Computational Models – Lecture 8

- Decidability of CFGs Questions (cont.)
- Universal Machines
- Undecidability of the Halting Problem
Decidability of CFG Emptiness

Define $E_{CFG} = \{ \langle G \rangle \mid G$ is a CFG and $L(G) = \emptyset \}$
Decidability of CFG Emptiness

Define $E_{CFG} = \{ \langle G \rangle | G$ is a CFG and $L(G) = \emptyset \}$

Theorem: $E_{CFG}$ is a decidable language.
Decidability of CFG Emptiness

Define $E_{CFG} = \{ \langle G \rangle \mid G$ is a CFG and $L(G) = \emptyset \}$

**Theorem:** $E_{CFG}$ is a decidable language.

Possible approaches for a proof:
Decidability of CFG Emptiness

Define \( E_{\text{CFG}} = \{ \langle G \rangle \mid G \text{ is a CFG and } L(G) = \emptyset \} \)

**Theorem:** \( E_{\text{CFG}} \) is a decidable language.

Possible approaches for a proof:

**Bad Idea:** We know how to test whether \( w \in L(G) \) for any string \( w \), so just try it for each \( w \). (criticize this...)

**Better Idea:** Can a particular variable generate a string of terminals?
Decidability of CFG Emptiness

Define \( E_{\text{CFG}} = \{ \langle G \rangle \mid G \text{ is a CFG and } L(G) = \emptyset \} \)

**Theorem:** \( E_{\text{CFG}} \) is a decidable language.

Possible approaches for a proof:

**Bad Idea:** We know how to test whether \( w \in L(G) \) for any string \( w \), so just try it for each \( w \). (criticize this...)

**Good Idea:** Can the start variable generate a string of terminals?
Decidability of CFG Emptiness

Define $E_{CFG} = \{ \langle G \rangle \mid G \text{ is a CFG and } L(G) = \emptyset \}$

**Theorem:** $E_{CFG}$ is a decidable language.

Possible approaches for a proof:

**Bad Idea:** We know how to test whether $w \in L(G)$ for any string $w$, so just try it for each $w$. (criticize this...)

**Good Idea:** Can the start variable generate a string of terminals?

**Better Idea:** Can a particular variable generate a string of terminals?
Decidability of CFG Emptiness (2)

Theorem: $E_{CFG}$ is a decidable language.

TM description: On input $\langle G \rangle$ where $G$ is a CFG,
Decidability of CFG Emptiness (2)

**Theorem:** $E_{CFG}$ is a decidable language.

TM description: On input $\langle G \rangle$ where $G$ is a CFG,

1. Mark all terminal symbols in $G$. 
Decidability of CFG Emptiness (2)

Theorem: \( E_{\text{CFG}} \) is a decidable language.

TM description: On input \( \langle G \rangle \) where \( G \) is a CFG,

1. Mark all terminal symbols in \( G \).
2. Repeat until no new variables become marked.
Decidability of CFG Emptiness (2)

**Theorem:** $E_{CFG}$ is a decidable language.

TM description: On input $\langle G \rangle$ where $G$ is a CFG,

1. Mark all terminal symbols in $G$.
2. Repeat until no new variables become marked.
3. Mark any $A$ where $A \rightarrow U_1 U_2 \ldots U_k$ and all $U_i$ have already been marked.
Decidability of CFG Emptiness (2)

**Theorem:** $E_{CFG}$ is a decidable language.

TM description: On input $\langle G \rangle$ where $G$ is a CFG,

1. Mark all terminal symbols in $G$.
2. Repeat until no new variables become marked.
3. Mark any $A$ where $A \rightarrow U_1 U_2 \ldots U_k$ and all $U_i$ have already been marked.
4. If start symbol marked, reject, else accept.
When Are Two CFGs equivalent?

\[ EQ_{CFG} = \{ \langle G, H \rangle \mid G, H \text{ are CFGs and } L(G) = L(H) \} \]
When Are Two CFGs equivalent?

\[ EQ_{CFG} = \{ \langle G, H \rangle \mid G, H \text{ are CFGs and } L(G) = L(H) \} \]

Hey, we did this already for \( EQ_{DFA} \)!

We constructed \( C \) from \( A \) and \( B \):

\[ L(C) = \left( L(A) \cap \overline{L(B)} \right) \cup \left( \overline{L(A)} \cap L(B) \right) . \]

and tested whether \( L(C) \) is empty.
When Are Two CFGs equivalent?

\[ EQ_{\text{CFG}} = \{ \langle G, H \rangle \mid G, H \text{ are CFGs and } L(G) = L(H) \} \]

Hey, we did this already for \( EQ_{\text{DFA}} \)!

We constructed \( C \) from \( A \) and \( B \):

\[ L(C) = \left( L(A) \cap \overline{L(B)} \right) \cup \left( \overline{L(A)} \cap L(B) \right) . \]

and tested whether \( L(C) \) is empty.

Stop! Danger! Abyss ahead!
When Are Two CFGs equivalent?

This approach was fine for DFAs, but not for CFLs!
When Are Two CFGs equivalent?

This approach was fine for DFAs, but not for CFLs!

The class of context-free languages is not closed under complementation or intersection.
When Are Two CFGs equivalent?

This approach was fine for DFAs, but not for CFLs!

The class of context-free languages is not closed under complementation or intersection.

Fact: $E_{CFG}$ is not a decidable language.
When Are Two CFGs equivalent?

This approach was fine for DFAs, but not for CFLs!

The class of context-free languages is not closed under complementation or intersection.

Fact: \( E_{\text{CFG}} \) is not a decidable language.

