High-Level Descriptions of Computation

- Instead of giving a Turing Machine, we shall often describe a program as code in some programming language (or often “pseudo-code”)
  - Possibly using high level data structures and subroutines
    (Recall that TM and RAM are equivalent (even polynomially))
- Inputs and outputs are complex objects, encoded as strings
- Examples of objects:
  - Matrices, graphs, geometric shapes, images, videos,...
  - DFAs, NFAs, Turing Machines, Algorithms, other machines ...
“Everything” finite can be encoded as a (finite) string of symbols from a finite alphabet (e.g. ASCII)

- Can in turn be encoded in binary (as modern day computers do). No special symbol: use self-terminating representations

Example: encoding a “graph.”

\[(1,2,3,4)((1,2)(2,3)(3,1)(1,4))\]

encodes the graph

![Graph Diagram]

2 - 1 - 4

3
High-Level Descriptions of Computation

- We have already seen several algorithms, for problems involving complex objects like DFAs, NFAs, regular expressions, and Turing Machines
  - For example, convert a NFA to DFA; Given a NFA $N$ and a word $w$, decide if $w \in L(N)$; …
- All these inputs can be encoded as strings and all these algorithms can be implemented as Turing Machines
- Some of these algorithms are for decision problems, while others are for computing more general functions
- All these algorithms terminate on all inputs
High-Level Descriptions of Computation

Examples: Problems regarding Computation

Some more decision problems that have algorithms that always halt (sketched in the textbook)

1. On input $\langle B, w \rangle$ where $B$ is a DFA and $w$ is a string, decide if $B$ accepts $w$.
   Algorithm: simulate $B$ on $w$ and accept iff simulated $B$ accepts

2. On input $\langle B \rangle$ where $B$ is a DFA, decide if $L(B) = \emptyset$.
   Algorithm: Use a fixed point algorithm to find all reachable states. See if any final state is reachable.

Code is just data: A TM can take “the code of a program” (DFA, NFA or TM) as part of its input and analyze or even execute this code
High-Level Descriptions of Computation

Examples: Problems regarding Computation

Some more decision problems that have algorithms that always halt (sketched in the textbook)

- On input \( \langle B, w \rangle \) where \( B \) is a DFA and \( w \) is a string, decide if \( B \) accepts \( w \).
  
  Algorithm: simulate \( B \) on \( w \) and accept iff simulated \( B \) accepts

- On input \( \langle B \rangle \) where \( B \) is a DFA, decide if \( L(B) = \emptyset \).
  
  Algorithm: Use a fixed point algorithm to find all reachable states. See if any final state is reachable.

**Code is just data:** A TM can take “the code of a program” (DFA, NFA or TM) as part of its input and analyze or even execute this code.

**Universal Turing Machine** (a simple “Operating System”): Takes a TM \( M \) and a string \( w \) and simulates the execution of \( M \) on \( w \)
Decidable and Recognizable Languages

Recall: Definition
A Turing machine $M$ is said to recognize a language $L$ if $L = L(M)$.
A Turing machine $M$ is said to decide a language $L$ if $L = L(M)$
and $M$ halts on every input.

▶ Every finite language is decidable: For example, by a TM that
has all the strings in the language “hard-coded” into it
▶ We just saw some example algorithms all of which terminate
in a finite number of steps, and output yes or no (accept or
reject). i.e., They decide the corresponding languages.
Decidable and Recognizable Languages

Recall: Definition
A Turing machine $M$ is said to recognize a language $L$ if $L = L(M)$. A Turing machine $M$ is said to decide a language $L$ if $L = L(M)$ and $M$ halts on every input.

$L$ is said to be Turing-recognizable (Recursively Enumerable (R.E.) or simply recognizable) if there exists a TM $M$ which recognizes $L$. $L$ is said to be Turing-decidable (Recursive or simply decidable) if there exists a TM $M$ which decides $L$. 

Every finite language is decidable: For example, by a TM that has all the strings in the language "hard-coded" into it. We just saw some example algorithms all of which terminate in a finite number of steps, and output yes or no (accept or reject).
Decidable and Recognizable Languages

Recall: Definition
A Turing machine $M$ is said to recognize a language $L$ if $L = L(M)$. A Turing machine $M$ is said to decide a language $L$ if $L = L(M)$ and $M$ halts on every input.

$L$ is said to be Turing-recognizable (Recursively Enumerable (R.E.) or simply recognizable) if there exists a TM $M$ which recognizes $L$. $L$ is said to be Turing-decidable (Recursive or simply decidable) if there exists a TM $M$ which decides $L$.

- Every finite language is decidable
Decidable and Recognizable Languages

Recall: Definition
A Turing machine $M$ is said to recognize a language $L$ if $L = L(M)$. A Turing machine $M$ is said to decide a language $L$ if $L = L(M)$ and $M$ halts on every input.

$L$ is said to be Turing-recognizable (Recursively Enumerable (R.E.) or simply recognizable) if there exists a TM $M$ which recognizes $L$. $L$ is said to be Turing-decidable (Recursive or simply decidable) if there exists a TM $M$ which decides $L$.

- Every finite language is decidable: For example, by a TM that has all the strings in the language “hard-coded” into it
- We just saw some example algorithms all of which terminate in a finite number of steps, and output yes or no (accept or reject). i.e., They decide the corresponding languages.
Decidable and Recognizable Languages

- But not all languages are decidable! We will show:
  - \( A_{TM} = \{(M, w) \mid M \text{ is a TM and } M \text{ accepts } w\} \) is undecidable
Decidable and Recognizable Languages

- But not all languages are decidable! We will show:
  - $A_{TM} = \{ \langle M, w \rangle \mid M \text{ is a TM and } M \text{ accepts } w \}$ is undecidable
- However $A_{TM}$ is Turing-recognizable!
Decidable and Recognizable Languages

- But not all languages are decidable! We will show:
  - $A_{TM} = \{ \langle M, w \rangle \mid M \text{ is a TM and } M \text{ accepts } w \}$ is undecidable
  - However $A_{TM}$ is Turing-recognizable!

Proposition

There are languages which are recognizable, but not decidable
Recognizing $A_{TM}$

Program $U$ for recognizing $A_{TM}$:

On input $\langle M, w \rangle$
  simulate $M$ on $w$
  if simulated $M$ accepts $w$, then accept
  else reject (by moving to $q_{rej}$)
Recognizing $A_{TM}$

Program $U$ for recognizing $A_{TM}$:

On input $\langle M, w \rangle$
  simulate $M$ on $w$
  if simulated $M$ accepts $w$, then accept
  else reject (by moving to $q_{rej}$)

$U$ (the Universal TM) accepts $\langle M, w \rangle$ iff $M$ accepts $w$. i.e.,

$$L(U) = A_{TM}$$
Recognizing $A_{TM}$

Program $U$ for recognizing $A_{TM}$:

On input $\langle M, w \rangle$

- simulate $M$ on $w$
- if simulated $M$ accepts $w$, then accept
- else reject (by moving to $q_{rej}$)

$U$ (the Universal TM) accepts $\langle M, w \rangle$ iff $M$ accepts $w$. i.e.,

$$L(U) = A_{TM}$$

But $U$ does not decide $A_{TM}$
Recognizing $A_{\text{TM}}$

Program $U$ for recognizing $A_{\text{TM}}$:

On input $\langle M, w \rangle$
- simulate $M$ on $w$
- if simulated $M$ accepts $w$, then accept
- else reject (by moving to $q_{\text{rej}}$)

$U$ (the Universal TM) accepts $\langle M, w \rangle$ iff $M$ accepts $w$. i.e.,

$$L(U) = A_{\text{TM}}$$

But $U$ does not decide $A_{\text{TM}}$: If $M$ rejects $w$ by not halting, $U$ rejects $\langle M, w \rangle$ by not halting.
Recognizing $A_{\text{TM}}$

Program $U$ for recognizing $A_{\text{TM}}$:

On input $\langle M, w \rangle$
- simulate $M$ on $w$
  - if simulated $M$ accepts $w$, then accept
  - else reject (by moving to $q_{\text{rej}}$)

$U$ (the Universal TM) accepts $\langle M, w \rangle$ iff $M$ accepts $w$. i.e.,

$$L(U) = A_{\text{TM}}$$

But $U$ does not decide $A_{\text{TM}}$: If $M$ rejects $w$ by not halting, $U$ rejects $\langle M, w \rangle$ by not halting. Indeed (as we shall see) no TM decides $A_{\text{TM}}$. 
Deciding vs. Recognizing

Proposition

If $L$ and $\overline{L}$ are recognizable, then $L$ is decidable

Proof.

