Today's learning goals

• Use diagonalization in a proof of undecidability.
• Trace TMs whose inputs are representations of other TMs
• Define and explain core examples of decision problems: $A_{DFA}$, $E_{DFA}$, $EQ_{DFA}$, $A_{TM}$, $HALT_{TM}$

Reminders:
- HW 6 due tonight — JUSTIFY!
- Review session tomorrow
- Exam 3 on Thursday
Each T-recognizable language is recognized by at least one TM so there are no more recognizable languages than there are TMs. Therefore: there are countably many T-recognizable sets.

The collection of all languages is the power set of the set $\Sigma^*$. Since $\Sigma^*$ is countable, the diagonal argument shows that its power set is uncountable.
Satisfied?

• Maybe not …

• What's a specific example of a language that is not Turing-recognizable? or not Turing-decidable?

• Idea: consider set that, were it to be Turing-decidable, would have to "talk" about itself.
$A_{TM}$

Recall $A_{DFA} = \{<B,w> \mid B \text{ is a DFA and } w \text{ is in } L(B) \}$

$A_{TM} = \{<M,w> \mid M \text{ is a TM and } w \text{ is in } L(M) \}$

What is $A_{TM}$?
A. A Turing machine whose input is codes of TMs and strings.
B. A set of pairs of TMs and strings.
C. A set of strings that encode TMs and strings.
D. Not well defined.
E. I don't know.
\[ A_{TM} = \{ <M, w> \mid M \text{ is a TM and } w \text{ is in } L(M) \} \]

Define the TM \( N \) = "On input \( <M, w> \):
1. Simulate \( M \) on \( w \).
2. If \( M \) accepts, accept. If \( M \) rejects, reject."
Define the TM $N$ = "On input $<M,w>$:
1. Simulate $M$ on $w$.
2. If $M$ accepts, accept. If $M$ rejects, reject."

What is $L(N)$?
A. $A_{TM}$
B. Some superset of $A_{TM}$
C. $\{<M,w> \mid M$ is a TM and $w$ is a string$\}$
D. I don't know.
Define the TM $N = "On input <M,w>:"
1. Simulate M on w.
2. If M accepts, accept. If M rejects, reject."

Which statement is true?
A. N decides $A_{TM}$
B. N recognizes $A_{TM}$
C. N always halts
D. I don't know.
Define the TM $N$ = "On input $<M,w>$:
1. Simulate $M$ on $w$.
2. If $M$ accepts, accept. If $M$ rejects, reject."

Conclude: $A_{TM}$ is Turing-recognizable.
Is it decidable?
Diagonalization proof: $A_{TM}$ not decidable

Assume, towards a contradiction, that it is.

I.e. let $M_{ATM}$ be a Turing machine such that for every TM $M$ and every string $w$,

- Computation of $M_{ATM}$ on $<M,w>$ halts and accepts if $w$ is in $L(M)$.
- Computation of $M_{ATM}$ on $<M,w>$ halts and rejects if $w$ is not in $L(M)$. 

Sipser 4.11
If $M_1$ is TM with $L(M_1) = \{ w \mid w \text{ starts with 0} \}$ and $M_1$ does not halt on all strings not in $L(M_1)$, what is the result of the computation of $M_{ATM}$ on $<M_1, 11>$?

A. $M_{ATM}$ halts and accepts.
B. $M_{ATM}$ halts and rejects.
C. $M_{ATM}$ loops.
D. I don't know.

I.e. let $M_{ATM}$ be a Turing machine such that for every TM $M$ and every string $w$,

- Computation of $M_{ATM}$ on $<M,w>$ halts and accepts if $w$ is in $L(M)$.
- Computation of $M_{ATM}$ on $<M,w>$ halts and rejects if $w$ is not in $L(M)$. 
Diagonalization proof: $A_{TM}$ not decidable  

Assume, towards a contradiction, that $M_{ATM}$ decides $A_{TM}$.

