The Halting Problem for Turing Machines

Mark Greenstreet, CpSc 421, Term 1, 2008/09

- The Undecidability of $A_{TM}$
  - Diagonalizing Turing Machines
  - Turing Recognizable $\supset$ Turing Decidable

- Turing Unrecognizable Languages
  - How do we know if $M$ is a decider?
  - The Halting Problem
  - Turing Unrecognizable Languages
Trying to Decide $A_{TM}$

- $A_{TM} = \{ M \#w \mid \text{Turing machine } M \text{ accepts string } w \}$
  - $A_{TM}$ is Turing recognizable:
    We constructed a Turing Machine, $U$ that recognizes $A_{TM}$ in the October 27 lecture.
  - $U$ was not a decider – it would loop on input $M \#w$ if $M$ loops on input $w$.
  - Can we make a Turing machine that decides $A_{TM}$?
    This machine must halt (either accept or reject) for all possible inputs.

- Assume that $E$ is a TM that decides $A_{TM}$.
  We’ll show that this leads to a contradiction on the next few slides.
\( A_{TM} \) Is Undecidable

- \( A_{TM} = \{ M \# w \mid M \text{ describes a TM that accepts string } w \} \)
- Let \( D \) be a Turing machine that does not have \( \# \) in its input alphabet. On input \( w \), \( D \) does the following:
  - Appends \( \# w \) onto its input tape to produce \( w \# w \).
  - Runs \( E \) (the decider for \( A_{TM} \)) as a “subroutine”.
    - If \( E \) accepts \( w \# w \), \( D \) rejects.
    - If \( E \) rejects \( w \# w \), \( D \) accepts.
- Now, run \( D \) with its own description as its input:
  - If \( E \) says that \( D \) accepts when run with \( D \) as input, then \( D \) rejects when run with \( D \) as input.
  - If \( E \) says that \( D \) rejects when run with \( D \) as input, then \( D \) accepts when run with \( D \) as input.
  - Either way, we have a contradiction.
- \( \therefore E \) cannot exist.
  - There is no TM that decides \( A_{TM} \).
  - \( A_{TM} \) is not Turing decidable.
Why is this Diagonalization?

The set of all Turing machines is countable:

- Turing Machines can be described by strings.
  - In the October 27 lecture we described TMs using strings over the alphabet $\Sigma_{TM} = \{0, 1, (, , )\}$.
  - Not all strings are valid TM descriptions. Thus, $|TM| \leq |\Sigma_{TM}^*| = |\mathbb{N}|$.
- For every $k \geq 3$ there is a valid TM with $k$ states. Thus $|TM| \geq |\mathbb{N}|$.
- We conclude that $|TM| = |\mathbb{N}|$.

The set of all languages is uncountable.

The set of all languages has size $2^{\Sigma^*} = 2^{\mathbb{N}}$.

There are more languages than there are Turing machines.

∴ There are languages that are neither Turing decidable nor recognizable.
Why is this Diagonalization?

- The set of all Turing machines is countable:
- The set of all languages is uncountable.
  The set of all languages has size $2^{|\Sigma^*|} = 2^{|\mathbb{N}|}$. For example, with $\Sigma = \{0, 1\}$ we have:

<table>
<thead>
<tr>
<th></th>
<th>$\epsilon$</th>
<th>0</th>
<th>1</th>
<th>00</th>
<th>01</th>
<th>10</th>
<th>11</th>
<th>000</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>$L_0$</td>
<td>$R$</td>
<td>$R$</td>
<td>$R$</td>
<td>$R$</td>
<td>$R$</td>
<td>$R$</td>
<td>$R$</td>
<td>$R$</td>
<td>$R$</td>
</tr>
<tr>
<td>$L_1$</td>
<td>$A$</td>
<td>$R$</td>
<td>$R$</td>
<td>$R$</td>
<td>$R$</td>
<td>$R$</td>
<td>$R$</td>
<td>$R$</td>
<td>$R$</td>
</tr>
<tr>
<td>$L_2$</td>
<td>$R$</td>
<td>$A$</td>
<td>$R$</td>
<td>$R$</td>
<td>$R$</td>
<td>$R$</td>
<td>$R$</td>
<td>$R$</td>
<td>$R$</td>
</tr>
<tr>
<td>$L_3$</td>
<td>$A$</td>
<td>$A$</td>
<td>$R$</td>
<td>$R$</td>
<td>$R$</td>
<td>$R$</td>
<td>$R$</td>
<td>$R$</td>
<td>$R$</td>
</tr>
<tr>
<td>$L_4$</td>
<td>$R$</td>
<td>$R$</td>
<td>$A$</td>
<td>$R$</td>
<td>$R$</td>
<td>$R$</td>
<td>$R$</td>
<td>$R$</td>
<td>$R$</td>
</tr>
<tr>
<td>...</td>
<td>:</td>
<td>:</td>
<td>:</td>
<td>:</td>
<td>:</td>
<td>:</td>
<td>:</td>
<td>:</td>
<td>:</td>
</tr>
</tbody>
</table>

