CSE 105
THEORY OF COMPUTATION

Spring 2016

http://cseweb.ucsd.edu/classes/sp16/cse105-ab/
Today's learning goals

- Use counting arguments to prove the existence of unrecognizable (undecidable) languages.
- Determine and prove whether sets are countable.
- Use diagonalization in a proof of uncountability.
- Use diagonalization in a proof of undecidability.
- Define reductions from one problem to another.
- Use reductions to prove undecidability.
- Describe the difference between diagonalization and reduction.
Recall $A_{DFA} = \{<B,w> | B \text{ is a DFA and } w \text{ is in } L(B) \}$

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

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

1. Simulate $M$ on $w$.
2. If $M$ accepts, accept. If $M$ rejects, reject.

What is $L(N)$?

A. $A_{TM} = \{<M,w> | w \in L(M) \}$
B. Some superset of $A_{TM}$
C. $\{<M,w> | M \text{ is a TM and } w \text{ is a string}\}$
D. I don't know.
Define the TM $N = "\text{On input } <M,w>:}\n1. \text{Simulate } M \text{ on } w.\n2. \text{If } M \text{ accepts, accept. If } M \text{ 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 $N = \text{"On input } <M,w>:\n1. \text{ Simulate } M \text{ on } w.\n2. \text{ If } M \text{ accepts, accept. If } M \text{ rejects, reject.}"$

**Conclude:** $A_{TM}$ is Turing-recognizable.

**Is it decidable?**
Diagonalization proof: $A_{TM}$ not decidable

Sipser 4.11

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 $L(M)$. 
Diagonalization proof: $A_{TM}$ not decidable \textit{Sipser} 4.11

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 $L(M)$.

If $X$ is TM with $L(X) = \{ w \mid w \text{ starts with } 0 \}$ and $X$ loops on all strings that are not in $L(X)$, what is result of the computation of $M_{ATM}$ on $<X, 11>$?

A. $M_{ATM}$ halts and accepts.
B. $M_{ATM}$ halts and rejects.
C. $M_{ATM}$ loops.
D. I don't know.
**Diagonalization proof: \( A_{TM} \) not decidable**  

*Sipser 4.11*

The set of all strings is countable, so list it.

<table>
<thead>
<tr>
<th>( M_1 )</th>
<th>( M_2 )</th>
<th>( M_3 )</th>
<th>( M_4 )</th>
<th>( \ldots )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( W_1 )</td>
<td>( \text{??} )</td>
<td>( \text{??} )</td>
<td>( \text{??} )</td>
<td>( \text{??} )</td>
</tr>
<tr>
<td>( W_2 )</td>
<td>( \text{??} )</td>
<td>( \text{??} )</td>
<td>( \text{??} )</td>
<td>( \text{??} )</td>
</tr>
<tr>
<td>( W_3 )</td>
<td>( \text{??} )</td>
<td>( \text{??} )</td>
<td>( \text{??} )</td>
<td>( \text{??} )</td>
</tr>
<tr>
<td>( W_4 )</td>
<td>( \text{??} )</td>
<td>( \text{??} )</td>
<td>( \text{??} )</td>
<td>( \text{??} )</td>
</tr>
<tr>
<td>( W_5 )</td>
<td>( \text{??} )</td>
<td>( \text{??} )</td>
<td>( \text{??} )</td>
<td>( \text{??} )</td>
</tr>
<tr>
<td>( \ldots )</td>
<td>( \text{??} )</td>
<td>( \text{??} )</td>
<td>( \text{??} )</td>
<td>( \text{??} )</td>
</tr>
</tbody>
</table>

\( M_{ATM} \): \( M_{ATM} \) on \( <M,w> \) halts and accepts iff \( w \) is in \( L(M) \).
Diagonalization proof: $A_{\text{TM}}$ not decidable \textit{Sipser 4.11}

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

Idea: Use this machine to build a decider that can’t exist.

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

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: $A_{\text{TM}}$ not decidable

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

Define the TM $D = \text{"On input } <M>:\text{"}$

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

If $M_1$ is a TM with $L(M_1) = \Sigma^*$, what is result of computation of $D$ with input $<M_1>$?

A. Halt and accept.  
B. Halt and reject.  
C. Loop.  
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 = \text{"On input } <M>:\n
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 …
### Decidable

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

Give algorithm!

Diagonalization
Do we have to diagonalize?

