Lecture 9

We have already

- Established Turing Machines as the gold standard of computers and computability ...
Lecture 9

We have already

- Established Turing Machines as the gold standard of computers and computability . . .
- seen examples of solvable problems . . .
Lecture 9

We have already

- Established Turing Machines as the gold standard of computers and computability . . .
- seen examples of solvable problems . . .
- and saw one problem, $A_{TM}$, that is computationally unsolvable.
Lecture 9

We have already

- Established Turing Machines as the gold standard of computers and computability . . .
- seen examples of solvable problems . . .
- and saw one problem, \( A_{TM} \), that is computationally unsolvable.

In this lecture, we look at other computationally unsolvable problems, and establish the technique of mapping reducibilities for prove that languages are undecidable/non-enumerable.
Reducibility

Example:

- Finding your way around a new city
Reducibility

Example:
- Finding your way around a new city
- reduces to . . .
Reducibility

Example:
- Finding your way around a new city
- reduces to . . .
- obtaining a city map.
Reducibility, In Our Context

Always involves two problems, $A$ and $B$. 
Reducibility, In Our Context

Always involves two problems, $A$ and $B$.

**Desired Property:** If $A$ reduces to $B$, then any solution of $B$ can be used to find a solution of $A$. 
Reducibility, In Our Context

Always involves two problems, $A$ and $B$.

**Desired Property:** If $A$ reduces to $B$, then any solution of $B$ can be used to find a solution of $A$.

**Remark:** This property says nothing about solving $A$ by itself or $B$ by itself.
Examples

Reductions:

- Traveling from Boshton to Paris …
Examples

Reductions:

- Traveling from Boshton to Paris . . .
- buying plane ticket . . .
Examples

Reductions:
- Traveling from Boshton to Paris . . .
- buying plane ticket . . .
- earning the money for that ticket . . .
Examples

Reductions:

- Traveling from Boshton to Paris . . .
- buying plane ticket . . .
- earning the money for that ticket . . .
- finding a job
  (or getting the $s from mom and dad. . . )
Examples

Reductions:

- Measuring area of rectangle . . .
Examples

Reductions:

- Measuring area of rectangle . . .
- measuring lengths of sides.
Examples

Reductions:

- Measuring area of rectangle . . .
- measuring lengths of sides.

Also:
Examples

Reductions:
- Measuring area of rectangle . . .
- measuring lengths of sides.

Also:
- Solving a system of linear equations . . .
- inverting a matrix.
Reducibility

If $A$ is reducible to $B$, then

- $A$ cannot be harder than $B$
Reducibility

If $A$ is reducible to $B$, then

- $A$ cannot be harder than $B$
- if $B$ is decidable, so is $A$. 
Reducibility

If \( A \) is reducible to \( B \), then

- \( A \) cannot be harder than \( B \)
- if \( B \) is decidable, so is \( A \).
- if \( A \) is undecidable and reducible to \( B \), then \( B \) is undecidable.
Undecidable Problems

We have already established that $A_{TM}$ is undecidable.

Here is a related problem.

$H_{TM} = \{ \langle M, w \rangle | M \text{ is a TM and } M \text{ halts on input } w \}$

Clarification: How does $H_{TM}$ differ from $A_{TM}$?
Undecidable Problems

\[ H_{TM} = \{ \langle M, w \rangle | M \text{ is a TM and } M \text{ halts on input } w \} \]

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

**Proof idea:**
- By contradiction.
Undecidable Problems

\[ H_{TM} = \{ \langle M, w \rangle | M \text{ is a TM and } M \text{ halts on input } w \} \]

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

Proof idea:
- By contradiction.
- Assume \( H_{TM} \) is decidable.
Undecidable Problems

\[ H_{TM} = \{ \langle M, w \rangle | M \text{ is a TM and } M \text{ halts on input } w \} \]

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

Proof idea:
- By contradiction.
- Assume \( H_{TM} \) is decidable.
- Let \( R \) be a TM that decides \( H_{TM} \).
Undecidable Problems

\[ H_{TM} = \{ \langle M, w \rangle \mid M \text{ is a TM and } M \text{ halts on input } w \} \]