Program $P$ for deciding $L$, given programs $P_L$ and $P_{\overline{L}}$ for recognizing $L$ and $\overline{L}$:
Deciding vs. Recognizing

**Proposition**

*If* $L$ *and* $\overline{L}$ *are recognizable, then* $L$ *is decidable*

**Proof.**

Program $P$ for **deciding** $L$, given programs $P_L$ and $P_{\overline{L}}$ for recognizing $L$ and $\overline{L}$:

- On input $x$, simulate $P_L$ and $P_{\overline{L}}$ on input $x$. 
Deciding vs. Recognizing

Proposition

*If $L$ and $\overline{L}$ are recognizable, then $L$ is decidable*

Proof.

Program $P$ for deciding $L$, given programs $P_L$ and $P_{\overline{L}}$ for recognizing $L$ and $\overline{L}$:

- On input $x$, simulate $P_L$ and $P_{\overline{L}}$ on input $x$. Whether $x \in L$ or $x \notin L$, one of $P_L$ and $P_{\overline{L}}$ will halt in finite number of steps.

- Which one to simulate first?
Deciding vs. Recognizing

Proposition
If \( L \) and \( \overline{L} \) are recognizable, then \( L \) is decidable

Proof.
Program \( P \) for deciding \( L \), given programs \( P_L \) and \( P_{\overline{L}} \) for recognizing \( L \) and \( \overline{L} \):

- On input \( x \), simulate \( P_L \) and \( P_{\overline{L}} \) on input \( x \). Whether \( x \in L \) or \( x \notin L \), one of \( P_L \) and \( P_{\overline{L}} \) will halt in finite number of steps.
- Which one to simulate first? Either could go on forever.
Deciding vs. Recognizing

Proposition

If $L$ and $\overline{L}$ are recognizable, then $L$ is decidable.

Proof.

Program $P$ for deciding $L$, given programs $P_L$ and $P_{\overline{L}}$ for recognizing $L$ and $\overline{L}$:

- On input $x$, simulate $P_L$ and $P_{\overline{L}}$ on input $x$. Whether $x \in L$ or $x \notin L$, one of $P_L$ and $P_{\overline{L}}$ will halt in finite number of steps.
- Which one to simulate first? Either could go on forever.
- On input $x$, simulate in parallel $P_L$ and $P_{\overline{L}}$ on input $x$ until either $P_L$ or $P_{\overline{L}}$ accepts.
Deciding vs. Recognizing

Proposition
If $L$ and $\overline{L}$ are recognizable, then $L$ is decidable

Proof.
Program $P$ for deciding $L$, given programs $P_L$ and $P_{\overline{L}}$ for recognizing $L$ and $\overline{L}$:

- On input $x$, simulate $P_L$ and $P_{\overline{L}}$ on input $x$. Whether $x \in L$ or $x \notin L$, one of $P_L$ and $P_{\overline{L}}$ will halt in finite number of steps.
- Which one to simulate first? Either could go on forever.
- On input $x$, simulate in parallel $P_L$ and $P_{\overline{L}}$ on input $x$ until either $P_L$ or $P_{\overline{L}}$ accepts
- If $P_L$ accepts, accept $x$ and halt. If $P_{\overline{L}}$ accepts, reject $x$ and halt.
Deciding vs. Recognizing

Proof (contd).

In more detail, \( P \) works as follows:

On input \( x \)
for \( i = 1, 2, 3, \ldots \)

simulate \( P_L \) on input \( x \) for \( i \) steps
simulate \( P_{\bar{L}} \) on input \( x \) for \( i \) steps
if either simulation accepts, break
if \( P_L \) accepted, accept \( x \) (and halt)
if \( P_{\bar{L}} \) accepted, reject \( x \) (and halt)
Deciding vs. Recognizing

Proof (contd).

In more detail, $P$ works as follows:

On input $x$
for $i = 1, 2, 3, \ldots$
    simulate $P_L$ on input $x$ for $i$ steps
    simulate $P_{\overline{L}}$ on input $x$ for $i$ steps
    if either simulation accepts, break
if $P_L$ accepted, accept $x$ (and halt)
if $P_{\overline{L}}$ accepted, reject $x$ (and halt)

(Alternately, maintain configurations of $P_L$ and $P_{\overline{L}}$, and in each iteration of the loop advance both their simulations by one step.)
Deciding vs. Recognizing

So far:

- \( A_{TM} \) is undecidable (will learn soon)
- But it is recognizable

Note: Decidable languages are closed under complementation, but recognizable languages are not.
Deciding vs. Recognizing

So far:

* $A_{TM}$ is undecidable (will learn soon)
* But it is recognizable
* Is every language recognizable?

Note: Decidable languages are closed under complementation, but recognizable languages are not.
Deciding vs. Recognizing

So far:

- $A_{TM}$ is undecidable (will learn soon)
- But it is recognizable
- Is every language recognizable? No!

Proposition

$A_{TM}$ is unrecognizable
Proof.

If $A_{TM}$ is recognizable, since $A_{TM}$ is recognizable, the two languages will be decidable too! □

Note: Decidable languages are closed under complementation, but recognizable languages are not.
Deciding vs. Recognizing

So far:
- $A_{TM}$ is undecidable (will learn soon)
- But it is recognizable
- Is every language recognizable? No!

Proposition
\[ \overline{A_{TM}} \text{ is unrecognizable} \]
Deciding vs. Recognizing

So far:

- \( A_{TM} \) is undecidable (will learn soon)
- But it is recognizable
- Is every language recognizable? No!

Proposition

\( \overline{A}_{TM} \) is unrecognizable

Proof.

If \( \overline{A}_{TM} \) is recognizable, since \( A_{TM} \) is recognizable, the two languages will be decidable too!

\( \blacksquare \)
Deciding vs. Recognizing

So far:
- $A_{TM}$ is undecidable (will learn soon)
- But it is recognizable
- Is every language recognizable? No!

Proposition

$\overline{A_{TM}}$ is unrecognizable

Proof.
If $\overline{A_{TM}}$ is recognizable, since $A_{TM}$ is recognizable, the two languages will be decidable too!

Note: Decidable languages are closed under complementation, but recognizable languages are not.
Decision Problems and Languages

- A decision problem requires checking if an input (string) has some property. Thus, a decision problem is a function from strings to boolean.
- A decision problem is represented as a formal language consisting of those strings (inputs) on which the answer is “yes”.
Recursive Enumerability

- A Turing Machine on an input $w$ either (halts and) accepts, or (halts and) rejects, or never halts.
Recursive Enumerability

- A Turing Machine on an input $w$ either (halts and) accepts, or (halts and) rejects, or never halts.
- The language of a Turing Machine $M$, denoted as $L(M)$, is the set of all strings $w$ on which $M$ accepts.
Recursive Enumerability

- A Turing Machine on an input \( w \) either (halts and) accepts, or (halts and) rejects, or never halts.
- The language of a Turing Machine \( M \), denoted as \( L(M) \), is the set of all strings \( w \) on which \( M \) accepts.
- A language \( L \) is recursively enumerable/Turing recognizable if there is a Turing Machine \( M \) such that \( L(M) = L \).
Decidability

A language $L$ is **decidable** if there is a Turing machine $M$ such that $L(M) = L$ and $M$ halts on every input.
Decidability

- A language $L$ is **decidable** if there is a Turing machine $M$ such that $L(M) = L$ and $M$ halts on every input.
- Thus, if $L$ is decidable then $L$ is recursively enumerable.
Undecidability

Definition
A language \( L \) is **undecidable** if \( L \) is not decidable.
Undecidability

Definition
A language $L$ is **undecidable** if $L$ is not decidable. Thus, there is no Turing machine $M$ that halts on every input and $L(M) = L$.