We are not prepared to prove this remarkable fact (yet).
Thm: Every CFL is Decidable

Bad Proof Idea:

...
Thm: Every CFL is Decidable

Bad Proof Idea:

convert a PDA for $L$ directly into a TM
Thm: Every CFL is Decidable

**Bad Proof Idea:**
- convert a PDA for $L$ directly into a TM
- not hard (use 2nd tape to simulate stack)
Thm: Every CFL is Decidable

Bad Proof Idea:
- convert a PDA for $L$ directly into a TM
- not hard (use 2nd tape to simulate stack)
- non-deter. PDA yields non-deter. TM
Thm: Every CFL is Decidable

Bad Proof Idea:
- convert a PDA for $L$ directly into a TM
- not hard (use 2nd tape to simulate stack)
- non-deter. PDA yields non-deter. TM
- no worries – non-deterministic TM same as deterministic TM
Thm: Every CFL is Decidable

Bad Proof Idea:
- convert a PDA for $L$ directly into a TM
- not hard (use 2nd tape to simulate stack)
- non-deter. PDA yields non-deter. TM
- no worries – non-deterministic TM same as deterministic TM

Problem:
Thm: Every CFL is Decidable

- **Bad Proof Idea:**
  - convert a PDA for $L$ directly into a TM
  - not hard (use 2nd tape to simulate stack)
  - non-deter. PDA yields non-deter. TM
  - no worries – non-deterministic TM same as deterministic TM

- **Problem:**
  - on some $w \notin L$ some branch of PDA may run forever
Thm: Every CFL is Decidable

Bad Proof Idea:

- Convert a PDA for $L$ directly into a TM
- Not hard (use 2nd tape to simulate stack)
- Non-deterministic PDA yields non-deterministic TM
- No worries – non-deterministic TM same as deterministic TM

Problem:

- On some $w \notin L$ some branch of PDA may run forever
- Some branch of non-deterministic TM might run forever
Thm: Every CFL is Decidable

- **Bad Proof Idea:**
  - convert a PDA for \( L \) directly into a TM
  - not hard (use 2nd tape to simulate stack)
  - non-deter. PDA yields non-deter. TM
  - no worries – non-deterministic TM same as deterministic TM

- **Problem:**
  - on some \( w \notin L \) some branch of PDA may run forever
  - some branch of non-deterministic TM might run forever
  - deterministic TM may loop on \( w \notin L \)
Thm: Every CFL is Decidable

Bad Proof Idea:
- convert a PDA for $L$ directly into a TM
- not hard (use 2nd tape to simulate stack)
- non-deterministic PDA yields non-deterministic TM
- no worries – non-deterministic TM same as deterministic TM

Problem:
- on some $w \notin L$ some branch of PDA may run forever
- some branch of non-deterministic TM might run forever
- deterministic TM may loop on $w \notin L$
- deterministic TM accepts $L$, but does not decide!
Last Word on Context-Free Languages

Reminder: The language $A_{CFG}$ is decidable.

$$A_{CFG} = \{ \langle G, w \rangle \mid \text{string } w \text{ is generated by CFG } G \}$$
Reminder: The language $A_{CFG}$ is decidable.

$$A_{CFG} = \{ \langle G, w \rangle \mid \text{string } w \text{ is generated by CFG } G \}$$

Let

- $S$ be the TM that decides $A_{CFG}$, and
- $G$ be a CFG for $L$. 
Last Word on Context-Free Languages

Reminder: The language $A_{\text{CFG}}$ is decidable.

$$A_{\text{CFG}} = \{ \langle G, w \rangle \mid \text{string } w \text{ is generated by CFG } G \}$$

Let

- $S$ be the TM that decides $A_{\text{CFG}}$, and
- $G$ be a CFG for $L$.

On input $w$

1. Run TM $S$ on input $\langle G, w \rangle$
2. Accept if $S$ accepts, otherwise reject.
Updated View of the World of Languages

- regular
- context free
- decidable
- enumerable
Univeral Turing Machines

We now define the universal Turing machine, $U$. On input $\langle M, w \rangle$, where $M$ is a TM and $w$ a string...
Univeral Turing Machines

We now define the universal Turing machine, $U$. On input $\langle M, w \rangle$, where $M$ is a TM and $w$ a string

1. Checks that $\langle M, w \rangle$ is a proper encoding of a TM, followed by a string from $\Sigma^*$. 
Universal Turing Machines

We now define the universal Turing machine, $U$. On input $\langle M, w \rangle$, where $M$ is a TM and $w$ a string

1. Checks that $\langle M, w \rangle$ is a proper encoding of a TM, followed by a string from $\Sigma^*$.
2. Simulates $M$ on input $w$. 
Universal Turing Machines

We now define the **universal Turing machine**, \( U \).

On input \( \langle M, w \rangle \), where \( M \) is a TM and \( w \) a string

1. Checks that \( \langle M, w \rangle \) is a proper encoding of a TM, followed by a string from \( \Sigma^* \).
2. Simulates \( M \) on input \( w \).
3. If \( M \) on input \( w \) enters its accept state, \( U \) accept, and if \( M \) on input \( w \) ever enters its reject state, \( U \) reject.
Univeral Turing Machines

We now define the universal Turing machine, $U$. On input $\langle M, w \rangle$, where $M$ is a TM and $w$ a string

1. Checks that $\langle M, w \rangle$ is a proper encoding of a TM, followed by a string from $\Sigma^*$.
2. Simulates $M$ on input $w$.
3. If $M$ on input $w$ enters its accept state, $U$ accept, and if $M$ on input $w$ ever enters its reject state, $U$ reject.

Notice that as a consequence, if $M$ on input $w$ enters an infinite loop, so does $U$ on input $\langle M, w \rangle$. 
Universal Turing Machines (2)

The universal machine $U$ obviously has a fixed number of states (100 should do).
Universal Turing Machines (2)

- The universal machine $U$ obviously has a **fixed number** of states (100 should do).
- Despite this, it can simulate machines $M$ with many more states.
Universal Turing Machines (2)

- The universal machine $U$ obviously has a fixed number of states (100 should do).
- Despite this, it can simulate machines $M$ with many more states.
- Most of you have seen a universal machine, and have even used one!
Universal Turing Machines (2)

- The universal machine $U$ obviously has a fixed number of states (100 should do).
- Despite this, it can simulate machines $M$ with many more states.
- Most of you have seen a universal machine, and have even used one!
- For example, Dr. Scheme (interpreter) is a universal Scheme machine.
Universal Turing Machines (2)

- The universal machine $U$ obviously has a fixed number of states (100 should do).
- Despite this, it can simulate machines $M$ with many more states.
- Most of you have seen a universal machine, and have even used one!
- For example, Dr. Scheme (interpreter) is a universal Scheme machine.
- It accepts a two part input: “Above the line” – the program (parallel to $\langle M \rangle$), and “below the line” the input to run it on (parallel to $w$).
Universal Turing Machines (2)

- The universal machine $U$ obviously has a fixed number of states (100 should do).
- Despite this, it can simulate machines $M$ with many more states.
- Most of you have seen a universal machine, and have even used one!
- For example, Dr. Scheme (interpreter) is a universal Scheme machine.
- It accepts a two part input: “Above the line” – the program (parallel to $\langle M \rangle$), and “below the line” the input to run it on (parallel to $w$).
- Universal machines inspired the development of stored-program computers in the 40s and 50s.
Halting Problem

One of the most philosophically important theorems of the theory of computation.
Halting Problem

One of the most philosophically important theorems of the theory of computation.