Define the TM $D =$ "On input $<M>$:
1. Run $M_{ATM}$ on $<M, <M>>$.
2. If $M_{ATM}$ accepts, reject; if $M_{ATM}$ rejects, accept."
Diagonalization proof: $A_{\text{TM}}$ not decidable

Assume, towards a contradiction, that $M_{\text{ATM}}$ decides $A_{\text{TM}}$.

Define the TM $D =$ "On input $<M>$:
1. Run $M_{\text{ATM}}$ on $<M, <M>>$. **MUST HALT**
2. If $M_{\text{ATM}}$ accepts, reject; if $M_{\text{ATM}}$ rejects, accept."

Is $D$ a decider?
A. Yes: it's a TM that always halts.
B. No: it's a well-defined TM but may loop.
C. No: it's not even a well-defined TM.
D. I don't know.
Diagonalization proof: $A_{TM}$ not decidable  

Assume, towards a contradiction, that $M_{ATM}$ decides $A_{TM}$

Define the TM $D = "On \ input \ <M>:\"
1. Run $M_{ATM}$ on $<M, <M>>$.
2. If $M_{ATM}$ accepts, reject; if $M_{ATM}$ rejects, accept."

If $M_0$ is a TM with $L(M_0) = \emptyset$, what is result of computation of $D$ with input $<M_0>$?
A. Halt and accept.
B. Halt and reject.
C. Loop.
D. I don't know.
Diagonalization proof: $A_{TM}$ not decidable \textit{Sipser 4.11}

Assume, towards a contradiction, that $M_{ATM}$ decides $A_{TM}$.

Define the TM $D =$ "On input $<M>$:
\begin{enumerate}
\item Run $M_{ATM}$ on $<M, <M>>$.
\item If $M_{ATM}$ accepts, reject; if $M_{ATM}$ rejects, accept.
\end{enumerate}"

If $M_1$ is a TM with $L(M_1) = \Sigma^*$, what is result of computation of $D$ with input $<M_1>$?
\begin{enumerate}
\item Halt and accept.
\item Halt and reject.
\item Loop.
\item I don't know.
\end{enumerate}
Diagonalization proof: $A_{TM}$ not decidable

Sipser 4.11

Assume, towards a contradiction, that $M_{ATM}$ decides $A_{TM}$.

Define the TM $D = "On input <M>:"

1. Run $M_{ATM}$ on $<M, <M>>$.
2. If $M_{ATM}$ accepts, reject; if $M_{ATM}$ rejects, accept.

Consider running $D$ on input $<D>$. Because $D$ is a decider:

- either computation halts and accepts ...
- or computation halts and rejects ...

So $M_{ATM}$ rej. $<D, <D>>$ i.e. $<D> \notin L(D)$
Diagonalization proof: $A_{TM}$ not decidable

Assume, towards a contradiction, that $M_{ATM}$ decides $A_{TM}$.

Define the TM $D$ = "On input $<M>$:
1. Run $M_{ATM}$ on $<M, <M>>$.
2. If $M_{ATM}$ accepts, reject; if $M_{ATM}$ rejects, accept."

Consider running $D$ on input $<D>$. Because $D$ is a decider:
- either computation halts and accepts …
- or computation halts and rejects …

Diagonalization???

Self-reference

"Is $<D>$ an element of $L(D)$?"
{a^n b^n | n ≥ 0}

{a^n b^n a^n | n ≥ 0}

??

A_{TM}

Regular

Context-Free

Decidable

Turing-Recognizable
So far

<table>
<thead>
<tr>
<th>Decidable</th>
<th>Recognizable (and not decidable)</th>
<th>Co-recognizable (and not decidable)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A_{DFA}$</td>
<td>$A_{TM}$</td>
<td>$A_{TM}^c$, i.e. $A_{TM}$ is rec.</td>
</tr>
<tr>
<td>$E_{DFA}$</td>
<td>emptiness</td>
<td></td>
</tr>
<tr>
<td>$EQ_{DFA}$</td>
<td>equality</td>
<td></td>
</tr>
</tbody>
</table>
Is there an unrecognizable set?