- There are more languages than there are Turing machines.
  $\therefore$ There are languages that are neither Turing decidable nor recognizable.
Consider the matrix where entry $(i, j)$ is 1 iff Turing machine $i$ accepts the string that encodes Turing machine $j$:

$$
\begin{array}{cccccccc}
M_0 & M_1 & M_2 & \ldots & M_{117} & M_{118} & M_{119} & \ldots \\
M_0 & \infty & \infty & \infty & \ldots & \infty & \infty & \infty & \ldots \\
M_2 & R & R & R & \ldots & R & R & R & \ldots \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
M_{117} & A & \infty & R & \ldots & R & R & A & \ldots \\
M_{118} & R & R & R & \ldots & \infty & \infty & \infty & \ldots \\
M_{119} & R & A & \infty & \ldots & R & A & A & \ldots \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
\end{array}
$$

Let $L_D$ be the language

$$\{ M_i \mid \text{Turing machine } M_i \text{ does not accept input } M_i \}$$
Consider the matrix where entry \((i, j)\) is 1 iff Turing machine \(i\) accepts the string that encodes Turing machine \(j\):

\[
\begin{array}{cccccccc}
M_0 & M_1 & M_2 & \ldots & M_{117} & M_{118} & M_{119} & \ldots \\
M_0 & R & R & R & \ldots & R & R & R & \ldots \\
M_2 & R & R & R & \ldots & R & R & R & \ldots \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
M_{117} & A & \infty & R & \ldots & R & R & A & \ldots \\
M_{118} & R & R & R & \ldots & \infty & \infty & \infty & \ldots \\
M_{119} & R & A & \infty & \ldots & R & A & A & \ldots \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
\end{array}
\]

Let \(L_D\) be the language
\[
\{M_i \mid \text{Turing machine } M_i \text{ does not accept input } M_i\}:
\]

\[
\begin{array}{cccccccc}
M_0 & M_1 & M_2 & \ldots & M_{117} & M_{118} & M_{119} & \ldots \\
L_D & A & R & A & \ldots & A & A & R & \ldots \\
\end{array}
\]
Constructing an Undecidable Language

- Consider the matrix where entry \((i, j)\) is 1 iff Turing machine \(i\) accepts the string that encodes Turing machine \(j\):
- Let \(L_D\) be the language
  \[
  \{M_i \mid \text{Turing machine } M_i \text{ does not accept input } M_i\}:
  \]
  \[
  \begin{array}{ccccccc}
  M_0 & M_1 & M_2 & \ldots & M_{117} & M_{118} & M_{119} & \ldots \\
  L_D & A & R & A & \ldots & A & A & R & \ldots \\
  \end{array}
  \]
- There is no TM in our list that recognizes \(L_D\) – that’s the diagonalization.
- \(L_D\) is the language that we tried to construct \(D\) to decide.
Diagonalization and Halting

- $A_{TM}$ is not Turing decidable (slide 3).
- $A_{TM}$ is Turing recognizable (October 27 lecture).
  - The set of Turing recognizable languages is strictly larger than the set of Turing decidable languages.
  - This is because a recognizer is allowed to loop: failure to halt means the input string is not in the language recognized by the recognizer.
- $L_D = \{ M \mid M \#M \in A_{TM} \}$ is not Turing recognizable (slide 5).
  - This is because the recognizer must halt whenever $M$ loops when run with input $M$.
  - In fact, we could modify our machines to never use the reject state — they could just loop to reject.
  - Then, recognizing $L_D$ would mean determining that the machine will never halt.
  - Our argument that $L_D$ is not Turing recognizable shows that this variant is not Turing recognizable.