- This means that either $L$ is not recursively enumerable. That is, there is no Turing machine $M$ such that $L(M) = L$, or
- $L$ is recursively enumerable but not decidable. That is, any Turing machine $M$ such that $L(M) = L$, $M$ does not halt on some inputs.
Big Picture

Languages

Recursively Enumerable (Recognizable)

Decidable (Recursive)

CFL

Regular

Relationship between classes of Languages
Machines as Strings

- For the rest of this lecture, let us fix the input alphabet to be \( \{0, 1\} \)
Machines as Strings

- For the rest of this lecture, let us fix the input alphabet to be \{0, 1\}; a string over any alphabet can be encoded in binary.
Machines as Strings

- For the rest of this lecture, let us fix the input alphabet to be \{0, 1\}; a string over any alphabet can be encoded in binary.
- Any Turing Machine/program $M$ can itself be encoded as a binary string.
Machines as Strings

- For the rest of this lecture, let us fix the input alphabet to be \{0, 1\}; a string over any alphabet can be encoded in binary.
- Any Turing Machine/program $M$ can itself be encoded as a binary string. Moreover every binary string can be thought of as encoding a TM/program.
Machines as Strings

- For the rest of this lecture, let us fix the input alphabet to be \{0, 1\}; a string over any alphabet can be encoded in binary.
- Any Turing Machine/program $M$ can itself be encoded as a binary string. Moreover every binary string can be thought of as encoding a TM/program. (If not the correct format, considered to be the encoding of a default TM.)
For the rest of this lecture, let us fix the input alphabet to be \{0, 1\}; a string over any alphabet can be encoded in binary.

Any Turing Machine/program $M$ can itself be encoded as a binary string. Moreover every binary string can be thought of as encoding a TM/program. (If not the correct format, considered to be the encoding of a default TM.)

We will consider decision problems (language) whose inputs are Turing Machine (encoded as a binary string)
The Diagonal Language

**Definition**
Define $L_d = \{ M \mid M \not\in L(M) \}$. 

Thus, $L_d$ is the collection of Turing machines (programs) $M$ such that $M$ does not halt and accept (i.e. either reject or never ends) when given itself as input.
Definition
Define $L_d = \{ M \mid M \not\in L(M) \}$. Thus, $L_d$ is the collection of Turing machines (programs) $M$ such that $M$ does not halt and accept (i.e. either reject or never ends) when given itself as input.
A non-Recursively Enumerable Language

Proposition

$L_d$ is not recursively enumerable.
A non-Recursively Enumerable Language

Proposition

\( L_d \) is not recursively enumerable.

Proof.
Recall that,
Proposition
\(L_d \text{ is not recursively enumerable.}\)

Proof.
Recall that,

- Inputs are strings over \(\{0, 1\}\)
A non-Recursively Enumerable Language

Proposition
$L_d$ is not recursively enumerable.

Proof.
Recall that,

- Inputs are strings over \{0, 1\}
- Every Turing Machine can be described by a binary string and every binary string can be viewed as Turing Machine
A non-Recursively Enumerable Language

Proposition

$L_d$ is not recursively enumerable.

Proof.
Recall that,

- Inputs are strings over \{0, 1\}
- Every Turing Machine can be described by a binary string and every binary string can be viewed as Turing Machine
- In what follows, we will denote the $i$th binary string (in lexicographic order) as the number $i$. 
A non-Recursively Enumerable Language

Proposition
$L_d$ is not recursively enumerable.

Proof.
Recall that,
- Inputs are strings over \{0, 1\}
- Every Turing Machine can be described by a binary string and every binary string can be viewed as a Turing Machine
- In what follows, we will denote the $i$th binary string (in lexicographic order) as the number $i$. Thus, we can say $j \in L(i)$, which means that the Turing machine corresponding to $i$th binary string accepts the $j$th binary string.
Completing the proof
Diagonalization: Cantor

Proof (contd).

We can organize all programs and inputs as a (infinite) matrix, where the \((i, j)\)th entry is \(Y\) if and only if \(j \in L(i)\).

\[
\begin{array}{cccccccc}
\text{Inputs} & 1 & 2 & 3 & 4 & 5 & 6 & 7 & \cdots \\
\hline
\text{TMs} \downarrow & N & N & N & N & N & N & N & N \\
3 & N & N & N & Y & N & Y & Y & Y \\
5 & N & Y & N & Y & Y & N & N & N \\
6 & N & N & Y & N & Y & N & Y & Y \\
\end{array}
\]

For the sake of contradiction, suppose \(L_d\) is recognized by a Turing machine. Say by the \(j\)th binary string, i.e., \(L_d = L(j)\).

But \(j \in L_d\) iff \(j \not\in L(j)\)! More concretely, suppose \(j \not\in L(j)\) – note that \(j\) can be a string or a TM. Then, by definition, \(j \in L_d = L(j)\). The other case \(j \in L(j)\) can be handled similarly. □
Completing the proof
Diagonalization: Cantor

Proof (contd).

We can organize all programs and inputs as a (infinite) matrix, where the \((i, j)\)th entry is \(Y\) if and only if \(j \in L(i)\).

\[
\begin{array}{cccccccc}
\text{Inputs} & \rightarrow \\
1 & 2 & 3 & 4 & 5 & 6 & 7 & \cdots \\
\hline
\text{TMs} & 1 & N & N & N & N & N & N & N \\
3 & Y & N & Y & N & Y & Y & Y \\
4 & N & Y & N & Y & Y & N & N \\
5 & N & Y & N & Y & Y & N & N \\
6 & N & N & Y & N & Y & N & Y \\
\end{array}
\]

For the sake of contradiction, suppose \(L_d\) is recognized by a Turing machine. Say by the \(j\)th binary string. i.e., \(L_d = L(j)\).
Completing the proof
Diagonalization: Cantor

Proof (contd).

We can organize all programs and inputs as a (infinite) matrix, where the $(i, j)$th entry is $Y$ if and only if $j \in L(i)$.

Inputs $\rightarrow$

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>$\cdots$</th>
</tr>
</thead>
<tbody>
<tr>
<td>TMs</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>$\mathbf{N}$</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>2</td>
<td>N</td>
<td>$\mathbf{N}$</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>3</td>
<td>Y</td>
<td>N</td>
<td>$\mathbf{Y}$</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>4</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>$\mathbf{Y}$</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>5</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>$\mathbf{Y}$</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>6</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>$\mathbf{N}$</td>
<td>Y</td>
<td>Y</td>
</tr>
</tbody>
</table>

For the sake of contradiction, suppose $L_d$ is recognized by a Turing machine. Say by the $j$th binary string. i.e., $L_d = L(j)$. But $j \in L_d$ iff $j \notin L(j)$! More concretly, suppose $j \notin L(j)$ – note that $j$ can be a string or a TM. Then, by definition, $j \in L_d = L(j)$. The other case $j \in L(j)$ can be handled similarly. □
Acceptor for $L_d$?

Consider the following program

On input $i$
  Run program $i$ on $i$
  Output ‘‘yes’’ if $i$ does not accept $i$
  Output ‘‘no’’ if $i$ accepts $i$
Acceptor for $L_d$?

Consider the following program

**On input** $i$
- **Run program** $i$ **on** $i$
- **Output** "yes" **if** $i$ **does not accept** $i$
- **Output** "no" **if** $i$ **accepts** $i$

Does the above program recognize $L_d$?
Acceptor for $L_d$?

Consider the following program

On input $i$
  Run program $i$ on $i$
  Output ‘‘yes’’ if $i$ does not accept $i$
  Output ‘‘no’’ if $i$ accepts $i$

Does the above program recognize $L_d$? No, because it may never output “yes” if $i$ does not halt on $i$. 
Recursively Enumerable but not Decidable

$L_d$ is not recursively enumerable, and therefore not decidable.
Recursively Enumerable but not Decidable

- $L_d$ not recursively enumerable, and therefore not decidable. Are there languages that are recursively enumerable but not decidable?
Recursively Enumerable but not Decidable

- $L_d$ not recursively enumerable, and therefore not decidable.
Are there languages that are recursively enumerable but not decidable?

- Yes, $A_{TM} = \{ \langle M, w \rangle \mid M \text{ is a TM and } M \text{ accepts } w \}$
Proposition

$A_{TM}$ is r.e. but not decidable.
Proposition

\( A_{TM} \) is r.e. but not decidable.

Proof.

We have already seen that \( A_{TM} \) is r.e.
The Universal Language

Proposition

\( A_{TM} \) is r.e. but not decidable.

Proof.

We have already seen that \( A_{TM} \) is r.e. Suppose (for contradiction) \( A_{TM} \) is decidable. Then there is a TM \( M \) that always halts and \( L(M) = A_{TM} \).
The Universal Language

Proposition

\( A_{TM} \) is r.e. but not decidable.

Proof.