Computers (and computation) are not omnipotent – they are limited in a very fundamental way.
Halting Problem

One of the most philosophically important theorems of the theory of computation.

Computers (and computation) are not omnipotent – they are limited in a very fundamental way.

Many common problems are unsolvable, e.g.

- Does a program sort an array of integers?
- Problem is well defined: Both program and specification are precise mathematical objects.
- Hey, proving program $\cong$ specification should be just like proving that triangle 1 $\cong$ triangle 2 ...
Halting Problem

One of the most philosophically important theorems of the theory of computation.

Computers (and computation) are not omnipotent – they are limited in a very fundamental way.

Many common problems are unsolvable, e.g.

- Does a program sort an array of integers?
- Problem is well defined: Both program and specification are precise mathematical objects.
- Hey, proving program $\cong$ specification should be just like proving that triangle 1 $\cong$ triangle 2 ...
- Well, this is not the case!
Halting Problem

One of the most philosophically important theorems of the theory of computation.

Computers (and computation) are not omnipotent – they are limited in a very fundamental way. Many common problems are unsolvable, e.g.

- Does a program sort an array of integers?
- Problem is well defined: Both program and specification are precise mathematical objects.
- Hey, proving program $\equiv$ specification should be just like proving that triangle 1 $\equiv$ triangle 2 . . .
- Well, this is not the case!
Halting Problem

One of the most philosophically important theorems of the theory of computation.

Computers (and computation) are not omnipotent – they are limited in a very fundamental way.

Many common problems are unsolvable, e.g.

- Does a program sort an array of integers?
- Problem is well defined: Both program and specification are precise mathematical objects.
- Hey, proving program $\approx$ specification should be just like proving that triangle 1 $\approx$ triangle 2 . . .
- Well, this is not the case!
Halting Problem

One of the most philosophically important theorems of the theory of computation.

Computers (and computation) are not omnipotent – they are limited in a very fundamental way.

Many common problems are unsolvable, e.g.

- Does a program sort an array of integers?
- Problem is well defined: Both program and specification are precise mathematical objects.
- Hey, proving program $\equiv$ specification should be just like proving that triangle 1 $\equiv$ triangle 2 . . .
- Well, this is not the case!
Accepting Problem

Does a Turing machine accept a string?

\[ A_{TM} = \{ \langle M, w \rangle | M \text{ is a TM that accepts } w \} \]

Theorem: \( A_{TM} \) is undecidable.
Accepting Problem

Does a Turing machine accept a string?

\[ A_{TM} = \{ \langle M, w \rangle | M \text{ is a TM that accepts } w \} \]

**Theorem:** \( A_{TM} \) is undecidable.

Recall that the corresponding languages for DFAs, NFAs, and CFGs, namely \( A_{DFA} \), \( A_{NFA} \), and \( A_{CFG} \), are decidable.
The Acceptance Problem

\[ A_{TM} = \{ \langle M, w \rangle | M \text{ is a TM that accepts } w \} \]
The Acceptance Problem

\[ A_{\text{TM}} = \{ \langle M, w \rangle | M \text{ is a TM that accepts } w \} \]

Before approaching the proof of undecidability, we first prove

**Theorem:** \( A_{\text{TM}} \) is enumerable.
The Acceptance Problem

$$A_{TM} = \{ \langle M, w \rangle | M \text{ is a TM that accepts } w \}$$

Before approaching the proof of undecidability, we first prove

**Theorem:** $A_{TM}$ is enumerable.

**Proof:** The universal machine accepts $A_{TM}$. ♣
The Acceptance Problem

\[ A_{TM} = \{ \langle M, w \rangle | M \text{ is a TM that accepts } w \} \]

We prove \( A_{TM} \) is undecidable by diagonalization.

But first, a short “diagonalization reminder”.

Comparing Sizes of Sets

Suppose $A$ and $B$ are two sets, and we wish to compare their sizes.
Comparing Sizes of Sets

Suppose $A$ and $B$ are two sets, and we wish to compare their sizes.

If both $A$ and $B$ are finite, we can count how many elements each of them has, and compare the numbers.
Comparing Sizes of Sets

Suppose $A$ and $B$ are two sets, and we wish to compare their sizes.

If both $A$ and $B$ are finite, we can count how many elements each of them has, and compare the numbers.

This method does not generalize to infinite sets.
Comparing Sizes of Sets (2)

Alternatively, we can pair the elements of \( A \) and \( B \). If they pair perfectly, they have equal sizes.
Comparing Sizes of Sets (2)

Alternatively, we can pair the elements of $A$ and $B$. If they pair perfectly, they have equal sizes.
Comparing Sizes of Sets (2)

Alternatively, we can pair the elements of $A$ and $B$. If they pair perfectly, they have equal sizes.
Correspondence

**Question:** What does it mean to say that two infinite sets are the *same size*?

Answered by Georg Cantor in 1873: Pair them off.
Correspondence

**Question:** What does it mean to say that two infinite sets are the *same size*?

Answered by Georg Cantor in 1873: Pair them off.

A map \( f : A \to B \) is a *correspondence* if \( f \) satisfies

- \( f \) one-to-one: if \( a_1 \neq a_2 \) then \( f(a_1) \neq f(a_2) \).
- \( f \) onto: for every \( b \in B \), there is an \( a \in A \) such that \( f(a) = b \).
Correspondence

**Question:** What does it mean to say that two infinite sets are the *same size*?

Answered by **Georg Cantor** in 1873: Pair them off.

A map $f : A \rightarrow B$ is a *correspondence* if $f$ satisfies

- $f$ one-to-one: if $a_1 \neq a_2$ then $f(a_1) \neq f(a_2)$.
- $f$ onto: for every $b \in B$, there is an $a \in A$ such that $f(a) = b$.

**Question:** What does it mean to say that sets $A$ and $B$ are the *same size*?
Correspondence

**Question:** What does it mean to say that two infinite sets are the *same size*?

Answered by **Georg Cantor** in 1873: Pair them off.