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

Proof idea:
- By contradiction.
- Assume \( H_{TM} \) is decidable.
- Let \( R \) be a TM that decides \( H_{TM} \).
- Use \( R \) to construct \( S \), a TM that decides \( A_{TM} \).
Undecidable Problems

\[ H_{TM} = \{ \langle M, w \rangle | M \text{ is a TM and } M \text{ halts on input } w \} \]

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

Proof idea:

- By contradiction.
- Assume \( H_{TM} \) is decidable.
- Let \( R \) be a TM that decides \( H_{TM} \).
- Use \( R \) to construct \( S \), a TM that decides \( A_{TM} \).
- So \( A_{TM} \) is reduced to \( H_{TM} \).
Undecidable Problems

\( H_{\text{TM}} = \{ \langle M, w \rangle | M \text{ is a TM and } M \text{ halts on input } w \} \)

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

**Proof idea:**
- By contradiction.
- Assume \( H_{\text{TM}} \) is decidable.
- Let \( R \) be a TM that decides \( H_{\text{TM}} \).
- Use \( R \) to construct \( S \), a TM that decides \( A_{\text{TM}} \).
- So \( A_{\text{TM}} \) is reduced to \( H_{\text{TM}} \).
- Since \( A_{\text{TM}} \) is undecidable, so is \( H_{\text{TM}} \).
Undecidable Problems

**Theorem:** $H_{TM}$ is undecidable.

**Proof:** Assume, by way of contradiction, that TM $R$ decides $H_{TM}$. Define a new TM, $S$, as follows:

- On input $\langle M, w \rangle$, ...
Undecidable Problems

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

**Proof:** Assume, by way of contradiction, that TM \( R \) decides \( H_{\text{TM}} \). Define a new TM, \( S \), as follows:

1. On input \( \langle M, w \rangle \),
2. run \( R \) on \( \langle M, w \rangle \).
Undecidable Problems

**Theorem:** $H_{TM}$ is undecidable.

**Proof:** Assume, by way of contradiction, that TM $R$ decides $H_{TM}$. Define a new TM, $S$, as follows:

- On input $\langle M, w \rangle$,
- run $R$ on $\langle M, w \rangle$.
- If $R$ rejects, reject.
Undecidable Problems

**Theorem:** $H_{TM}$ is undecidable.

**Proof:** Assume, by way of contradiction, that TM $R$ decides $H_{TM}$. Define a new TM, $S$, as follows:

1. On input $\langle M, w \rangle$,
2. run $R$ on $\langle M, w \rangle$.
3. If $R$ rejects, reject.
4. If $R$ accepts (meaning $M$ halts on $w$), simulate $M$ on $w$ until it halts.
Undecidable Problems

**Theorem:** $H_{TM}$ is undecidable.

**Proof:** Assume, by way of contradiction, that TM $R$ decides $H_{TM}$. Define a new TM, $S$, as follows:

- On input $\langle M, w \rangle$,
- run $R$ on $\langle M, w \rangle$.
- If $R$ rejects, reject.
- If $R$ accepts (meaning $M$ halts on $w$), simulate $M$ on $w$ until it halts.
- If $M$ accepted, accept; otherwise reject.
Undecidable Problems (2)

Does a TM accept any string at all?

\[ E_{TM} = \{ \langle M \rangle | M \text{ is a TM and } L(M) = \emptyset \} \]
Undecidable Problems (2)

Does a TM accept any string at all?

\[ E_{TM} = \{ \langle M \rangle | M \text{ is a TM and } L(M) = \emptyset \} \]

**Theorem**: \( E_{TM} \) is undecidable.
Undecidable Problems (2)

Does a TM accept any string at all?

\[ E_{\text{TM}} = \{ \langle M \rangle | M \text{ is a TM and } L(M) = \emptyset \} \]

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

**Proof structure:**
Undecidable Problems (2)

Does a TM accept any string at all?

\[ E_{TM} = \{ \langle M \rangle | M \text{ is a TM and } L(M) = \emptyset \} \]

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

**Proof structure:**

- By contradiction.
- Assume \( E_{TM} \) is decidable.
- Let \( R \) be a TM that decides \( E_{TM} \).
- Use \( R \) to construct \( S \), a TM that decides \( A_{TM} \).
Undecidable Problems (2)

\[ E_{TM} = \{ \langle M \rangle | M \text{ is a TM and } L(M) = \emptyset \} \]
Undecidable Problems (2)

\[ E_{TM} = \{ \langle M \rangle | M \text{ is a TM and } L(M) = \emptyset \} \]

First attempt: When \( S \) receives input \( \langle M, w \rangle \), it calls \( R \) with input \( \langle M \rangle \).