We have already seen that \( A_{TM} \) is r.e. Suppose (for contradiction) \( A_{TM} \) is decidable. Then there is a TM \( M \) that always halts and \( L(M) = A_{TM} \). Consider a TM \( D \) as follows:

On input \( i \)
- Run \( M \) on input \( \langle i, i \rangle \)
- Output ‘‘yes’’ if \( i \) rejects \( i \)
- Output ‘‘no’’ if \( i \) accepts \( i \)

Observe that \( L(D) \neq L(M) \) which gives us the contradiction. □
Proposition

\( A_{\text{TM}} \) is r.e. but not decidable.

Proof.
We have already seen that \( A_{\text{TM}} \) is r.e. Suppose (for contradiction) \( A_{\text{TM}} \) is decidable. Then there is a TM \( M \) that always halts and \( L(M) = A_{\text{TM}} \). Consider a TM \( D \) as follows:

On input \( i \)
- Run \( M \) on input \( \langle i, i \rangle \)
- Output ‘‘yes’’ if \( i \) rejects \( i \)
- Output ‘‘no’’ if \( i \) accepts \( i \)

Observe that \( L(D) = L_d \)! 
Proposition

\(A_{\text{TM}}\) is r.e. but not decidable.

Proof.
We have already seen that \(A_{\text{TM}}\) is r.e. Suppose (for contradiction) \(A_{\text{TM}}\) is decidable. Then there is a TM \(M\) that always halts and \(L(M) = A_{\text{TM}}\). Consider a TM \(D\) as follows:

On input \(i\)

- Run \(M\) on input \(\langle i, i \rangle\)
- Output ‘‘yes’’ if \(i\) rejects \(i\)
- Output ‘‘no’’ if \(i\) accepts \(i\)

Observe that \(L(D) = L_d\)! But, \(L_d\) is not r.e. which gives us the contradiction. \(\square\)
A more complete Big Picture

Languages

Recursively Enumerable

Decidable

CFL

Regular

$L_d, \overline{A_{TM}}$

$A_{TM}$

$L_{anbncn}$

$L_{0n1n}$
Reductions

A reduction is a way of converting one problem into another problem such that a solution to the second problem can be used to solve the first problem. We say the first problem reduces to the second problem.
A reduction is a way of converting one problem into another problem such that a solution to the second problem can be used to solve the first problem. We say the first problem reduces to the second problem.

- Informal Examples: Measuring the area of rectangle reduces to measuring the length of the sides.
Reductions

A **reduction** is a way of converting one problem into another problem such that a solution to the second problem can be used to solve the first problem. We say the first problem **reduces** to the second problem.

▶ **Informal Examples:** Measuring the area of rectangle reduces to measuring the length of the sides; Solving a system of linear equations reduces to inverting a matrix.
A **reduction** is a way of converting one problem into another problem such that a solution to the second problem can be used to solve the first problem. We say the first problem reduces to the second problem.

**Informal Examples:** Measuring the area of rectangle reduces to measuring the length of the sides; Solving a system of linear equations reduces to inverting a matrix

**The problem** $L_d$ **reduces to the problem** $A_{TM}$ **as follows:** “To see if $w \in L_d$ check if $\langle w, w \rangle \in A_{TM}$.”
Undecidability using Reductions

Proposition

Suppose $L_1$ reduces to $L_2$ and $L_1$ is undecidable. Then $L_2$ is undecidable.

Proof Sketch.
Suppose for contradiction $L_2$ is decidable. Then there is a $M$ that always halts and decides $L_2$. Then the following algorithm decides $L_1$:

\[ \text{On input } w, \text{ apply reduction to transform } w \text{ into an input } w' \text{ for problem 2.} \]
\[ \text{Run } M \text{ on } w', \text{ and use its answer.} \]
Undecidability using Reductions

Proposition

Suppose $L_1$ reduces to $L_2$ and $L_1$ is undecidable. Then $L_2$ is undecidable.

Proof Sketch.

Suppose for contradiction $L_2$ is decidable. Then there is a $M$ that always halts and decides $L_2$. Then the following algorithm decides $L_1$.
Undecidability using Reductions

Proposition

Suppose $L_1$ reduces to $L_2$ and $L_1$ is undecidable. Then $L_2$ is undecidable.

Proof Sketch.

Suppose for contradiction $L_2$ is decidable. Then there is a $M$ that always halts and decides $L_2$. Then the following algorithm decides $L_1$

- On input $w$, apply reduction to transform $w$ into an input $w'$ for problem 2
- Run $M$ on $w'$, and use its answer.
Reductions schematically
Schematic View

Reduction $f$

Reductions schematically
Algorithm for Problem 1

Reduction $f$

$w \rightarrow f(w)$

Algorithm for Problem 2

Reductions schematically

yes

no
Algorithm for Problem 1

Reduction $f$

Algorithm for Problem 2

$w \rightarrow f(w) \rightarrow \text{yes, no}$

Reductions schematically
The Halting Problem

Proposition

The language $\text{HALT} = \{\langle M, w \rangle \mid M \text{ halts on input } w\}$ is undecidable.
The Halting Problem

Proposition

The language \( \text{HALT} = \{ \langle M, w \rangle \mid M \text{ halts on input } w \} \) is undecidable.

Proof.

We will reduce \( A_{\text{TM}} \) to \( \text{HALT} \). Based on a machine \( M \), let us consider a new machine \( f(M) \) as follows:
The Halting Problem

Proposition

The language $HALT = \{\langle M, w \rangle \mid M \text{ halts on input } w \}$ is undecidable.

Proof.

We will reduce $A_{TM}$ to $HALT$. Based on a machine $M$, let us consider a new machine $f(M)$ as follows:

On input $x$

- Run $M$ on $x$
- If $M$ accepts then halt and accept
- If $M$ rejects then go into an infinite loop
The Halting Problem

Proposition

The language $\text{HALT} = \{ \langle M, w \rangle \mid M \text{ halts on input } w \}$ is undecidable.

Proof.

We will reduce $A_{\text{TM}}$ to HALT. Based on a machine $M$, let us consider a new machine $f(M)$ as follows:

On input $x$

Run $M$ on $x$

If $M$ accepts then halt and accept
If $M$ rejects then go into an infinite loop

Observe that $f(M)$ halts on input $w$ if and only if $M$ accepts $w$. 
The Halting Problem
Completing the proof

Proof (contd).
Suppose HALT is decidable. Then there is a Turing machine $H$ that always halts and $L(H) = \text{HALT}$. But, $\text{A}_{tm}$ is undecidable, which gives us the contradiction. □
The Halting Problem
Completing the proof

Proof (contd).
Suppose HALT is decidable. Then there is a Turing machine $H$ that always halts and $L(H) = \text{HALT}$. Consider the following program $T$

On input $\langle M, w \rangle$
- Construct program $f(M)$
- Run $H$ on $\langle f(M), w \rangle$
- Accept if $H$ accepts and reject if $H$ rejects

But, $A_{tm}$ is undecidable, which gives us the contradiction. $\square$
Proof (contd).

Suppose HALT is decidable. Then there is a Turing machine $H$ that always halts and $L(H) = \text{HALT}$. Consider the following program $T$

On input $\langle M, w \rangle$
- Construct program $f(M)$
- Run $H$ on $\langle f(M), w \rangle$
- Accept if $H$ accepts and reject if $H$ rejects

$T$ decides $A_{\text{TM}}$. 
The Halting Problem
Completing the proof

Proof (contd).

Suppose HALT is decidable. Then there is a Turing machine $H$ that always halts and $L(H) = \text{HALT}$. Consider the following program $T$

On input $\langle M, w \rangle$
   Construct program $f(M)$
   Run $H$ on $\langle f(M), w \rangle$
   Accept if $H$ accepts and reject if $H$ rejects

$T$ decides $A_{TM}$. But, $A_{TM}$ is undecidable, which gives us the contradiction. \(\square\)
Mapping Reductions

Definition
A function $f : \Sigma^* \to \Sigma^*$ is computable if there is some Turing Machine $M$ that on every input $w$ halts with $f(w)$ on the tape.
Mapping Reductions

Definition
A function $f : \Sigma^* \rightarrow \Sigma^*$ is computable if there is some Turing Machine $M$ that on every input $w$ halts with $f(w)$ on the tape.