A map $f : A \to B$ is a *correspondence* if $f$ satisfies

- $f$ one-to-one: if $a_1 \neq a_2$ then $f(a_1) \neq f(a_2)$.
- $f$ onto: for every $b \in B$, there is an $a \in A$ such that $f(a) = b$.

**Question:** What does it mean to say that sets $A$ and $B$ are the *same size*?

**Answer:** $A$ and $B$ are the *same size* if there is a correspondence from $A$ to $B$. 
Correspondence (2)

**Question:** In a crowded room, how can we tell if there are more people than chairs, or more chairs than people?

**Answer:** Establish a correspondence: ask everyone to sit down.

(c.f., Mathematician’s trick for counting a herd of cows . . .)
Correspondence

Claim: The set $\mathcal{N}$ of natural numbers has the same size as the set $\mathcal{E}$ of even numbers

Proof: Let $f(i) = 2i$.

Remark: The set $\mathcal{E}$ is a proper subset of the set $\mathcal{N}$, yet they are the same size!
Countable Sets

Definition: A set $A$ is countable if

- either $A$ is finite, or
- $A$ has the same size as $\mathbb{N}$, the natural numbers.

We have just seen that $\mathcal{E}$ is countable.

A countable set is sometimes said to have size $\aleph_0$. 
Countable Sets

**Definition:** A set $A$ is *countable* if

- either $A$ is finite, or
- $A$ has the same size as $\mathbb{N}$, the natural numbers.

We have just seen that $E$ is countable.

A countable set is sometimes said to have size $\aleph_0$.

**Claim:** The set $\mathbb{Z}$ of integers is countable.

**Proof:** Define $f : \mathbb{N} \rightarrow \mathbb{Z}$ by

$$f(i) = \begin{cases} 
  i/2 & \text{if } i \text{ is even} \\
  -(\lfloor i/2 \rfloor + 1) & \text{if } i \text{ is odd}
\end{cases}$$
Pop Quiz

In Heaven, there is a hotel with a countable number of rooms.
Pop Quiz

In Heaven, there is a hotel with a countable number of rooms.

One day, the society of Prophets, Oracles, and AI Researchers holds a 3-day convention that books every room in the hotel.

Answer: Ask the guest in room $i$ to move to room $i + 1$, and put the newcomer in room 1.
Pop Quiz

In Heaven, there is a hotel with a countable number of rooms.

One day, the society of Prophets, Oracles, and AI Researchers holds a 3-day convention that books every room in the hotel.

Then one more guest arrives, claiming he invented Lisp, and angrily demanding a room.
Pop Quiz

In Heaven, there is a hotel with a countable number of rooms.

One day, the society of Prophets, Oracles, and AI Researchers holds a 3-day convention that books every room in the hotel.

Then one more guest arrives, claiming he invented Lisp, and angrily demanding a room.

You are the manager. What do you do?
Pop Quiz

In Heaven, there is a hotel with a countable number of rooms.

One day, the society of Prophets, Oracles, and AI Researchers holds a 3-day convention that books every room in the hotel.

Then one more guest arrives, claiming he invented Lisp, and angrily demanding a room.

You are the manager. What do you do?

Answer: Ask the guest in room $i$ to move to room $i + 1$, and put the newcomer in room 1.
Pop Quiz #2

Then a countable number of guests arrive, all angrily demanding rooms. *What a noise!*
Pop Quiz #2

Then a countable number of guests arrive, all angrily demanding rooms. (What a noise!)

Now what do you do?
Pop Quiz #2

Then a countable number of guests arrive, all angrily demanding rooms. *(What a noise!)*

Now what do you do?

**Answer:** Ask the guest in room $i$ to move to room $2i$, and put the newcomers in the odd-numbered rooms.
Rational Numbers

Let

\[ \mathbb{Q} = \left\{ \frac{m}{n} \mid m, n \in \mathbb{N} \right\} \]

Theorem: \( \mathbb{Q} \) is countable.

This claim may seem counterintuitive.

Idea
Rational Numbers

Let

\[ Q = \left\{ \frac{m}{n} \mid m, n \in \mathbb{N} \right\} \]

**Theorem:** \( Q \) is countable.

This claim may seem counterintuitive.

Idea

- list \( Q \) as 2-dim array
Rational Numbers

Let

\[ Q = \left\{ \frac{m}{n} \mid m, n \in \mathbb{N} \right\} \]

**Theorem:** \( Q \) is countable.

This claim may seem counterintuitive.

Idea

- list \( Q \) as 2-dim array
- begin counting with the first row . . .
Rational Numbers

Let

\[ Q = \left\{ \frac{m}{n} \mid m, n \in \mathbb{N} \right\} \]

**Theorem:** \( Q \) is countable.

This claim may seem counterintuitive.

Idea

- list \( Q \) as 2-dim array
- begin counting with the first row . . .

Why doesn’t this work?
Enumerate numbers along northeast and diagonals, skipping duplicates. Does this mean that every infinite set is countable?
Rational Numbers (2)

Enumerate numbers along northeast and diagonals, skipping duplicates.
Enumerate numbers along northeast and diagonals, skipping duplicates.

Does this mean that every infinite set is countable?
The Real Numbers

Every *real number* has a decimal representation. For example, \( \pi = 3.1415926 \ldots \), \( \sqrt{2} = 1.4142136 \ldots \), and \( 0 = 0.0000000 \ldots \).
The Real Numbers

Every *real number* has a decimal representation. For example, $\pi = 3.1415926 \ldots$, $\sqrt{2} = 1.4142136 \ldots$, and $0 = 0.0000000 \ldots$

Let $\mathbb{R}$ be the set of real numbers.
The Real Numbers

Every *real number* has a decimal representation. For example, \( \pi = 3.1415926 \ldots \), \( \sqrt{2} = 1.4142136 \ldots \), and \( 0 = 0.0000000 \ldots \).

Let \( \mathbb{R} \) be the set of real numbers.

**Theorem:** \( \mathbb{R} \) is uncountable.
The Real Numbers

Every \textit{real number} has a decimal representation. For example, \( \pi = 3.1415926 \ldots \), \( \sqrt{2} = 1.4142136 \ldots \), and \( 0 = 0.0000000 \ldots \).

Let \( \mathcal{R} \) be the set of real numbers.

\textbf{Theorem:} \( \mathcal{R} \) is uncountable.

\( \mathcal{R} \) is sometimes said to have size \( \aleph_1 \).
The Real Numbers

Every real number has a decimal representation. For example, \( \pi = 3.1415926 \ldots \), \( \sqrt{2} = 1.4142136 \ldots \), and \( 0 = 0.0000000 \ldots \).