- *Turning subroutines on their head …

\[ \text{HALT}_{\text{TM}} = \{ <M,w> \mid M \text{ is a TM and } M \text{ halts on input } w \} \]

\[ A_{\text{TM}} = \{ <M,w> \mid M \text{ is a TM and } w \text{ is in } L(M) \} \]
Do we have to diagonalize?

- **Turning subroutines on their head …**

\[ \text{HALT} = \{ <M,w> \mid M \text{ is a TM and } M \text{ halts on input } w \} \]

\[ \text{A} = \{ <M,w> \mid M \text{ is a TM and } w \text{ is in } L(M) \} \]

**Claim:** \( \text{HALT} \) is undecidable.

**How is \( \text{HALT} \) related to \( \text{A} \) ?**

A. They're the same set.
B. One is a subset of the other.
C. They have the same type of elements but no other relation.
D. I don't know.
Claim: $\text{HALT}_{\text{TM}}$ is undecidable.

- Proof by contradiction …

Assume towards the contrary that $\text{HALT}_{\text{TM}}$ is decided by some TM
Claim: $\text{HALT}_{TM}$ is undecidable.

- Proof by contradiction ...

- Assume we have a machine $R$ that \textit{decides} $\text{HALT}_{TM}$
- Build an algorithm \textit{that uses $R$ as a subroutine} that decides $A_{TM}$
- This is impossible!
Claim: $\text{HALT}_{TM}$ is undecidable.

- Proof: Assume, towards a contradiction, that $\text{HALT}_{TM}$ is decidable and the TM $R$ decides it. Construct a TM $M_{ATM}$ by: "On input $<M,w>$"
  1. Run $R$ on input $<M,w>$.
  2. If $R$ rejects, then reject; else, run $M$ on $w$.
     a. If this computation accepts, accept.
     b. If this computation rejects, reject."

Which of the machines in this proof are deciders?

A. All of them: $R$, $M_{ATM}$, $M$
B. Definitely $R$ and $M_{ATM}$; $M$ may or may not be.
C. Definitely $R$, $M_{ATM}$, and $M$ may or may not be.
D. None of them has to be.
E. I don't know.
Claim: $\text{HALT}_{\text{TM}}$ is undecidable.

- Proof: Assume, towards a contradiction, that $\text{HALT}_{\text{TM}}$ is decidable and the TM $R$ decides it. Construct a TM $M_{\text{ATM}}$ by: "On input $<M,w>$"
  1. Run $R$ on input $<M,w>$.
  2. If $R$ rejects, then reject; else, run $M$ on $w$.
     a. If this computation accepts, accept.
     b. If this computation rejects, reject."

Lemma: $M_{\text{ATM}}$ decides $A_{\text{TM}}$. (Proof of correctness of construction.)

Therefore, $A_{\text{TM}}$ is decidable, a contradiction with our earlier work.
Scooping the Loop Snooper
A proof that the Halting Problem is undecidable
Geoffrey K. Pullum
(http://www.lel.ed.ac.uk/~gpullum/loopsnoop.html)

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>
<tr>
<td>$\text{ALL}_{DFA} (HW)$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Give algorithm!

Diagonalization OR reduction
Reduction?

A problem $P_1$ reduces to a problem $P_2$ if any solution for $P_2$ can be used to solve $P_1$.

In other words: using a solution for $P_2$ as a subroutine gives a solution for $P_1$.

In our example: we used a solution for $\text{HALT}_{\text{TM}}$ to get a solution for $A_{\text{TM}}$. This means that $A_{\text{TM}}$ reduces to $\text{HALT}_{\text{TM}}$. 
Reduction?

A problem $P_1$ reduces to a problem $P_2$ if any solution for $P_2$ can be used to solve $P_1$.

If $P_1$ reduces to $P_2$ and

A. $P_1$ is decidable, then $P_2$ is also decidable.
B. $P_2$ is decidable, then $P_1$ is also decidable.
C. Both of the above.
D. None of the above.
E. I don't know.
Reduction?

A problem $P_1$ **reduces to** a problem $P_2$ if any solution for $P_2$ can be used to solve $P_1$.

If $P_1$ reduces to $P_2$ and

A. $P_1$ is undecidable, then $P_2$ is also undecidable.
B. $P_2$ is undecidable, then $P_1$ is also undecidable.
C. Both of the above.
D. None of the above.
E. I don't know.
Reduction?

A problem $P_1$ reduces to a problem $P_2$ if any solution for $P_2$ can be used to solve $P_1$.