Definition
A mapping/many-one reduction from $A$ to $B$ is a computable function $f : \Sigma^* \rightarrow \Sigma^*$ such that

$$w \in A \text{ if and only if } f(w) \in B$$
Mapping Reductions

Definition
A function $f : \Sigma^* \rightarrow \Sigma^*$ is computable if there is some Turing Machine $M$ that on every input $w$ halts with $f(w)$ on the tape.

Definition
A mapping/many-one reduction from $A$ to $B$ is a computable function $f : \Sigma^* \rightarrow \Sigma^*$ such that

$$w \in A \text{ if and only if } f(w) \in B$$

In this case, we say $A$ is mapping/many-one reducible to $B$, and we denote it by $A \leq_m B$. 
In this course, we will drop the adjective “mapping” or “many-one”, and simply talk about reductions and reducibility.
Reductions and Recursive Enumerability

Proposition

If $A \leq_m B$ and $B$ is recursively enumerable then $A$ is recursively enumerable.
Reductions and Recursive Enumerability

Proposition

If $A \leq_m B$ and $B$ is recursively enumerable then $A$ is recursively enumerable.

Proof.
Let $f$ be the reduction from $A$ to $B$ and let $M_B$ be the Turing Machine recognizing $B$. 
Proposition

If $A \leq_m B$ and $B$ is recursively enumerable then $A$ is recursively enumerable.

Proof.
Let $f$ be the reduction from $A$ to $B$ and let $M_B$ be the Turing Machine recognizing $B$. Then the Turing machine recognizing $A$ is

On input $w$
- Compute $f(w)$
- Run $M_B$ on $f(w)$
- Accept if $M_B$ does and reject if $M_B$ rejects
Corollary

If $A \leq_m B$ and $A$ is not recursively enumerable then $B$ is not recursively enumerable.
Proposition

If $A \leq_m B$ and $B$ is decidable then $A$ is decidable.
Proposition
If \( A \leq_m B \) and \( B \) is decidable then \( A \) is decidable.

Proof.
Let \( M_B \) be the Turing machine deciding \( B \) and let \( f \) be the reduction. Then the algorithm deciding \( A \), on input \( w \), computes \( f(w) \) and runs \( M_B \) on \( f(w) \).
Reductions and Decidability

Proposition
If $A \leq_m B$ and $B$ is decidable then $A$ is decidable.

Proof.
Let $M_B$ be the Turing machine deciding $B$ and let $f$ be the reduction. Then the algorithm deciding $A$, on input $w$, computes $f(w)$ and runs $M_B$ on $f(w)$. □

Corollary
If $A \leq_m B$ and $A$ is undecidable then $B$ is undecidable.
Mapping Reductions

Definition
A function \( f : \Sigma^* \rightarrow \Sigma^* \) is computable if there is some Turing Machine \( M \) that on every input \( w \) halts with \( f(w) \) on the tape.

Definition
A reduction (a.k.a. mapping reduction/many-one reduction) from a language \( A \) to a language \( B \) is a computable function \( f : \Sigma^* \rightarrow \Sigma^* \) such that

\[
    w \in A \text{ if and only if } f(w) \in B
\]

In this case, we say \( A \) is reducible to \( B \), and we denote it by \( A \leq_m B \).
Proposition

If $A \leq_m B$ and $B$ is r.e., then $A$ is r.e.

Proof.

Let $f$ be a reduction from $A$ to $B$ and let $M_B$ be a Turing Machine recognizing $B$. Then the Turing machine recognizing $A$ is

On input $w$

- Compute $f(w)$
- Run $M_B$ on $f(w)$
- Accept if $M_B$ accepts, and reject if $M_B$ rejects  

Corollary

If $A \leq_m B$ and $A$ is not r.e., then $B$ is not r.e.
Proposition

If $A \leq_m B$ and $B$ is decidable, then $A$ is decidable.

Proof.
Let $f$ be a reduction from $A$ to $B$ and let $M_B$ be a Turing Machine deciding $B$. Then a Turing machine that decides $A$ is

On input $w$
- Compute $f(w)$
- Run $M_B$ on $f(w)$
- Accept if $M_B$ accepts, and reject if $M_B$ rejects

Corollary

If $A \leq_m B$ and $A$ is undecidable, then $B$ is undecidable.
The Halting Problem

Proposition

The language $\text{HALT} = \{ \langle M, w \rangle \mid M \text{ halts on input } w \}$ is undecidable.

Proof.
Recall $A_{\text{TM}} = \{ \langle M, w \rangle \mid w \in L(M) \}$ is undecidable. Will give reduction $f$ to show $A_{\text{TM}} \leq_m \text{HALT} \implies \text{HALT} \text{ undecidable}$.
Let $f(\langle M, w \rangle) = \langle N, w \rangle$ where $N$ is a TM that behaves as follows:

On input $x$

- Run $M$ on $x$
  - If $M$ accepts then halt and accept
  - If $M$ rejects then go into an infinite loop

$N$ halts on input $w$ if and only if $M$ accepts $w$. □
Proposition

The language $\text{HALT} = \{ \langle M, w \rangle \mid M \text{ halts on input } w \}$ is undecidable.

Proof.
Recall $A_{\text{TM}} = \{ \langle M, w \rangle \mid w \in L(M) \}$ is undecidable. Will give reduction $f$ to show $A_{\text{TM}} \leq_m \text{HALT} \implies \text{HALT}$ undecidable.

Let $f(\langle M, w \rangle) = \langle N, w \rangle$ where $N$ is a TM that behaves as follows:

**On input $x$**
- Run $M$ on $x$
- If $M$ accepts then halt and accept
- If $M$ rejects then go into an infinite loop

$N$ halts on input $w$ if and only if $M$ accepts $w$. i.e., $\langle M, w \rangle \in A_{\text{TM}}$ iff $f(\langle M, w \rangle) \in \text{HALT}$
Proposition

The language $E_{TM} = \{ M \mid L(M) = \emptyset \}$ is not decidable.

Note: in fact, $E_{TM}$ is not recognizable.
Emptiness of Turing Machines

Proposition

The language $E_{TM} = \{ M \mid L(M) = \emptyset \}$ is not decidable.

Note: in fact, $E_{TM}$ is not recognizable.

Proof.

Recall $A_{TM} = \{ \langle M, w \rangle \mid w \in L(M) \}$ is undecidable.
Emptiness of Turing Machines

Proposition
The language $E_{TM} = \{ M \mid L(M) = \emptyset \}$ is not decidable.

Note: in fact, $E_{TM}$ is not recognizable.

Proof.
Recall $A_{TM} = \{ \langle M, w \rangle \mid w \in L(M) \}$ is undecidable. For the sake of contradiction, suppose there is a decider $B$ for $E_{TM}$. 


Emptiness of Turing Machines

Proposition

The language $E_{TM} = \{ M \mid L(M) = \emptyset \}$ is not decidable.

Note: in fact, $E_{TM}$ is not recognizable.

Proof.
Recall $A_{TM} = \{ \langle M, w \rangle \mid w \in L(M) \}$ is undecidable. For the sake of contradiction, suppose there is a decider $B$ for $E_{TM}$. Then we first transform $\langle M, w \rangle$ to $\langle M_1 \rangle$ which is the following:

On input $x$

If $x \neq w$, reject

else run $M$ on $w$, and accept if $M$ accepts $w$

, and accept if $B$ rejects $\langle M_1 \rangle$, and rejects if $B$ accepts $\langle M_1 \rangle$. 
Emptiness of Turing Machines

Proposition

The language $E_{TM} = \{ M \mid L(M) = \emptyset \}$ is not decidable.

Note: in fact, $E_{TM}$ is not recognizable.

Proof.

Recall $A_{TM} = \{ \langle M, w \rangle \mid w \in L(M) \}$ is undecidable. For the sake of contradiction, suppose there is a decider $B$ for $E_{TM}$. Then we first transform $\langle M, w \rangle$ to $\langle M_1 \rangle$ which is the following:

On input $x$
- If $x \neq w$, reject
- else run $M$ on $w$, and accept if $M$ accepts $w$

, and accept if $B$ rejects $\langle M_1 \rangle$, and rejects if $B$ accepts $\langle M_1 \rangle$.

Then we show that (1) if $\langle M, w \rangle \in A_{TM}$, then accept, and (2) $\langle M, w \rangle \in A_{TM}$, then reject. (how?)
Emptiness of Turing Machines

Proposition

The language $E_{TM} = \{ M \mid L(M) = \emptyset \}$ is not decidable.