- $\therefore \text{HALT} = \{ M \#w \mid \text{Turing machine } M \text{ halts when run with input } w \}$ is Turing recognizable but not Turing decidable.
  - $\text{HALT}$ is not even Turing recognizable.
The class of Turing decidable languages is closed under complement.

The class of Turing recognizable languages is not closed under complement.

We say that a language, $L$, is Turing co-recognizable iff the complement of $L$ is Turing recognizable.

For example, the language $LOOPS = \{ M \# w \mid \text{Turing machine } M \text{ loops when run with input } w \}$ is Turing co-recognizable because it is the complement of $HALT$, a Turing recognizable language.
Relating Recognizability

- If a language is Turing recognizable and Turing co-recognizable, then it is Turing decidable.
  - Let $L$ be a language that is both Turing recognizable and co-recognizable.
  - Because $L$ is Turing recognizable, there is a Turing machine, $M_L$ that for any $w \in L$ accepts $w$, and for any $w \notin L$ rejects or loops.
  - Because $L$ is Turing co-recognizable, there is a Turing machine, $M_{co-L}$ that for any $w \notin L$ rejects $w$, and for any $w \in L$ accepts or loops.
  - Now, we build a new TM, $N$ that has two tapes, one for $M_L$ and one for $M_{co-L}$. Each step of $L$ takes a step for each of $M_L$ and $M_{co-L}$. If $M_L$ accepts or $M_{co-L}$ rejects, then $N$ accepts. Likewise, if $M_L$ rejects or $M_{co-L}$ accepts, $N$ rejects. $N$ is guaranteed to halt.
  - $N$ is a TM that decides $L$.
  - $\therefore L$ is Turing decidable.
Why Allow Loopy Machines?

- Couldn’t we just insist that we’ll only consider TM’s that halt on all inputs (i.e. deciders)?

- Problem 1:
  - We could do this, and our diagonalization would still work.
  - The obvious way to construct a TM for the diagonal (slide 3) produces a TM that loops. Language $L_D$ remains undecidable.

- Problem 2: How do we know if a TM is a decider?
  - This is the question of whether or not a TM halts on all inputs, not just on one, specific input.
  - We say that a TM is total iff it halts on all inputs, and we write

  $$TOTAL = \{ M \mid M \text{ is a TM that halts on all inputs} \}$$

  - The language $TOTAL$ is neither Turing recognizable nor co-recognizable.
  - Thus, deciding whether or not a TM is a decider is even harder than the halting problem.
This coming week (and beyond)

- **Reading**
  - Today: Sipser, 4.1
  - Oct. 29 (Today): Sipser, 4.2
  - Oct. 31 (Friday): Sipser, 5.1
  - Nov. 3 (Monday): Midterm review.
  - Nov. 5 (A week from Today): Midterm 2.

- **Homework**
  - Oct. 31 (Friday): Homework 7 due, Homework 8 goes out.
    No late homework accepted for homework 7.
    Homework 8 is extra credit.
Where did $E$ come from?

The proof is by contradiction. To prove that $A_{TM}$ is undecidable, we assume the opposite, namely that $A_{TM}$ is decidable, and show that this leads to a contradiction. This contradiction shows that one of our assumptions must have been false. In particular, it shows that our assumption that $A_{TM}$ is not undecidable (i.e. that it is decidable) is false. From that, we conclude that $A_{TM}$ is undesirable.

You can think of this as a “game with an adversary.”

- You claim that $A_{TM}$ is Turing undecidable.
- I (the adversary) claims that $A_{TM}$ is Turing decidable.
- You go to the definition of “Turing decidable:”

A language is Turing decidable iff there exists a TM that decides it. A TM, $M$, decides a language $A$ iff for every input string $w$:

- if $w \in A$ then $M$ accepts $w$;
- if $w \notin A$, $M$ rejects $w$;
- there is no $w$ for which $M$ loops.

Based on this definition, you ask me to show you a TM that decides $A_{TM}$.

Continued on the next slide.
Where did $E$ come from?

You can think of this as a “game with an adversary.”

- You claim that $A_{TM}$ is Turing undecidable.
- I (the adversary) claims that $A_{TM}$ is Turing decidable.
- You ask me to show you a TM that decides $A_{TM}$.
- I give you the description of some TM, $E$.