Let \( \mathcal{R} \) be the set of real numbers.

**Theorem:** \( \mathcal{R} \) is uncountable.

\( \mathcal{R} \) is sometimes said to have size \( \aleph_1 \).

- This is Cantor’s historic proof, which
- introduced the *diagonalization* method.
The Real Numbers

Assume there is a correspondence between $\mathbb{N}$ and $\mathbb{R}$. Write it down:

<table>
<thead>
<tr>
<th>$n$</th>
<th>$f(n)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3.14159...</td>
</tr>
<tr>
<td>2</td>
<td>55.55555...</td>
</tr>
<tr>
<td>3</td>
<td>40.18642...</td>
</tr>
<tr>
<td>4</td>
<td>15.20601...</td>
</tr>
</tbody>
</table>

We now show that there is a number $x$ not in this list.
Diagonalization

Pick \(0 \leq x \leq 1\), so its significant digits follow decimal point. Will ensure \(x \neq f(n)\) for all \(n\).

<table>
<thead>
<tr>
<th>(n)</th>
<th>(f(n))</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3.14159...</td>
</tr>
<tr>
<td>2</td>
<td>55.55555...</td>
</tr>
<tr>
<td>3</td>
<td>40.18643...</td>
</tr>
<tr>
<td>4</td>
<td>15.20607...</td>
</tr>
</tbody>
</table>
Diagonalization

<table>
<thead>
<tr>
<th>$n$</th>
<th>$f(n)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$3.14159\ldots$</td>
</tr>
<tr>
<td>2</td>
<td>$55.55555\ldots$</td>
</tr>
<tr>
<td>3</td>
<td>$40.18643\ldots$</td>
</tr>
<tr>
<td>4</td>
<td>$15.20607\ldots$</td>
</tr>
</tbody>
</table>

- First fractional digit of $f(1)$ is 1, so pick first fractional digit of $x$ to be something else (say, 2).
- Second fractional digit of $f(2)$ is 5, so pick second fractional digit of $x$ to be something else (say, 6).
- and so on . . .
- $x = 0.2691\ldots$
Diagonalization

A similar proof shows there are languages that are not enumerable.
Diagonalization

A similar proof shows there are languages that are not enumerable.

- the set of Turing machines is countable, but
Diagonalization

A similar proof shows there are languages that are not enumerable.

- the set of Turing machines is countable, but
- the set of languages is uncountable!
Diagonalization

A similar proof shows there are languages that are not enumerable.

- the set of Turing machines is countable, but
- the set of languages is uncountable!

Ergo,
Diagonalization

A similar proof shows there are languages that are not enumerable.

- the set of Turing machines is countable, but
- the set of languages is uncountable!

Ergo,

- there exist languages that are not enumerable (why?)
Diagonalization

A similar proof shows there are languages that are not enumerable.

- the set of Turing machines is countable, but
- the set of languages is uncountable!

Ergo,
- there exist languages that are not enumerable (why?)
- indeed, “most” languages are not enumerable (explain)
∃ Countably Many Turing Machines

Claim: The set of strings, $\Sigma^*$, is countable.
∃ Countably Many Turing Machines

**Claim:** The set of strings, $\Sigma^*$, is countable.

**Proof:** List strings of length 0, then length 1, then 2, and so on. This exhausts all of $\Sigma^*$. The union of countably many finite sets is countable.
∃ Countably Many Turing Machines (2)

**Claim:** The set of all Turing machines is countable.
∃ Countably Many Turing Machines (2)

Claim: The set of all Turing machines is countable.

Proof: Each TM $M$ has an encoding as a string $⟨M⟩$. Therefore there is a one-to-one mapping from the set of all TMs into (but not onto) $Σ^*$. Since $Σ^*$ is countable, so is the set of all TMs.
The Set of All Languages is Uncountable

Let $\mathcal{B}$ be the set of infinite binary sequences.

**Claim:** $\mathcal{B}$ is uncountable.
The Set of All Languages is Uncountable

Let $B$ be the set of of infinite binary sequences.

Claim: $B$ is uncountable.

Proof: Diagonalization argument, essentially identical to the proof that $\mathcal{R}$ is uncountable.

(additional helpful clue: think of binary sequence as binary expansion!)
The Set of Languages is Uncountable (2)

Let $\mathcal{L}$ be the set of all languages over alphabet $\Sigma$. Recall $\mathcal{B}$ is the set of infinite binary sequences. We give a correspondence

$$\chi : \mathcal{L} \rightarrow \mathcal{B}$$

called the language’s *characteristic sequence*. 
The Set of Languages is Uncountable (2)

Let $\mathcal{L}$ be the set of all languages over alphabet $\Sigma$. Recall $\mathcal{B}$ is the set of infinite binary sequences. We give a correspondence

$$\chi : \mathcal{L} \rightarrow \mathcal{B}$$

called the language’s characteristic sequence.

Let $\Sigma^* = \{s_1, s_2, s_3, \ldots\}$ (in lexicographic order).
The Set of Languages is Uncountable (2)

Let $\mathcal{L}$ be the set of all languages over alphabet $\Sigma$. Recall $\mathcal{B}$ is the set of infinite binary sequences. We give a correspondence

$$\chi : \mathcal{L} \rightarrow \mathcal{B}$$

called the language’s characteristic sequence.

- Let $\Sigma^* = \{s_1, s_2, s_3, \ldots\}$ (in lexicographic order).
- Each language $L \in \mathcal{L}$ is associated with a unique sequence $\chi(L) \in \mathcal{B}:$
The Set of Languages is Uncountable (2)

Let $\mathcal{L}$ be the set of all languages over alphabet $\Sigma$. Recall $\mathcal{B}$ is the set of infinite binary sequences. We give a correspondence

$$\chi : \mathcal{L} \rightarrow \mathcal{B}$$

called the language’s characteristic sequence.

Let $\Sigma^* = \{s_1, s_2, s_3, \ldots\}$ (in lexicographic order).

Each language $L \in \mathcal{L}$ is associated with a unique sequence $\chi(L) \in \mathcal{B}$:

the $i$-th bit of $\chi(L)$ is 1 if and only if $s_i \in L$. 
The Set of Languages is Uncountable (3)

Each language $L \in \mathcal{L}$ has a unique sequence $\chi(L) \in \mathcal{B}$:

the $i$-th bit of $\chi(L)$ is 1 if and only if $s_i \in L$. 
The Set of Languages is Uncountable (3)

Each language $L \in \mathcal{L}$ has a unique sequence $\chi(L) \in \mathcal{B}$:

the $i$-th bit of $\chi(L)$ is 1 if and only if $s_i \in L$.