Note: in fact, $E_{TM}$ is not recognizable.

Proof.

Recall $A_{TM} = \{ \langle M, w \rangle \mid w \in L(M) \}$ is undecidable. For the sake of contradiction, suppose there is a decider $B$ for $E_{TM}$. Then we first transform $\langle M, w \rangle$ to $\langle M_1 \rangle$ which is the following:

On input $x$

- If $x \neq w$, reject
- else run $M$ on $w$, and accept if $M$ accepts $w$

, and accept if $B$ rejects $\langle M_1 \rangle$, and rejects if $B$ accepts $\langle M_1 \rangle$.

Then we show that (1) if $\langle M, w \rangle \in A_{TM}$, then accept, and (2) $\langle M, w \rangle \in A_{TM}$, then reject. (how?) This implies $A_{TM}$ is decidable, which is a contradiction. □
Proposition

The language \( \text{REGULAR} = \{ M \mid L(M) \text{ is regular} \} \) is undecidable.
Checking Regularity

Proposition

The language \( \text{REGULAR} = \{ M \mid L(M) \text{ is regular} \} \) is undecidable.

Proof.

We give a reduction \( f \) from \( A_{TM} \) to \( \text{REGULAR} \).
Proposition

The language \( \text{REGULAR} = \{ M \mid L(M) \text{ is regular} \} \) is undecidable.

Proof.

We give a reduction \( f \) from \( A_{\text{TM}} \) to \( \text{REGULAR} \). Let \( f(\langle M, w \rangle) = N \), where \( N \) is a TM that works as follows:

On input \( x \)

   If \( x \) is of the form \( 0^n1^n \) then accept \( x \)
   else run \( M \) on \( w \) and accept \( x \) only if \( M \) does
Checking Regularity

Proposition

The language \(\text{REGULAR} = \{ M \mid L(M) \text{ is regular} \} \) is undecidable.

Proof.

We give a reduction \(f\) from \(A_{\text{TM}}\) to \(\text{REGULAR}\). Let \(f(\langle M, w \rangle) = N\), where \(N\) is a TM that works as follows:

On input \(x\)

- If \(x\) is of the form \(0^n1^n\) then accept \(x\)
- else run \(M\) on \(w\) and accept \(x\) only if \(M\) does

If \(w \in L(M)\) then \(L(N) =\)
Proposition

The language \( \text{REGULAR} = \{M \mid L(M) \text{ is regular}\} \) is undecidable.

Proof.

We give a reduction \( f \) from \( A_{\text{TM}} \) to \( \text{REGULAR} \). Let \( f(\langle M, w \rangle) = N \), where \( N \) is a TM that works as follows:

On input \( x \)
- If \( x \) is of the form \( 0^n1^n \) then accept \( x \)
- else run \( M \) on \( w \) and accept \( x \) only if \( M \) does

If \( w \in L(M) \) then \( L(N) = \Sigma^* \).
Proposition

The language \( \text{REGULAR} = \{ M \mid L(M) \text{ is regular} \} \) is undecidable.

Proof.

We give a reduction \( f \) from \( A_{\text{TM}} \) to \( \text{REGULAR} \). Let \( f(\langle M, w \rangle) = N \), where \( N \) is a TM that works as follows:

On input \( x \)

- If \( x \) is of the form \( 0^n1^n \) then accept \( x \)
- Else run \( M \) on \( w \) and accept \( x \) only if \( M \) does

\( f(\langle M, w \rangle) = N \).

If \( w \in L(M) \) then \( L(N) = \Sigma^* \). If \( w \notin L(M) \) then

\( L(N) = \)
Proposition

The language $\text{REGULAR} = \{ M \mid L(M) \text{ is regular} \}$ is undecidable.

Proof.

We give a reduction $f$ from $A_{\text{TM}}$ to $\text{REGULAR}$. Let $f(\langle M, w \rangle) = N$, where $N$ is a TM that works as follows:

On input $x$
- If $x$ is of the form $0^n1^n$ then accept $x$
- Else run $M$ on $w$ and accept $x$ only if $M$ does

If $w \in L(M)$ then $L(N) = \Sigma^*$. If $w \not\in L(M)$ then $L(N) = \{0^n1^n \mid n \geq 0\}$. 
Checking Regularity

Proposition

The language \( \text{REGULAR} = \{ M \mid L(M) \text{ is regular} \} \) is undecidable.

Proof.

We give a reduction \( f \) from \( A_{\text{TM}} \) to \( \text{REGULAR} \). Let \( f(\langle M, w \rangle) = N \), where \( N \) is a TM that works as follows:

On input \( x \)
- If \( x \) is of the form \( 0^n1^n \) then accept \( x \)
- else run \( M \) on \( w \) and accept \( x \) only if \( M \) does

If \( w \in L(M) \) then \( L(N) = \Sigma^* \). If \( w \not\in L(M) \) then \( L(N) = \{ 0^n1^n \mid n \geq 0 \} \). Thus, \( \langle N \rangle \in \text{REGULAR} \) if and only if \( \langle M, w \rangle \in A_{\text{TM}} \). \( \square \)
Checking Equality

Proposition

\[ EQ_{\text{TM}} = \{ \langle M_1, M_2 \rangle \mid L(M_1) = L(M_2) \} \text{ is not r.e.} \]
Proposition

\[ EQ_{TM} = \{ \langle M_1, M_2 \rangle \mid L(M_1) = L(M_2) \} \text{ is not r.e.} \]

Proof.

We will give a reduction \( f \) from \( E_{TM} \) (assume that we know \( E_{TM} \) is R.E.) to \( EQ_{TM} \).
Checking Equality

Proposition
\[ EQ_{TM} = \{ \langle M_1, M_2 \rangle \mid L(M_1) = L(M_2) \} \text{ is not r.e.} \]

Proof.
We will give a reduction \( f \) from \( E_{TM} \) (assume that we know \( E_{TM} \) is R.E.) to \( EQ_{TM} \). Let \( M_1 \) be the Turing machine that on any input, halts and rejects
Checking Equality

Proposition

\[ EQ_{\text{TM}} = \{ \langle M_1, M_2 \rangle \mid L(M_1) = L(M_2) \} \text{ is not r.e.} \]

Proof.

We will give a reduction \( f \) from \( E_{\text{TM}} \) (assume that we know \( E_{\text{TM}} \) is R.E.) to \( EQ_{\text{TM}} \). Let \( M_1 \) be the Turing machine that on any input, halts and rejects i.e., \( L(M_1) = \emptyset \). Take \( f(M) = \langle M, M_1 \rangle \).
Checking Equality

Proposition

$EQ_{tm} = \{\langle M_1, M_2 \rangle \mid L(M_1) = L(M_2) \}$ is not r.e.

Proof.

We will give a reduction $f$ from $E_{tm}$ (assume that we know $E_{tm}$ is R.E.) to $EQ_{tm}$. Let $M_1$ be the Turing machine that on any input, halts and rejects i.e., $L(M_1) = \emptyset$. Take $f(M) = \langle M, M_1 \rangle$. Observe $M \in E_{tm}$ iff $L(M) = \emptyset$ iff $L(M) = L(M_1)$ iff $\langle M, M_1 \rangle \in EQ_{tm}$. □
Checking Properties

Given $M$

Does $L(M)$ contain $M$?
Is $L(M)$ non-empty? \{ Undecidable
Is $L(M)$ empty?
Is $L(M)$ infinite?
Is $L(M)$ finite?
Is $L(M)$ co-finite (i.e., is $\overline{L(M)}$ finite)?
Is $L(M) = \Sigma^*$?

Which of these properties can be decided?

By Rice's Theorem
Checking Properties

Given $M$

Does $L(M)$ contain $M$?  
Is $L(M)$ non-empty?  
Is $L(M)$ empty?  
Is $L(M)$ infinite?  
Is $L(M)$ finite?  
Is $L(M)$ co-finite (i.e., is $\overline{L(M)}$ finite)?  
Is $L(M) = \Sigma^*$?

Which of these properties can be decided? None!

By Rice's Theorem
Checking Properties

Given $M$

Does $L(M)$ contain $M$? \{ Undecidable 
Is $L(M)$ non-empty? \{ Undecidable 
Is $L(M)$ empty? 
Is $L(M)$ infinite? 
Is $L(M)$ finite? 
Is $L(M)$ co-finite (i.e., is $\overline{L(M)}$ finite)? 
Is $L(M) = \Sigma^*$? 