**New strategy:** to prove that a problem is undecidable, prove that a problem we know to be undecidable reduces to it.
Claim: \( E_{\text{TM}} \) is undecidable.

\[
E_{\text{TM}} = \{ <M> \mid M \text{ is a TM and } L(M) \text{ is empty} \}
\]
i.e. want to recognize codes of TMs that always reject / loop

- **Proof by reduction**?

To use proof by reduction to prove that \( E_{\text{TM}} \) is undecidable, we must reduce an undecidable set to \( E_{\text{TM}} \)
Claim: $E_{\text{TM}}$ is undecidable.

- Proof by reduction
  - **Goal**: show that $A_{\text{TM}}$ reduces to $E_{\text{TM}}$.
  - i.e. Build an algorithm that uses a decider for $E_{\text{TM}}$ as a subroutine and that decides $A_{\text{TM}}$
  - Assume: have a TM, $R$, that decides $E_{\text{TM}}$
  - Build: new TM, $M_{\text{ATM}}$, that decides $A_{\text{TM}}$
    - Always halts
    - Accepts iff input $<M,w>$ and $w$ is in $L(M)$. 
Claim: $E_{TM}$ is undecidable.

- **Proof by reduction…**
  - Assume: have a TM, $R$, that decides $E_{TM}$
  - Build: new TM, $M_{ATM}$, that decides $A_{TM}$
    - Always halts
    - Accepts iff input $<M, w>$ and $w$ is in $L(M)$.
  - "On input $<M, w>$:
    - Run $R$ on input $<M>$. If rejects, reject.
    - If accepts, run $M$ on input $w$.
      - If accepts, accept; if reject, reject."
Claim: $E_{TM}$ is undecidable.

- Proof by reduction...
  - Assume: have a TM, $R$, that decides $E_{TM}$.
  - Build: new TM, $M_{ATM}$, that decides $A_{TM}$.
    - Always halts
    - Accepts iff input $<M,w>$ and $w$ is in $L(M)$.
    - "On input $<M,w>$:
      - Run $R$ on input $<M>$. If rejects, reject.
      - If accepts, run $M$ on input $w$.
        - If accepts, accept; if reject, reject."

Does the machine $M_{ATM}$ always halt?

A. Yes.
B. No, not if $L(M)$ is empty.
C. No, not if $L(M)$ is nonempty.
D. No, not if $M$ is not a decider.
E. I don't know.
Claim: $E_{TM}$ is undecidable.

- **Proof by reduction…**
  - Assume: have a TM, R, that decides $E_{TM}$
  - Build: new TM, $M_{ATM}$, that decides $A_{TM}$
    - Always halts
    - Accepts iff input $<M,w>$ and $w$ is in $L(M)$.
  - "On input $<M,w>$:
    - First, build the TM $X$ which, on input $x$, ignores $x$ and simulates $M$ on $w$."

Fixed version
Claim: $E_{TM}$ is undecidable.

- **Proof by reduction...**
  - Assume: have a TM, $R$, that decides $E_{TM}$
  - Build: new TM, $M_{ATM}$, that decides $A_{TM}$
    - Always halts
    - Accepts iff input $<M,w>$ and $w$ is in $L(M)$.
  - "On input $<M,w>$:
    - First, build the TM $X$ which, on input $x$, ignores $x$ and simulates $M$ on $w$.

For a given $<M,w>$, what's $L(X)$?
A. $\{w\}$
B. $w$
C. $\{x \mid x \neq w\}$
D. $\Sigma^*$
E. The empty set.
Claim: $E_{TM}$ is undecidable.

- **Proof by reduction…**
  - Assume: have a TM, $R$, that decides $E_{TM}$
  - Build: new TM, $M_{ATM}$, that decides $A_{TM}$
    - Always halts
    - Accepts iff input $<M,w>$ and $w$ is in $L(M)$.
  - "On input $<M,w>$:
    - First, build the TM $X$ which, on input $x$, ignores $x$ and simulates $M$ on $w$.
    - Run $R$ on $<X>$.
      - If accepts, reject; if rejects; accept."
Proof by reduction…

Assume: have a TM, R, that decides $E_{TM}$
Build: new TM, $M_{ATM}$, that decides $A_{TM}$
  • Always halts
  • Accepts iff input $<M,w>$ and $w$ is in $L(M)$.
  • "On input $<M,w>$:
    • First, build the TM X which, on input x, ignores x and simulates M on w.
    • Run R on $<X>$.
      • If accepts, reject; if rejects; accept."

Correctness: Is $M_{ATM}$ a decider for $A_{TM}$?
So far

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

Give algorithm!

Diagonalization OR reduction