- If \( R \) accepts, then reject, because \( M \) does not accept any string, let alone \( w \).
- But what if \( R \) rejects?
Undecidable Problems (2)

\[ E_{\text{TM}} = \{ \langle M \rangle | M \text{ is a TM and } L(M) = \emptyset \} \]

First attempt: When \( S \) receives input \( \langle M, w \rangle \), it calls \( R \) with input \( \langle M \rangle \).

- If \( R \) accepts, then reject, because \( M \) does not accept any string, let alone \( w \).
- But what if \( R \) rejects?

Second attempt: Let’s modify \( M \).
Undecidable Problems (2)

\[ E_{TM} = \{ \langle M \rangle | M \text{ is a TM and } L(M) = \emptyset \} \]

Define \( M_1 \): on input \( x \),
1. if \( x \neq w \), reject.
2. if \( x = w \), run \( M \) on \( w \) and accept if \( M \) does.
Undecidable Problems (2)

\[ E_{TM} = \{ \langle M \rangle | M \text{ is a TM and } L(M) = \emptyset \} \]

Define \( M_1 \): on input \( x \),

1. if \( x \neq w \), reject.
2. if \( x = w \), run \( M \) on \( w \) and accept if \( M \) does.

\( M_1 \) either

- accepts just \( w \), or
- accepts nothing.
Undecidable Problems (2)

Machine $M_1$: on input $x$,

1. if $x \neq w$, reject.
2. if $x = w$, run $M$ on $w$ and accept if $M$ does.
Undecidable Problems (2)

Machine $M_1$: on input $x$,

1. if $x \neq w$, reject.
2. if $x = w$, run $M$ on $w$ and accept if $M$ does.

**Question:** Can a TM construct $M_1$ from $M$?
Undecidable Problems (2)

Machine $M_1$: on input $x$,

1. if $x \neq w$, reject.
2. if $x = w$, run $M$ on $w$ and accept if $M$ does.

Question: Can a TM construct $M_1$ from $M$?

Answer: Yes, because we need only hardwire $w$, and add a few extra states to perform the “$x = w$?” test.
Undecidable Problems (2)

\[ E_{TM} = \{ \langle M \rangle | M \text{ is a TM and } L(M) = \emptyset \} \]

**Theorem:** \( E_{TM} \) is undecidable.
Undecidable Problems (2)

$$E_{TM} = \{ \langle M \rangle | M \text{ is a TM and } L(M) = \emptyset \}$$

**Theorem:** $E_{TM}$ is undecidable.

Define $S$ as follows:
On input $\langle M, w \rangle$, where $M$ is a TM and $w$ a string,
Undecidable Problems (2)

\[ E_{TM} = \{ \langle M \rangle | M \text{ is a TM and } L(M) = \emptyset \} \]

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

Define \( S \) as follows:

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

- Construct \( M_1 \) from \( M \) and \( w \).
- Run \( R \) on input \( \langle M_1 \rangle \),
- if \( R \) accepts, reject; if \( R \) rejects, accept.
Undecidable Problems (3)

Does a TM accept a regular language?

\[ R_{\text{TM}} = \{ \langle M \rangle | M \text{ is a TM and } L(M) \text{ is regular} \} \]
Undecidable Problems (3)

Does a TM accept a regular language?

\[ R_{TM} = \{ \langle M \rangle | M \text{ is a TM and } L(M) \text{ is regular} \} \]

**Theorem:** \( R_{TM} \) is undecidable.
Undecidable Problems (3)

Does a TM accept a regular language?

\[ R_{TM} = \{ \langle M \rangle | M \text{ is a TM and } L(M) \text{ is regular} \} \]

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

**Skeleton of Proof:**
- By contradiction.
- Assume \( R_{TM} \) is decidable.
- Let \( R \) be a TM that decides \( R_{TM} \).
- Use \( R \) to construct \( S \), a TM that decides \( A_{TM} \).
Undecidable Problems (3)

Does a TM accept a regular language?

\[ R_{TM} = \{ \langle M \rangle | M \text{ is a TM and } L(M) \text{ is regular} \} \]

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

**Skeleton of Proof:**

- By contradiction.
- Assume \( R_{TM} \) is decidable.
- Let \( R \) be a TM that decides \( R_{TM} \).
- Use \( R \) to construct \( S \), a TM that decides \( A_{TM} \).