Which of these properties can be decided? None! By Rice’s Theorem
Properties

Definition
A *property of languages* is simply a set of languages.
Definition

A *property of languages* is simply a set of languages. We say $L$ *satisfies* the property $\mathcal{P}$ if $L \in \mathcal{P}$. 

**Example:**

\[
\{M | L(M) \text{ is infinite}\}; E_{tm} = \{M | L(M) = \emptyset\}
\]

**Non-example:**

\[
\{M | M \text{ has 15 states}\}
\]

← This is a property of TMs, and not languages!
Properties

Definition
A *property of languages* is simply a set of languages. We say $L$ *satisfies* the property $\mathbb{P}$ if $L \in \mathbb{P}$.

Definition
For any property $\mathbb{P}$, define language $L_\mathbb{P}$ to consist of Turing Machines which accept a language in $\mathbb{P}$:

$$L_\mathbb{P} = \{ M | L(M) \in \mathbb{P} \}$$
Properties

Definition
A property of languages is simply a set of languages. We say $L$ satisfies the property $\mathbb{P}$ if $L \in \mathbb{P}$.

Definition
For any property $\mathbb{P}$, define language $L_\mathbb{P}$ to consist of Turing Machines which accept a language in $\mathbb{P}$:

$$L_\mathbb{P} = \{ M \mid L(M) \in \mathbb{P} \}$$

Deciding $L_\mathbb{P}$: deciding if a language represented as a TM satisfies the property $\mathbb{P}$.

▶ Example: $\{ M \mid L(M) \text{ is infinite} \}$
Properties

Definition
A *property of languages* is simply a set of languages. We say $L$ *satisfies* the property $\mathbb{P}$ if $L \in \mathbb{P}$.

Definition
For any property $\mathbb{P}$, define language $L_\mathbb{P}$ to consist of Turing Machines which accept a language in $\mathbb{P}$:

$$L_\mathbb{P} = \{ M \mid L(M) \in \mathbb{P} \}$$

Deciding $L_\mathbb{P}$: deciding if a language represented as a TM satisfies the property $\mathbb{P}$.

▶ Example: $\{ M \mid L(M) \text{ is infinite} \}$; $E_{TM} = \{ M \mid L(M) = \emptyset \}$
Properties

Definition
A *property of languages* is simply a set of languages. We say $L$ *satisfies* the property $\mathbb{P}$ if $L \in \mathbb{P}$.

Definition
For any property $\mathbb{P}$, define language $L_\mathbb{P}$ to consist of Turing Machines which accept a language in $\mathbb{P}$:

$$L_\mathbb{P} = \{M \mid L(M) \in \mathbb{P}\}$$

Deciding $L_\mathbb{P}$: deciding if a language represented as a TM satisfies the property $\mathbb{P}$.

- **Example:** $\{M \mid L(M) \text{ is infinite}\}$; $E_{\text{TM}} = \{M \mid L(M) = \emptyset\}$
- **Non-example:** $\{M \mid M \text{ has 15 states}\}$
Properties

Definition
A *property of languages* is simply a set of languages. We say $L$ *satisfies* the property $\mathbb{P}$ if $L \in \mathbb{P}$.

Definition
For any property $\mathbb{P}$, define language $L_\mathbb{P}$ to consist of Turing Machines which accept a language in $\mathbb{P}$:

$$L_\mathbb{P} = \{ M \mid L(M) \in \mathbb{P} \}$$

Deciding $L_\mathbb{P}$: deciding if a language represented as a TM satisfies the property $\mathbb{P}$.

- **Example:** $\{ M \mid L(M) \text{ is infinite} \}$; $E_{TM} = \{ M \mid L(M) = \emptyset \}$
- **Non-example:** $\{ M \mid M \text{ has 15 states} \}$ ← This is a property of TMs, and not languages!
Trivial Properties

Definition
A property is *trivial* if either it is not satisfied by any r.e. language, or if it is satisfied by all r.e. languages.
Definition
A property is *trivial* if either it is not satisfied by any r.e. language, or if it is satisfied by all r.e. languages. Otherwise it is *non-trivial*.
Trivial Properties

Definition
A property is *trivial* if either it is not satisfied by any r.e. language, or if it is satisfied by all r.e. languages. Otherwise it is *non-trivial*.

Example
Some trivial properties:

- \( P_{\text{ALL}} = \text{set of all languages} \)
- \( P_{\text{R.E.}} = \text{set of all r.e. languages} \)
- \( \overline{P} \) where \( P \) is trivial
Trivial Properties

Definition
A property is *trivial* if either it is not satisfied by any r.e. language, or if it is satisfied by all r.e. languages. Otherwise it is *non-trivial*.

Example
Some trivial properties:

- $P_{\text{ALL}} = \text{set of all languages}$
- $P_{\text{R.E.}} = \text{set of all r.e. languages}$
- $\overline{P}$ where $P$ is trivial
- $P = \{L \mid L \text{ is recognized by a TM with an even number of states}\}$

Observation. For any trivial property $P$, $L \in P$ is decidable. (Why?) Then $L \in P = \Sigma^*$ or $L \in P = \emptyset$. 
Trivial Properties

Definition
A property is *trivial* if either it is not satisfied by any r.e. language, or if it is satisfied by all r.e. languages. Otherwise it is *non-trivial*.

Example
Some trivial properties:
- $P_{\text{ALL}} = \text{set of all languages}$
- $P_{\text{R.E.}} = \text{set of all r.e. languages}$
- $\overline{P}$ where $P$ is trivial
- $P = \{ L \mid L \text{ is recognized by a TM with an even number of states} \} = P_{\text{R.E.}}$
Definition
A property is *trivial* if either it is not satisfied by any r.e. language, or if it is satisfied by all r.e. languages. Otherwise it is *non-trivial*.

Example
Some trivial properties:
- $\mathcal{P}_{\text{ALL}} = \text{set of all languages}$
- $\mathcal{P}_{\text{R.E.}} = \text{set of all r.e. languages}$
- $\overline{\mathcal{P}}$ where $\mathcal{P}$ is trivial
- $\mathcal{P} = \{L \mid L \text{ is recognized by a TM with an even number of states}\} = \mathcal{P}_{\text{R.E.}}$

Observation. For any trivial property $\mathcal{P}$, $L_{\mathcal{P}}$ is decidable. (Why?)
Trivial Properties

Definition
A property is *trivial* if either it is not satisfied by any r.e. language, or if it is satisfied by all r.e. languages. Otherwise it is *non-trivial*.

Example
Some trivial properties:

- $\mathbb{P}_{\text{ALL}} =$ set of all languages
- $\mathbb{P}_{\text{R.E.}} =$ set of all r.e. languages
- $\overline{\mathbb{P}}$ where $\mathbb{P}$ is trivial
- $\mathbb{P} = \{ L \mid L \text{ is recognized by a TM with an even number of states} \} = \mathbb{P}_{\text{R.E.}}$

Observation. For any trivial property $\mathbb{P}$, $L_{\mathbb{P}}$ is decidable. (Why?) Then $L_{\mathbb{P}} = \Sigma^*$ or $L_{\mathbb{P}} = \emptyset$. 
Rice’s Theorem

Proposition

If $P$ is a non-trivial property, then $L_P$ is undecidable.
Rice’s Theorem

Proposition

If $\mathbb{P}$ is a non-trivial property, then $L_\mathbb{P}$ is undecidable.

- Thus $\{M \mid L(M) \in \mathbb{P}\}$ is not decidable (unless $\mathbb{P}$ is trivial)
Rice’s Theorem

Proposition

If $\mathcal{P}$ is a non-trivial property, then $L_{\mathcal{P}}$ is undecidable.

Thus $\{M \mid L(M) \in \mathcal{P}\}$ is not decidable (unless $\mathcal{P}$ is trivial)

We cannot algorithmically determine any interesting property of languages represented as Turing Machines!
Properties of TMs

Note. Properties of TMs, as opposed to those of languages they accept, may or may not be decidable.
Properties of TMs

Note. Properties of TMs, as opposed to those of languages they accept, may or may not be decidable.