This is where $E$ comes from: I (the adversary) am obligated to produce an $E$ for you if $A_{TM}$ is indeed Turing decidable.

Based on $E$, you construct a new TM, $D$ such that

- $D$ accepts $w$ if $E$ rejects $w\#w$;
- $D$ rejects $w$ if $E$ accepts $w\#w$.

Because $E$ is a decider, $E$ never loops. Thus, $D$ never loops as well. See slide 3 for more details on how to construct $D$.

Now, you propose running $D$ with the string that describes $D$ as its input.

(continued on the next slide).
Where did $E$ come from?

You can think of this as a “game with an adversary.”

Now, you propose running $D$ with the string that describes $D$ as its input.

- $D$ constructs the string $D\#D$ and hands it to $E$.
- From the definition of $A_{TM}$, $E$ accepts $D\#D$ iff $D$ accepts when run with its own description as its input. If fact, we are running $D$ with its own description as its input.
- If $E$ accepts then $D$ rejects. This means that $E$ said that $D$ accepts when run with its own description as its input, and $D$ in fact rejected when run with its own description as its input.
- If $E$ rejects then $D$ accepts. This means that $E$ said that $D$ rejects when run with its own description as its input, and $D$ in fact accepted when run with its own description as its input.
- Either way, $E$ is wrong. Thus, $E$ is not the decider for $A_{TM}$ that I (the adversary) claimed it is.

This shows that there is no TM that decides $A_{TM}$. In other words, $A_{TM}$ is not Turing decidable.
Where did $E$ come from?

- Game with an adversary (summary):
  - You claim that $A_{TM}$ is Turing undecidable.
  - I (the adversary) claims that $A_{TM}$ is Turing decidable.
  - You ask me to show you a TM that decides $A_{TM}$.
  - I give you the description of some TM, $E$.
  - Based on $E$, you construct a new TM, $D$ such that accepts $w$ iff $E$ rejects $w\#w$.
  - Now, you propose running $D$ with the string that describes $D$ as its input.
    - $D$ constructs the string $D\#D$ and hands it to $E$.
    - If $E$ accepts then $D$ rejects and thereby contradicts $E$’s decision.
    - If $E$ rejects then $D$ accepts and thereby contradicts $E$’s decision.
    - Either way, $E$ is wrong.
  - This shows that there is no TM that decides $A_{TM}$. In other words, $A_{TM}$ is not Turing decidable.
Undecidability FAQ: does $E$ loop?

- Can we conclude that $E$ loops when run input $D\#D$?
  - This may seem reasonable, this is what machine $U$ from the October 27 slides would do.
  - But that’s not the only way that $E$ can fail.
    - For example, we could keep track of all configurations that we’ve seen so far and detect looping if a configuration is repeated.
    - We could apply more sophisticated tests as well, but
    - What if one of these tests is wrong?
  - $E$ could report that TM $M$ accepts string $w$ when $M$ in fact loops on input $w$.
    - How would you know that $E$ was wrong?
    - You could try running $M$ with input $w$, but if after a while you came back and told me that it seems to be looping even though $E$ says it should accept, I can reply that you just haven’t run it for long enough yet.
    - How can you determine that you’ve run $M$ long enough? – How can you decide that $E$ is wrong?
    - In general, you can’t.
  - See the next slide for a bit more.
Can we conclude that $E$ loops when run input $D\#D$?

It was Penrose’s mistake in *The Emperor’s New Mind*.

- Penrose assumed that because $E$ would be wrong if it accepted or if it rejected, then $E$ must loop when run with $D\#D$ as described above.
- BUT, $E$ is wrong if it loops.
  - $E$ is supposed to be a decider.
  - If I (in the adversary argument described above) give you a TM that loops for some inputs and claim that it’s a decider, then I’ve failed to hold up my end of the bargain.
  - If $E$ is supposed to be a decider and it loops, then it is just as wrong as it is if it incorrectly accepts or rejects.

When Penrose concluded that $E$ loops, he inserted a contradiction into his argument because he had previously assumed that $E$ was a decider.

- Given that Penrose was arguing from inconsistent assumption, he could conclude anything.
- Penrose has the excuse that he’s a brilliant physicist who happens to be clueless about computer science.
- You are a computer science student and don’t have Penrose’s excuse. Read this FAQ and avoid those mistakes – you wouldn’t want to embarrass yourself at a party this weekend making silly claims about decidability results.