Example:

$\Sigma^*$

$\{ \varepsilon, 0, 1, 00, 01, 10, 11, 000 \ldots \}$

$A$

$\{ 0, 00, 01, 000 \ldots \}$

$\chi(A)$

$\{ 0, 1, 0 1, 1, 0 0 1 \ldots \}$
The Set of Languages is Uncountable (3)

Each language $L \in \mathcal{L}$ has a unique sequence $\chi(L) \in \mathcal{B}$:
the $i$-th bit of $\chi(L)$ is 1 if and only if $s_i \in L$.

**Example:**

$\Sigma^* \{ \varepsilon, 0, 1, 00, 01, 10, 11, 000 \ldots \}$
$A \{ 0, 00, 01, \quad 000 \ldots \}$
$\chi(A) \{ 0, 1, 0 \ 1, \ 1, \ 0 \ 0 \ 1 \ \ldots \}$

The map $\chi : \mathcal{L} \rightarrow \mathcal{B}$

- is one-to-one and onto (why?),
- and is hence a correspondence.
- It follows that $\mathcal{L}$ is uncountable.
TMs vs. Languages

We saw that the set of all Turing machines is countable.
TMs vs. Languages

We saw that the set of all Turing machines is countable.

We saw that the set $\mathcal{L}$ of all languages over alphabet $\Sigma$ is uncountable.
TMs vs. Languages

We saw that the set of all Turing machines is countable.

We saw that the set \( \mathcal{L} \) of all languages over alphabet \( \Sigma \) is uncountable.

Therefore there are languages that are not accepted by any TM.
TMs vs. Languages

We saw that the set of all Turing machines is countable.

We saw that the set $\mathcal{L}$ of all languages over alphabet $\Sigma$ is uncountable.

Therefore there are languages that are not accepted by any TM.

This is an existential proof – it does not explicitly show any such language.
Halting, Again

At long last, we are able to prove the undecidability of

\[ A_{\text{TM}} = \{ \langle M, w \rangle | M \text{ is a TM that accepts } w \} \]
Halting, Again

At long last, we are able to prove the undecidability of

\[ A_{TM} = \{ \langle M, w \rangle | M \text{ is a TM that accepts } w \} \].

**Proof:** By contradiction. Suppose a TM, \( H \), is a decider for \( A_{TM} \).

On input \( \langle M, w \rangle \), where \( M \) is a TM and \( w \) is a string, \( H \) halts and accepts if and only if \( M \) accepts \( w \). Furthermore, \( H \) halts and rejects if \( M \) fails to accept \( w \).
Halting (2)

On input $\langle M, w \rangle$, where $M$ is a TM and $w$ is a string, $H$ halts and accepts if and only if $M$ accepts $w$. Furthermore, $H$ halts and rejects if $M$ fails to accept $w$.

$$H(\langle M, w \rangle) = \begin{cases} 
    \text{accept} & \text{if } M \text{ accepts } w \\
    \text{reject} & \text{if } M \text{ does not accept } w
\end{cases}$$
Halting (3)

Now we construct a new TM, $D$, with $H$ as a subroutine.

$D$ does the following

- Calls $H$ to determine what TM, $M$, does when the input to $M$ is its own description, $\langle M \rangle$. 
Halting (3)

Now we construct a new TM, $D$, with $H$ as a subroutine.

$D$ does the following

- Calls $H$ to determine what TM, $M$, does when the input to $M$ is its own description, $\langle M \rangle$.
- When $D$ determines this, it does the opposite.
Halting (3)

Now we construct a new TM, $D$, with $H$ as a subroutine.

$D$ does the following

- Calls $H$ to determine what TM, $M$, does when the input to $M$ is its own description, $\langle M \rangle$.
- When $D$ determines this, it does the opposite.
- So $D$ rejects if $M$ accepts $\langle M \rangle$, and accepts if $M$ does not accept $\langle M \rangle$. 

Slides modified by Benny Chor, based on original slides by Maurice Herlihy, Brown University.
Halting (4)

More precisely, \( D \) does the following:

- Run \( H \) on input \( \langle M, \langle M \rangle \rangle \).
More precisely, $D$ does the following:

- Run $H$ on input $\langle M, \langle M \rangle \rangle$.
- Output the opposite of what $H$ outputs:
Halting (4)

More precisely, $D$ does the following:

- Run $H$ on input $\langle M, \langle M \rangle \rangle$.
- Output the opposite of what $H$ outputs:
  - If $H$ accepts, reject, and
Halting (4)

More precisely, $D$ does the following:

- Run $H$ on input $\langle M, \langle M \rangle \rangle$.
- Output the opposite of what $H$ outputs:
  - If $H$ accepts, reject, and
  - If $H$ rejects, accept.
Self Reference (4)

Don’t be confused by the notion of running a machine on its own description!

Actually, you should get used to it.

- Notion of self-reference comes up again and again in diverse areas.
- Read “Gödel, Escher, Bach, an Eternal Golden Braid”, by Douglas Hofstadter.
- This notion of self-reference is the basic idea behind Gödel’s revolutionary result.

Compilers do this all the time . . . .
The Punch Line

So far we have,

\[ D(\langle M \rangle) = \begin{cases} 
\text{reject} & \text{if } M \text{ accepts } \langle M \rangle \\
\text{accept} & \text{if } M \text{ does not accept } \langle M \rangle 
\end{cases} \]
The Punch Line

So far we have,

\[ D(\langle M \rangle) = \begin{cases} 
  \text{reject} & \text{if } M \text{ accepts } \langle M \rangle \\
  \text{accept} & \text{if } M \text{ does not accept } \langle M \rangle 
\end{cases} \]

What happens if we run \( D \) on its own description?
The Punch Line

So far we have,

\[ D(\langle M \rangle) = \begin{cases} 
\text{reject} & \text{if } M \text{ accepts } \langle M \rangle \\
\text{accept} & \text{if } M \text{ does not accept } \langle M \rangle 
\end{cases} \]

What happens if we run \( D \) on its own description?

\[ D(\langle D \rangle) = \begin{cases} 
\text{reject} & \text{if } D \text{ accepts } \langle D \rangle \\
\text{accept} & \text{if } D \text{ does not accept } \langle D \rangle 
\end{cases} \]

Oh, oh...