Example

\[
\begin{align*}
\{\langle M \rangle \mid M \text{ has 193 states}\} & \quad \text{Decidable} \\
\{\langle M \rangle \mid M \text{ uses at most 32 tape cells on blank input}\} & \quad \text{Decidable} \\
\{\langle M \rangle \mid M \text{ halts on blank input}\} & \quad \text{Undecidable} \\
\{\langle M \rangle \mid \text{on input 0011 } M \text{ at some point writes the symbol } $ \text{ on its tape}\} & \quad \text{Undecidable}
\end{align*}
\]
Proof of Rice’s Theorem

Rice’s Theorem
If $\mathcal{P}$ is a non-trivial property, then $L_\mathcal{P}$ is undecidable.

Proof.
Proof of Rice’s Theorem

Rice’s Theorem
If $\mathcal{P}$ is a non-trivial property, then $L_{\mathcal{P}}$ is undecidable.

Proof.

▶ Suppose $\mathcal{P}$ non-trivial and $\emptyset \notin \mathcal{P}$. 
Proof of Rice’s Theorem

Rice’s Theorem
If $\mathbb{P}$ is a non-trivial property, then $L_{\mathbb{P}}$ is undecidable.

Proof.

- Suppose $\mathbb{P}$ non-trivial and $\emptyset \not\in \mathbb{P}$.
  - (If $\emptyset \in \mathbb{P}$, then in the following we will be showing $L_{\overline{\mathbb{P}}}$ is undecidable. Then $L_{\mathbb{P}} = \overline{L_{\overline{\mathbb{P}}}}$ is also undecidable.)
Proof of Rice’s Theorem

Rice’s Theorem
If $\mathcal{P}$ is a non-trivial property, then $L_\mathcal{P}$ is undecidable.

Proof.

- Suppose $\mathcal{P}$ non-trivial and $\emptyset \not\in \mathcal{P}$.
  - (If $\emptyset \in \mathcal{P}$, then in the following we will be showing $L_{\overline{\mathcal{P}}}$ is undecidable. Then $L_\mathcal{P} = \overline{L_{\overline{\mathcal{P}}}}$ is also undecidable.)

- Recall $L_\mathcal{P} = \{\langle M \rangle \mid L(M) \text{ satisfies } \mathcal{P}\}$. We’ll reduce $A_{TM}$ to $L_\mathcal{P}$. 
Proof of Rice’s Theorem

Rice’s Theorem
If \( \mathcal{P} \) is a non-trivial property, then \( L_{\mathcal{P}} \) is undecidable.

Proof.

- Suppose \( \mathcal{P} \) non-trivial and \( \emptyset \notin \mathcal{P} \).
  - (If \( \emptyset \in \mathcal{P} \), then in the following we will be showing \( L_{\mathcal{P}} \) is undecidable. Then \( L_{\mathcal{P}} = \overline{L_{\mathcal{P}}} \) is also undecidable.)
- Recall \( L_{\mathcal{P}} = \{ \langle M \rangle \mid L(M) \) satisfies \( \mathcal{P} \} \). We’ll reduce \( A_{TM} \) to \( L_{\mathcal{P}} \).
- Then, since \( A_{TM} \) is undecidable, \( L_{\mathcal{P}} \) is also undecidable.
Rice’s Theorem
If $\mathcal{P}$ is a non-trivial property, then $L_\mathcal{P}$ is undecidable.

Proof.

- Suppose $\mathcal{P}$ non-trivial and $\emptyset \not\in \mathcal{P}$.
  - (If $\emptyset \in \mathcal{P}$, then in the following we will be showing $L_\overline{\mathcal{P}}$ is undecidable. Then $L_\mathcal{P} = \overline{L_\overline{\mathcal{P}}}$ is also undecidable.)
- Recall $L_\mathcal{P} = \{ \langle M \rangle \mid L(M) \text{ satisfies } \mathcal{P} \}$. We’ll reduce $A_{\text{TM}}$ to $L_\mathcal{P}$.
- Then, since $A_{\text{TM}}$ is undecidable, $L_\mathcal{P}$ is also undecidable. \(\rightarrow\)
Proof of Rice’s Theorem

Proof (contd).

Since $\mathbb{P}$ is non-trivial, at least one r.e. language satisfies $\mathbb{P}$. 

Thus, $\langle M, w \rangle \in A_{tm}$ iff $N \in \mathbb{P}$. 

Proof of Rice’s Theorem

Proof (contd).

Since \( \mathbb{P} \) is non-trivial, at least one r.e. language satisfies \( \mathbb{P} \). i.e.,
\[ L(M_0) \in \mathbb{P} \]
for some TM \( M_0 \).
Proof (contd).

Since $\mathbb{P}$ is non-trivial, at least one r.e. language satisfies $\mathbb{P}$. i.e., $L(M_0) \in \mathbb{P}$ for some TM $M_0$.

Will show a reduction $f$ that maps an instance $\langle M, w \rangle$ for $A_{TM}$, to $N$ such that

- If $M$ accepts $w$ then $N$ accepts the same language as $M_0$.
  - Then $L(N) = L(M_0) \in \mathbb{P}$
- If $M$ does not accept $w$ then $N$ accepts $\emptyset$.
  - Then $L(N) = \emptyset \not\in \mathbb{P}$

Thus, $\langle M, w \rangle \in A_{TM}$ iff $N \in L(\mathbb{P})$.  

··→
Proof of Rice’s Theorem

Proof (contd).

Since \( \mathbb{P} \) is non-trivial, at least one r.e. language satisfies \( \mathbb{P} \). i.e.,
\( L(M_0) \in \mathbb{P} \) for some TM \( M_0 \).

Will show a reduction \( f \) that maps an instance \( \langle M, w \rangle \) for \( A_{TM} \), to \( N \) such that

- If \( M \) accepts \( w \) then \( N \) accepts the same language as \( M_0 \).
  - Then \( L(N) = L(M_0) \in \mathbb{P} \)
- If \( M \) does not accept \( w \) then \( N \) accepts \( \emptyset \).
  - Then \( L(N) = \emptyset \notin \mathbb{P} \)

Thus, \( \langle M, w \rangle \in A_{TM} \) iff \( N \in L_{\mathbb{P}} \).
Proof of Rice’s Theorem

Proof (contd).

Since $\mathbb{P}$ is non-trivial, at least one r.e. language satisfies $\mathbb{P}$. i.e., $L(M_0) \in \mathbb{P}$ for some TM $M_0$.

Will show a reduction $f$ that maps an instance $\langle M, w \rangle$ for $A_{TM}$, to $N$ such that

- If $M$ accepts $w$ then $N$ accepts the same language as $M_0$.
  - Then $L(N) = L(M_0) \in \mathbb{P}$
- If $M$ does not accept $w$ then $N$ accepts $\emptyset$.
  - Then $L(N) = \emptyset \notin \mathbb{P}$

Thus, $\langle M, w \rangle \in A_{TM}$ iff $N \in L_{\mathbb{P}}$. ..→
Proof of Rice’s Theorem

Proof (contd).

The reduction $f$ maps $\langle M, w \rangle$ to $N$, where $N$ is a TM that behaves as follows:

On input $x$
- Ignore the input and run $M$ on $w$
- If $M$ does not accept (or doesn’t halt)
  - then do not accept $x$ (or do not halt)
- If $M$ does accept $w$
  - then run $M_0$ on $x$ and accept $x$ iff $M_0$ does.

Notice that indeed if $M$ accepts $w$ then $L(N) = L(M_0)$. Otherwise $L(N) = \emptyset$. □
Rice’s Theorem

Recap

Every non-trivial property of r.e. languages is undecidable
Rice’s Theorem

Recap

Every non-trivial property of r.e. languages is undecidable
  ▶ Rice’s theorem says nothing about properties of Turing machines
Rice’s Theorem

Recap

Every non-trivial property of r.e. languages is undecidable

- Rice’s theorem says nothing about properties of Turing machines
- Rice’s theorem says nothing about whether a property of languages is recursively enumerable or not.
Big Picture . . . again

Languages
- $L_d$, $\overline{A_{TM}}$, $E_{TM}$

Recursively Enumerable
- $A_{TM}$, $\overline{E_{TM}}$, HALT

Decidable
- $L_{anbncn}$

CFL
- $L_{0n1n}$

Regular
Big Picture . . . again

Languages

Recursively Enumerable

Decidable

CFL

Regular

Languages

$L_d, \overline{A_{TM}}, E_{TM}$

Recursively Enumerable

$A_{TM}, \overline{E_{TM}}, \text{HALT}$

Decidable

$L_{anbncn}$

CFL

$L_{0n1n}$

Regular

“almost all” properties!