But how?
Undecidable Problems (3)

\[ R_{TM} = \{ \langle M \rangle | M \text{ is a TM and } L(M) \text{ is regular} \} \]

Modify \( M \) so that the resulting TM accepts a regular language if and only if \( M \) accepts \( w \).
 Undecidable Problems (3)

\[ R_{TM} = \{ \langle M \rangle | M \text{ is a TM and } L(M) \text{ is regular} \} \]

Modify \( M \) so that the resulting TM accepts a regular language if and only if \( M \) accepts \( w \).

Design \( M_2 \) so that

- if \( M \) does not accept \( w \), then \( M_2 \) accepts \( \{0^n1^n | n \geq 0\} \) (non-regular)
- if \( M \) accepts \( w \), then \( M_2 \) accepts \( \Sigma^* \) (regular).
Undecidable Problems (3)

From $M$ and $w$, define $M_2$:

1. If $x$ has the form $0^n1^n$, accept it.
2. Otherwise, run $M$ on input $w$ and accept $x$ if $M$ accepts $w$.

Claim: If $M$ does not accept $w$, then $M_2$ accepts \{$0^n1^n | n \geq 0$\}.

If $M$ accepts $w$, then $M_2$ accepts $\Sigma^\ast$. 

Undecidable Problems (3)

From $M$ and $w$, define $M_2$:

On input $x$,

1. If $x$ has the form $0^n1^n$, accept it.
2. Otherwise, run $M$ on input $w$ and accept $x$ if $M$ accepts $w$. 

Claim: If $M$ does not accept $w$, then $M_2$ accepts $\{0^n1^n \mid n \geq 0\}$. If $M$ accepts $w$, then $M_2$ accepts $\Sigma^*$. 

Slides modified by Benny Chor, based on original slides by Maurice Herlihy, Brown University. – p.17
Undecidable Problems (3)

From $M$ and $w$, define $M_2$:

On input $x$,

1. If $x$ has the form $0^n1^n$, accept it.
2. Otherwise, run $M$ on input $w$ and accept $x$ if $M$ accepts $w$.

Claim:

- If $M$ does not accept $w$, then $M_2$ accepts $\{0^n1^n | n \geq 0\}$.
- If $M$ accepts $w$, then $M_2$ accepts $\Sigma^*$. 
Undecidable Problems (3)

\[ R_{TM} = \{ \langle M \rangle | M \text{ is a TM and } L(M) \text{ is regular} \} \]

**Theorem:** \( R_{TM} \) is undecidable.
Undecidable Problems (3)

\[ R_{\text{TM}} = \{ \langle M \rangle | M \text{ is a TM and } L(M) \text{ is regular} \} \]

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

Define \( S \):

On input \( \langle M, w \rangle \),

1. Construct \( M_2 \) from \( M \) and \( w \).
2. Run \( R \) on input \( \langle M_2 \rangle \).
3. If \( R \) accepts, accept; if \( R \) rejects, reject.
Undecidable Problems (4)

Are two TMs equivalent?

\[
\text{EQ}_{\text{TM}} = \{ \langle M_1, M_2 \rangle \mid M_1, M_2 \text{ are TMs and } L(M_1) = L(M_2) \} 
\]

**Theorem:** \( \text{EQ}_{\text{TM}} \) is undecidable.
Undecidable Problems (4)

Are two TMs equivalent?

\[ EQ_{TM} = \{ \langle M_1, M_2 \rangle \mid M_1, M_2 \text{ are TMs and } L(M_1) = L(M_2) \} \]

Theorem:  \( EQ_{TM} \) is undecidable.

We are getting tired of reducing \( A_{TM} \) to everything.
Undecidable Problems (4)

Are two TMs equivalent?

\[ \text{EQ}_{\text{TM}} = \{ \langle M_1, M_2 \rangle \mid M_1, M_2 \text{ are TMs and } L(M_1) = L(M_2) \} \]

Theorem: \text{EQ}_{\text{TM}} \text{ is undecidable.}

We are getting tired of reducing \( A_{\text{TM}} \) to everything.

