CSE 135: Introduction to Theory of Computation
Decidability and Recognizability

Sungjin Im

University of California, Merced

04-14 and 4-16-2015
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

3

4
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)

- 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
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.
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

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.
But not all languages are decidable! We will show:

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

However, $A_{\text{TM}}$ is Turing-recognizable!

Proposition: There are languages which are recognizable, but not decidable.
But not all languages are decidable! We will show:

- $A_{TM} = \{ \langle M, w \rangle | M \text{ is a TM and } M \text{ accepts } w \}$ 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 $q_{\text{rej}}$)

But $U$ does not decide $A_{TM}$: If $M$ rejects $w$ by not halting (does not halt on $w$), $U$ rejects $\langle M, w \rangle$ by not halting (does not halt on $\langle M, w \rangle$).

Indeed (as we shall see) no TM decides $A_{TM}$.
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.,

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

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 $A_{TM}$: If $M$ rejects $w$ by not halting (does not halt on $w$), $U$ rejects $\langle M, w \rangle$ by not halting (does not halt on $\langle M, w \rangle$).
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 $A_{TM}$: If $M$ rejects $w$ by not halting (does not halt on $w$), $U$ rejects $\langle M, w \rangle$ by not halting (does not halt on $\langle M, w \rangle$). Indeed (as we shall see) no TM decides $A_{TM}$.
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 \( x \not\in L \), 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 \( \bar{L} \) are recognizable, then \( L \) is decidable

Proof.

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

- On input \( x \), simulate \( P_L \) and \( P_{\bar{L}} \) on input \( x \). Whether \( x \in L \) or \( x \notin L \), one of \( P_L \) and \( P_{\bar{L}} \) 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 \( P_{\bar{L}} \) on input \( x \) until either \( P_L \) or \( P_{\bar{L}} \) 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 $i = 1, 2, 3, ...$
    simulate $P_L$ on input $x$ for $i$ steps
    simulate $P_{-L}$ on input $x$ for $i$ steps
    if either simulation accepts, break
if $P_L$ accepted, accept $x$ (and halt)
if $P_{-L}$ accepted, reject $x$ (and halt)
Deciding vs. Recognizing

Proof (contd).

In more detail, $P$ works as follows:

On input $x$
for $i = 1, 2, 3, ...$

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:

- $A_{\text{TM}}$ is undecidable (will learn soon)
- But it is recognizable
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.
So far:

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

\( \exists \) 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 the \( i \)th binary string accepts the \( j \)th binary string.
Proposition

$L_d$ is not recursively enumerable.

Proof.

Recall that,

- Inputs are strings over \{0, 1\}
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
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 \(j \in L(i)\).

\[
\begin{array}{c|cccccccc}
\text{TM} & 1 & 2 & 3 & 4 & 5 & 6 & 7 & \ldots \\
\hline
3 & Y & N & Y & N & Y & Y & Y & Y \\
5 & N & Y & N & Y & Y & N & N & N \\
6 & N & N & Y & N & Y & N & Y & Y \\
\end{array}
\]

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 \(j /\in L(j)\) – 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 \(j \in L(i)\).

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

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

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

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. □
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 $L_d$?
Accept 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

- $L_d$ 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

\[ A_{TM} \text{ is r.e. but not decidable.} \]
The Universal Language

Proposition

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

Proof.

We have already seen that $A_{TM}$ is r.e.
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 \( i \)

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

Languages

Recursively Enumerable

Decidable

CFL

Regular

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

$A_{TM}$

$L_{anbncn}$

$L_{0^n1^n}$
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.
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 a 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 $\langle w, w \rangle \in A_{TM}$. “
Proposition

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

Reductions schematically

\[ w \xrightarrow{} \]
Reductions schematically
Schematic View

Reductions schematically

$w \rightarrow \text{Reduction } f \rightarrow f(w) \rightarrow \text{Algorithm for Problem 2}$

$\rightarrow \text{yes}$

$\rightarrow \text{no}$
Algorithm for Problem 1

Reduction $f$

Algorithm for Problem 2

$w \xrightarrow{f(w)}$ yes

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

Proof.
We will reduce \( A_{TM} \) to \( \text{HALT} \). 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 $A_{TM}$ 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. □
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
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 \( A_{TM} \).
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 $A_{\text{TM}}$. But, $A_{\text{TM}}$ is undecidable, which gives us the contradiction. □
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.
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 $A \leq_m B$. 
Convention

In this course, we will drop the adjective “mapping” or “many-one”, and simply talk about reductions and reducibility.
Proposition

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

\[ \text{On input } w \]
\[ \quad \text{Compute } f(w) \]
\[ \quad \text{Run } M_B \text{ on } f(w) \]
\[ \quad \text{Accept if } M_B \text{ does and reject if } M_B \text{ rejects} \]
\[ \square \]
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.
Reducions 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.
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 $f(w)$. □
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.