Or, more accurately, a contradiction (to what?)
Once Again

Assume that TM $H$ decides $A_{TM}$.
Once Again

- Assume that TM $H$ decides $A_{TM}$.
- Then use $H$ to build a TM, $D$, that when given $\langle M \rangle$, accepts exactly when $M$ does not accept.

Last step leads to contradiction. Therefore neither TM $D$ nor $H$ can exist.

So $A_{TM}$ is undecidable!
Once Again

- Assume that TM $H$ decides $A_{TM}$.
- Then use $H$ to build a TM, $D$, that when given $\langle M \rangle$, accepts exactly when $M$ does not accept.
- Run $D$ on its own description.
Once Again

- Assume that TM $H$ decides $A_{TM}$.
- Then use $H$ to build a TM, $D$, that when given $\langle M \rangle$, accepts exactly when $M$ does not accept.
- Run $D$ on its own description.
- $D$ does:
Once Again

- Assume that TM $H$ decides $A_{TM}$.
- Then use $H$ to build a TM, $D$, that when given $\langle M \rangle$, accepts exactly when $M$ does not accept.
- Run $D$ on its own description.
- $D$ does:
  - $H$ accepts $\langle M, w \rangle$ when $M$ accepts $w$.
Once Again

- Assume that TM $H$ decides $A_{TM}$.
- Then use $H$ to build a TM, $D$, that when given $\langle M \rangle$, accepts exactly when $M$ does not accept.
- Run $D$ on its own description.
- $D$ does:
  - $H$ accepts $\langle M, w \rangle$ when $M$ accepts $w$.
  - $D$ rejects $\langle M \rangle$ exactly when $M$ accepts $\langle M \rangle$. 

Last step leads to contradiction.
Therefore neither TM $D$ nor $H$ can exist.
So $A_{TM}$ is undecidable!
Once Again

- Assume that TM $H$ decides $A_{TM}$.
- Then use $H$ to build a TM, $D$, that when given $\langle M \rangle$, accepts exactly when $M$ does not accept.
- Run $D$ on its own description.
- $D$ does:
  - $H$ accepts $\langle M, w \rangle$ when $M$ accepts $w$.
  - $D$ rejects $\langle M \rangle$ exactly when $M$ accepts $\langle M \rangle$.
  - $D$ rejects $\langle D \rangle$ exactly when $D$ accepts $\langle D \rangle$.

Therefore neither TM $D$ nor $H$ can exist. So $A_{TM}$ is undecidable!
Once Again

- Assume that TM $H$ decides $A_{TM}$.
- Then use $H$ to build a TM, $D$, that when given $\langle M \rangle$, accepts exactly when $M$ does not accept.
- Run $D$ on its own description.
- $D$ does:
  - $H$ accepts $\langle M, w \rangle$ when $M$ accepts $w$.
  - $D$ rejects $\langle M \rangle$ exactly when $M$ accepts $\langle M \rangle$.
  - $D$ rejects $\langle D \rangle$ exactly when $D$ accepts $\langle D \rangle$.
- Last step leads to contradiction.
Once Again

- Assume that TM $H$ decides $A_{TM}$.
- Then use $H$ to build a TM, $D$, that when given $⟨M⟩$, accepts exactly when $M$ does not accept.
- Run $D$ on its own description.
- $D$ does:
  - $H$ accepts $⟨M, w⟩$ when $M$ accepts $w$.
  - $D$ rejects $⟨M⟩$ exactly when $M$ accepts $⟨M⟩$.
  - $D$ rejects $⟨D⟩$ exactly when $D$ accepts $⟨D⟩$.
- Last step leads to contradiction.
- Therefore neither TM $D$ nor $H$ can exist.
Once Again

- Assume that TM $H$ decides $A_{TM}$.
- Then use $H$ to build a TM, $D$, that when given $\langle M \rangle$, accepts exactly when $M$ does not accept.
- Run $D$ on its own description.
- $D$ does:
  - $H$ accepts $\langle M, w \rangle$ when $M$ accepts $w$.
  - $D$ rejects $\langle M \rangle$ exactly when $M$ accepts $\langle M \rangle$.
  - $D$ rejects $\langle D \rangle$ exactly when $D$ accepts $\langle D \rangle$.
- Last step leads to contradiction.
- Therefore neither TM $D$ nor $H$ can exist.
- So $A_{TM}$ is undecidable!
Diagonalization

This proof is diagonalization in transparent disguise. To unveil this, let’s start by making a table.
Diagonalization

This proof is diagonalization in transparent disguise. To unveil this, let’s start by making a table.

<table>
<thead>
<tr>
<th></th>
<th>$\langle M_1 \rangle$</th>
<th>$\langle M_2 \rangle$</th>
<th>$\langle M_3 \rangle$</th>
<th>$\langle M_4 \rangle$</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>$M_1$</td>
<td>accept</td>
<td>accept</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$M_2$</td>
<td>accept</td>
<td>accept</td>
<td>accept</td>
<td>accept</td>
<td></td>
</tr>
<tr>
<td>$M_3$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$M_4$</td>
<td>accept</td>
<td>accept</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Entry $(i, j)$ is accept if $M_i$ accepts $\langle M_j \rangle$, and blank if $M_i$ rejects or loops on $\langle M_j \rangle$. 
## Diagonalization (2)

<table>
<thead>
<tr>
<th></th>
<th>$\langle M_1 \rangle$</th>
<th>$\langle M_2 \rangle$</th>
<th>$\langle M_3 \rangle$</th>
<th>$\langle M_4 \rangle$</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>$M_1$</td>
<td>accept</td>
<td>accept</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$M_2$</td>
<td>accept</td>
<td>accept</td>
<td>accept</td>
<td>accept</td>
<td></td>
</tr>
<tr>
<td>$M_3$</td>
<td></td>
<td></td>
<td>reject</td>
<td>reject</td>
<td></td>
</tr>
<tr>
<td>$M_4$</td>
<td>accept</td>
<td>accept</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Run $H$ on corresponding inputs. In new table, entry $(i,j)$ states whether $H$ accepts $\langle M_i, \langle M_j \rangle \rangle$. 