Let’s try instead a reduction from \( E_{\text{TM}} \) to \( \text{EQ}_{\text{TM}} \).
Undecidable Problems (4)

\[ \text{EQ}_{\text{TM}} = \{ \langle M_1, M_2 \rangle \mid M_1, M_2 \text{ are TMs and } L(M_1) = L(M_2) \} \]

**Theorem:** \( \text{EQ}_{\text{TM}} \) is undecidable.

**Idea:**

- \( \text{E}_{\text{TM}} \) is the problem of testing whether a TM language is empty.

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

\[ \text{EQ}_{\text{TM}} = \{ \langle M_1, M_2 \rangle \mid M_1, M_2 \text{ are TMs and } L(M_1) = L(M_2) \} \]

**Theorem:** \( \text{EQ}_{\text{TM}} \) is undecidable.

**Idea:**

- \( E_{\text{TM}} \) is the problem of testing whether a TM language is empty.
- \( \text{EQ}_{\text{TM}} \) is the problem of testing whether two TM languages are the same.
Undecidable Problems (4)

\[
\text{EQ}_{\text{TM}} = \{ \langle M_1, M_2 \rangle \mid M_1, M_2 \text{ are TMs and } L(M_1) = L(M_2) \}\]

**Theorem:** \( \text{EQ}_{\text{TM}} \) is undecidable.

**Idea:**

- \( E_{\text{TM}} \) is the problem of testing whether a TM language is empty.
- \( \text{EQ}_{\text{TM}} \) is the problem of testing whether two TM languages are the same.
- If one of these two TM languages happens to be empty, then we are back to \( E_{\text{TM}} \).
Undecidable Problems (4)

$$\text{EQ}_{\text{TM}} = \{ \langle M_1, M_2 \rangle \mid M_1, M_2 \text{ are TMs and } L(M_1) = L(M_2) \}$$

**Theorem:** $\text{EQ}_{\text{TM}}$ is undecidable.

**Idea:**

- $E_{\text{TM}}$ is the problem of testing whether a TM language is empty.
- $\text{EQ}_{\text{TM}}$ is the problem of testing whether two TM languages are the same.
- If one of these two TM languages happens to be empty, then we are back to $E_{\text{TM}}$.
- So $E_{\text{TM}}$ is a special case of $\text{EQ}_{\text{TM}}$.

The rest is easy.
Undecidable Problems (4)

$$EQ_{TM} = \{ \langle M_1, M_2 \rangle \mid M_1, M_2 \text{ are TMs and } L(M_1) = L(M_2) \}$$

**Theorem:** $EQ_{TM}$ is undecidable.

Let $M_{NO}$ be the TM: On input $x$, reject.

Let $R$ decide $EQ_{TM}$. 
Undecidable Problems (4)

\[ \text{EQ}_{\text{TM}} = \{ \langle M_1, M_2 \rangle \mid M_1, M_2 \text{ are TMs and} \]
\[ L(M_1) = L(M_2) \} \]

**Theorem:** \( \text{EQ}_{\text{TM}} \) is undecidable.

Let \( M_{\text{NO}} \) be the TM: On input \( x \), reject.
Let \( R \) decide \( \text{EQ}_{\text{TM}} \).

Let \( S \) be: On input \( \langle M \rangle \): 
1. Run \( R \) on input \( \langle M, M_{\text{NO}} \rangle \).
2. If \( R \) accepts, accept; if \( R \) rejects, reject.
Undecidable Problems (4)

**EQ\textsubscript{TM}** = \{⟨M\textsubscript{1}, M\textsubscript{2}⟩ | M\textsubscript{1}, M\textsubscript{2} are TMs and \(L(M\textsubscript{1}) = L(M\textsubscript{2})\)\}

**Theorem:** EQ\textsubscript{TM} is undecidable.

Let \(M\textsubscript{NO}\) be the TM: On input \(x\), reject.

Let \(R\) decide \(E\textsubscript{TM}\).