- **Unsatisfying answer:**
  - "Yes, because of counting arguments"

- How do we prove that a set is not Turing-recognizable?

Later… First, let's get more comfortable with undecidability
So far

<table>
<thead>
<tr>
<th>Decidable</th>
<th>Recognizable (and not decidable)</th>
<th>Not recognizable</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A_{DFA}$</td>
<td>$A_{TM}$</td>
<td>??</td>
</tr>
<tr>
<td>$E_{DFA}$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$EQ_{DFA}$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Give algorithm!

Diagonalization

except are
not dec.
sets that but
Co-rec
Do we have to diagonalize?

• *Turning subroutines on their head …*

\[
\text{HALT}_{\text{TM}} = \{ \langle M, w \rangle \mid M \text{ is a TM and } M \text{ halts on input } w \} \\
\text{A}_{\text{TM}} = \{ \langle M, w \rangle \mid M \text{ is a TM and } w \text{ is in } L(M) \}
\]
No general procedure for bug checks will do.

Now, I won’t just assert that, I’ll prove it to you. I will prove that although you might work till you drop, you cannot tell if computation will stop.

For imagine we have a procedure called P that for specified input permits you to see whether specified source code, with all of its faults, defines a routine that eventually halts.

You feed in your program, with suitable data, and P gets to work, and a little while later (in finite compute time) correctly infers whether infinite looping behavior occurs.

If there will be no looping, then P prints out ‘Good.’ That means work on this input will halt, as it should. But if it detects an unstoppable loop, then P reports ‘Bad!’ — which means you’re in the soup.

Well, the truth is that P cannot possibly be, because if you wrote it and gave it to me, I could use it to set up a logical bind that would shatter your reason and scramble your mind.

Here’s the trick that I’ll use — and it’s simple to do. I’ll define a procedure, which I will call Q, that will use P’s predictions of halting success to stir up a terrible logical mess.

For a specified program, say A, one supplies, the first step of this program called Q I devise is to find out from P what’s the right thing to say of the looping behavior of A run on A.

If P’s answer is ‘Bad!’, Q will suddenly stop. But otherwise, Q will go back to the top, and start off again, looping endlessly back, till the universe dies and turns frozen and black.

And this program called Q wouldn’t stay on the shelf; I would ask it to forecast its run on itself. When it reads its own source code, just what will it do? What’s the looping behavior of Q run on Q?

If P warns of infinite loops, Q will quit; yet P is supposed to speak truly of it!

And if Q’s going to quit, then P should say ‘Good.’ Which makes Q start to loop! (P denied that it would.)

No matter how P might perform, Q will scoop it: Q uses P’s output to make P look stupid.

Whatever P says, it cannot predict Q: P is right when it’s wrong, and is false when it’s true!

I’ve created a paradox, neat as can be — and simply by using your putative P.

When you posited P you stepped into a snare; Your assumption has led you right into my lair.

So where can this argument possibly go? I don’t have to tell you; I’m sure you must know.

A reductio: There cannot possibly be a procedure that acts like the mythical P.

You can never find general mechanical means for predicting the acts of computing machines; it’s something that cannot be done. So we users must find our own bugs. Our computers are losers!
So far

<table>
<thead>
<tr>
<th>Decidable</th>
<th>Recognizable (and not decidable)</th>
<th>Not recognizable</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A_{DFA}$</td>
<td>$A_{TM}$</td>
<td>??</td>
</tr>
<tr>
<td>$E_{DFA}$</td>
<td>$\text{HALT}_{TM}$</td>
<td></td>
</tr>
<tr>
<td>$\text{EQ}_{DFA}$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Give algorithm!

Diagonalization OR reduction