Slides modified by Benny Chor, based on original slides by Maurice Herlihy, Brown University.
Diagonalization (2)

<table>
<thead>
<tr>
<th></th>
<th>(M_1)</th>
<th>(M_2)</th>
<th>(M_3)</th>
<th>(M_4)</th>
<th>\ldots</th>
</tr>
</thead>
<tbody>
<tr>
<td>(M_1)</td>
<td>accept</td>
<td>accept</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(M_2)</td>
<td>accept</td>
<td>accept</td>
<td>accept</td>
<td>accept</td>
<td></td>
</tr>
<tr>
<td>(M_3)</td>
<td>accept</td>
<td>accept</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(M_4)</td>
<td>accept</td>
<td>accept</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>\vdots</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Run \(H\) on on corresponding inputs. In new table, entry \((i, j)\) states whether \(H\) accepts \(\langle M_i, \langle M_j \rangle \rangle\).

<table>
<thead>
<tr>
<th></th>
<th>(M_1)</th>
<th>(M_2)</th>
<th>(M_3)</th>
<th>(M_4)</th>
<th>\ldots</th>
</tr>
</thead>
<tbody>
<tr>
<td>(M_1)</td>
<td>accept</td>
<td>reject</td>
<td>accept</td>
<td>reject</td>
<td></td>
</tr>
<tr>
<td>(M_2)</td>
<td>accept</td>
<td>accept</td>
<td>accept</td>
<td>accept</td>
<td></td>
</tr>
<tr>
<td>(M_3)</td>
<td>reject</td>
<td>reject</td>
<td>reject</td>
<td>reject</td>
<td></td>
</tr>
<tr>
<td>(M_4)</td>
<td>accept</td>
<td>accept</td>
<td>reject</td>
<td>reject</td>
<td></td>
</tr>
<tr>
<td>\vdots</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Diagonalization (3)

Now we add $D$ to the table.

- By assumption, $H$ is a TM, and therefore so is $D$.
- It occurs on the list $M_1, M_2, \ldots$ of all TMs.
- $D$ computes the opposite of the diagonal entries.
- At diagonal entry, $D$ computes its own opposite!

<table>
<thead>
<tr>
<th></th>
<th>$\langle M_1 \rangle$</th>
<th>$\langle M_2 \rangle$</th>
<th>$\langle M_3 \rangle$</th>
<th>...</th>
<th>$\langle D \rangle$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$M_1$</td>
<td>accept</td>
<td>reject</td>
<td>accept</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$M_2$</td>
<td>accept</td>
<td>accept</td>
<td>accept</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$M_3$</td>
<td>reject</td>
<td>reject</td>
<td>reject</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$M_4$</td>
<td>accept</td>
<td>accept</td>
<td>reject</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
| $D$   | reject                | reject                | accept                |     | ???

... ... ...
A Non-enumerable Language

We already saw a non-decidable language: $A_{TM}$. 
A Non-enumerable Language

- We already saw a non-decidable language: $A_{TM}$.
- Can we do better (i.e., worse)?
A Non-enumerable Language

- We already saw a non-decidable language: $A_{TM}$.
- Can we do better (i.e., worse)?
- Mais, oui!
A Non-enumerable Language

- We already saw a non-decidable language: $\mathcal{A}^\text{TM}$.
- Can we do better (i.e., worse)?
- Mais, oui!
- We now display a language that isn’t even enumerable . . . .
A Non-enumerable Language

In Lecture 7 we proved

**Theorem:** If $L$ and $\overline{L}$ are both enumerable, then $L$ is decidable.
A Non-enumerable Language

In Lecture 7 we proved

**Theorem:** If $L$ and $\overline{L}$ are both enumerable, then $L$ is decidable.

**Corollary:** If $L$ is not decidable, then either $L$ or $\overline{L}$ is not enumerable.
A Non-enumerable Language

In Lecture 7 we proved

**Theorem:** If $L$ and $\overline{L}$ are both enumerable, then $L$ is decidable.

**Corollary:** If $L$ is not decidable, then either $L$ or $\overline{L}$ is not enumerable.

**Definition:** A language is co-enumerable if it is the complement of an enumerable language.
A Non-enumerable Language

In Lecture 7 we proved

**Theorem:** If $L$ and $\overline{L}$ are both enumerable, then $L$ is decidable.

**Corollary:** If $L$ is not decidable, then either $L$ or $\overline{L}$ is not enumerable.

**Definition:** A language is co-enumerable if it is the complement of an enumerable language.

Reformulating theorem

**Theorem:** A language is decidable if and only if it is both enumerable and co-enumerable.
$\overline{A_{TM}}$ is not Enumerable

**Theorem:** If $L$ and $\overline{L}$ are both enumerable, then $L$ is decidable.

We proved that $A_{TM}$ is undecidable.
\( \overline{A_{\text{TM}}} \) is not Enumerable

**Theorem:** If \( L \) and \( \overline{L} \) are both enumerable, then \( L \) is decidable.

- We proved that \( A_{\text{TM}} \) is undecidable.
- On the other hand, we saw that the universal TM, \( U \), accepts \( A_{\text{TM}} \).
\( \overline{A_{TM}} \) is not Enumerable

**Theorem:** If \( L \) and \( \overline{L} \) are both enumerable, then \( L \) is decidable.

- We proved that \( A_{TM} \) is undecidable.
- On the other hand, we saw that the universal TM, \( U \), accepts \( A_{TM} \).
- Therefore \( A_{TM} \) is enumerable.
\( \overline{A_{TM}} \) is not Enumerable

**Theorem:** If \( L \) and \( \overline{L} \) are both enumerable, then \( L \) is decidable.

- We proved that \( A_{TM} \) is undecidable.
- On the other hand, we saw that the universal TM, \( U \), accepts \( A_{TM} \).
- Therefore \( A_{TM} \) is enumerable.
- If \( A_{TM} \) were also enumerable, then by theorem \( A_{TM} \) was decidable.
\( \overline{A_{TM}} \) is not Enumerable

**Theorem:** If \( L \) and \( \overline{L} \) are both enumerable, then \( L \) is decidable.

- We proved that \( A_{TM} \) is undecidable.
- On the other hand, we saw that the universal TM, \( U \), accepts \( A_{TM} \).
- Therefore \( A_{TM} \) is enumerable.
- If \( A_{TM} \) were also enumerable, then by theorem \( A_{TM} \) was decidable.
- Therefore \( \overline{A_{TM}} \) is not enumerable.
Question: Are there any languages in the area marked ??? ?

Answer: Yes, heaps (why?)