Concrete Undecidability and Non-Recognizability
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

Corollary: $\neg A_{TM}$ 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 machine \( D \) 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>...</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></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>...</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>
</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>...</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></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>

$D(x)$ 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: (constructive proof)

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

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

Define a new machine $D_H$ as follows:

$D_H(M):$ 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!

\(H(D_H, D_H)\) 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!

Is there a problem we care about that in non-recognizable?
Theorem: L is decidable iff both L and \( \overline{L} \) are recognizable.

L is decidable (recursive)

L is recognizable (recursively enumerable)
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!
Recall: Given $L \subseteq \Sigma^*$, define $\overline{L} := \Sigma^* \setminus L$

Theorem: $L$ is decidable iff both $L$ and $\overline{L}$ are recognizable

Given:
- a TM $M_1$ that recognizes $L$ and
- a TM $M_2$ that recognizes $\overline{L}$,

want to build a new machine $M$ that decides $L$

How? Any ideas?
$M_1$ 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 both $L$ and $\neg L$ are recognizable

Given:
- a TM $M_1$ that recognizes $L$
- a TM $M_2$ that recognizes $\neg L$,

want to build a new machine $M$ that decides $L$

Simulate $M_1(x)$ on one tape, $M_2(x)$ on another.
Exactly one of the two will accept
If $M_1$ accepts then accept
If $M_2$ accepts then reject
Parting thoughts:
There is at least two problems we care about that are not decidable $A_{TM}$ and $\neg A_{TM}$ (which is not recognizable). Anything else?