} of those sets!

Therefore \( f \) is \textit{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 $L$ (and all subsets of $L$).
Thm: There are *unrecognizable* languages

Proof: Suppose all languages are recognizable. Then for all $L$, there’s a TM $M$ that recognizes $L$. Thus the function $R: \{\text{Turing Machines}\} \rightarrow \{\text{Languages}\}$ defined by $M \mapsto L(M)$ is an onto function.

But we showed there is *no* onto function from $\{\text{Turing Machines}\} \subseteq T$ to its power set $2^T$. Contradiction!
A Concrete Undecidable Problem:
The Acceptance Problem for TMs

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

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

Thm: There are *unrecognizable* languages
$A_{TM} = \{ \langle M, w \rangle \mid M \text{ is a TM that accepts string } w \}$

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

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

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

Define a new TM $D$ with the following spec:

$D(\langle M \rangle):$ Run $H$ on $\langle M, M \rangle$ and output the *opposite* of $H$

$D(\langle D \rangle) = \begin{cases} 
\text{Reject} & \text{if } D \text{ accepts } \langle D \rangle \\
\text{Accept} & \text{if } D \text{ does not accept } \langle D \rangle
\end{cases}$

Set $M = D$?
The table of outputs of $H$ on $\langle x, y \rangle$

<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 on \( \langle x, y \rangle \)

<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>( x )</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( M_1 )</td>
<td>accept</td>
<td>accept</td>
<td>accept</td>
<td>reject</td>
<td></td>
<td>accept</td>
</tr>
<tr>
<td>( M_2 )</td>
<td>reject</td>
<td>accept</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></td>
<td>accept</td>
</tr>
<tr>
<td>( M_4 )</td>
<td>accept</td>
<td>reject</td>
<td>reject</td>
<td>reject</td>
<td></td>
<td>accept</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 on \( \langle x \rangle \) outputs the opposite of H on \( \langle x, x \rangle \)
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>$D$</td>
<td>reject</td>
<td>reject</td>
<td>accept</td>
<td>accept</td>
<td>accept</td>
<td></td>
</tr>
</tbody>
</table>

$D$ on $\langle x \rangle$ outputs the **opposite** of $H$ on $\langle x,x \rangle$

$D$ on $\langle D \rangle$ outputs the **opposite** of $D$ on $\langle D \rangle$
$A_{TM} = \{ \langle M,w \rangle \mid M \text{ is a TM that accepts string } w \}$

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

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

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

Define a new TM $D_U$ as follows:

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

\textbf{Note: There is no contradiction here!}

\( D_U \) must run forever on \( \langle D_U \rangle \)

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

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

1. $\langle D_U, D_U \rangle \notin A_{TM}$ ($D_U$ does not accept $D_U$)
2. $U$ runs forever on the input $\langle D_U, D_U \rangle$

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

Given any universal Turing machine, we can efficiently construct an input on which the program hangs!

Note how generic this argument is: it does not depend on Turing machines!
A Concrete Unrecognizable Problem: The “Non-Acceptance Problem” for TMs

\[ A_{TM} = \{ \langle M, w \rangle \mid M \text{ encodes a TM over some } \Sigma, \]
\[ w \text{ encodes a string over } \Sigma \]
\[ \text{and } M \text{ accepts } w \} \]

We choose a decoding of pairs, TMs, and strings so that every binary string decodes to some TM M and string w

If \( z \in \{0,1\}^* \) doesn’t decode to \( \langle M, w \rangle \) in the usual way, then we define that \( z \) decodes to a TM D and \( \varepsilon \)

where D is a “dummy” TM that accepts nothing.

Then, \( \neg A_{TM} = \{ z \mid z \text{ decodes to } M \text{ and } w \}
\] such that M does not accept w \}
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 if and only if $L$ and $\neg L$ are recognizable
Recall: Given \( L \subseteq \Sigma^* \), define \( \neg L := \Sigma^* \setminus L \)

Theorem: \( L \) is decidable
\[
\iff \text{L and } \neg L \text{ are recognizable}
\]

(\( \iff \)) 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?

\textit{Hint:} \( M_1 \) always accepts \( x \), when \( x \) is in \( L \)
\( M_2 \) always accepts \( x \), when \( x \) isn’t in \( L \)
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]

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

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 $A_{\text{TM}} \Rightarrow H$ does not exist!
Does $M$ halt on $w$?

Output reject

$M'$ decides $A_{TM}$
R. Ryan Williams
@rrwilliams

6.045 health reminder: wash your hands for the time it takes to prove that the Halting problem is undecidable.

10:55 AM · Mar 10, 2020 · Twitter for Android

R. Ryan Williams @rrwilliams · 8m
Replies to @rrwilliams

"Suppose Turing machine H can decide, given any string (M,w), whether TM M halts on w. Define a TM D which, on input (M), runs H on (M,M) and halts iff H rejects. So D on (D) halts iff H on (D,D) rejects iff D on (D) does not halt. D cannot both halt and not halt. Contradiction!"
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 $A_{TM}$
Theorem [Turing]: \( \text{HALT}_{TM} \) is undecidable

Proof: Assume some TM \( H \) decides \( \text{HALT}_{TM} \)

We’ll make an \( M' \) that decides \( A_{TM} \)

\( M'(M,w): \) Run \( H \) on \( \langle M, w \rangle \)

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

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

If \( M \) accepts, then \( \text{accept} \)

If \( M \) rejects, then \( \text{reject} \)

This is called a TURING REDUCTION:

Using a TM for deciding \( \text{HALT}_{TM} \) we could decide \( T_{TM} \)
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 \textit{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

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

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

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

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

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

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

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

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

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

Theorem: \( A_{\text{DFA}} \leq_m A_{\text{NFA}} \)

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

\( f(\langle D, w \rangle) := \text{Write down NFA } N \text{ equivalent to } D \)

Output \( \langle N, w \rangle \)

Theorem: \( A_{\text{NFA}} \leq_m A_{\text{DFA}} \)

\( f(\langle N, w \rangle) := \text{Use subset construction to convert NFA } N \text{ into an equivalent DFA } D. \) Output \( \langle D, w \rangle \)
Theorem: If $A \leq_m B$ and $B$ is decidable, then $A$ is decidable

"If $A$ is as hard as $B$, 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.