Let \(S\) be: On input \(⟨M⟩\):

1. Run \(R\) on input \(⟨M, M\textsubscript{NO}⟩\).
2. If \(R\) accepts, accept; if \(R\) rejects, reject.

If \(R\) decides \(E\textsubscript{TM}\), then \(S\) decides \(E\textsubscript{TM}\).
Bucket of Undecidable Problems

Same techniques prove undecidability of

Does a TM accept a **decidable** language?
Bucket of Undecidable Problems

Same techniques prove undecidability of

- Does a TM accept a **decidable** language?
- Does a TM accept a **enumerable** language?
Bucket of Undecidable Problems

Same techniques prove undecidability of

- Does a TM accept a **decidable** language?
- Does a TM accept a **enumerable** language?
- Does a TM accept a **context-free** language?
Bucket of Undecidable Problems

Same techniques prove undecidability of

- Does a TM accept a **decidable** language?
- Does a TM accept a **enumerable** language?
- Does a TM accept a **context-free** language?
- Does a TM accept a **finite** language?
Bucket of Undecidable Problems

Same techniques prove undecidability of

- Does a TM accept a **decidable** language?
- Does a TM accept a **enumerable** language?
- Does a TM accept a **context-free** language?
- Does a TM accept a **finite** language?
- Does a TM halt on **all inputs**?
Bucket of Undecidable Problems

Same techniques prove undecidability of

- Does a TM accept a **decidable** language?
- Does a TM accept a **enumerable** language?
- Does a TM accept a **context-free** language?
- Does a TM accept a **finite** language?
- Does a TM halt on all inputs?
- Is there an input string that causes a TM to traverse all its states?
Rice’s Theorem

By now, some of you may have become cynical and embittered.

Like, been there, done that, bought the T-shirt.
Rice’s Theorem

By now, some of you may have become cynical and embittered.

- Like, been there, done that, bought the T-shirt.
- Looks like any non-trivial property of TMs is undecidable.
Rice’s Theorem

By now, some of you may have become cynical and embittered.

- Like, been there, done that, bought the T-shirt.
- Looks like any non-trivial property of TMs is undecidable.

That is correct.
Rice’s Theorem

**Theorem:** If $C$ is a proper non-empty subset of the set of enumerable languages, then it is undecidable whether for a given TM, $M$, $L(M)$ is in $C$. 
Rice’s Theorem

**Theorem:** If $C$ is a proper non-empty subset of the set of enumerable languages, then it is undecidable whether for a given TM, $M$, $L(M)$ is in $C$.

Proof by reduction from $H_{TM}$ (does $M$ halt on input $x$?).
Rice’s Theorem

**Theorem:** If $C$ is a proper non-empty subset of the set of enumerable languages, then it is undecidable whether for a given TM, $M$, $L(M)$ is in $C$.

Proof by reduction from $H_{TM}$ (does $M$ halt on input $x$?).

- Assume $R$ decides if $L(M) \in C$.
- Use $R$ to implement $S$, which decides $H_{TM}$.

Further details of proof not given at the moment . . .
Reducibility

So far, we have seen many examples of reductions from one language to another, but the notion was neither defined nor treated formally.

Reductions play an important role in

- decidability theory
- complexity theory (to come)

Time to get formal.
Computable Functions

A TM computes a function

\[ f : \Sigma^* \rightarrow \Sigma^* \]

if the TM
Computable Functions

A TM computes a function

\[ f : \Sigma^* \rightarrow \Sigma^* \]

if the TM

starts with input \( w \), and
Computable Functions

A TM computes a function

\[ f : \Sigma^* \longrightarrow \Sigma^* \]

if the TM

- starts with input \( w \), and
- halts with only \( f(w) \) on tape.
Computable Functions

Claim: All the usual arithmetic functions on integers are computable.
Computable Functions

**Claim:** All the usual arithmetic functions on integers are computable.

These include addition, subtraction, multiplication, division (quotient and remainder), exponentiation, roots (to a specified precision).

Exercise: Design a TM that on input $\langle m, n \rangle$, halts with $\langle m + n \rangle$ on tape.
Computable Functions

**Claim:** All the usual arithmetic functions on integers are computable.

These include addition, subtraction, multiplication, division (quotient and remainder), exponentiation, roots (to a specified precision).

Even **non-arithmetic** functions, like logarithms and trigonometric functions, can be computed (to a specified precision), using Taylor expansion or other numeric mathematic techniques.
Computable Functions

Claim: All the usual arithmetic functions on integers are computable.

These include addition, subtraction, multiplication, division (quotient and remainder), exponentiation, roots (to a specified precision).

Even non-arithmetic functions, like logarithms and trigonometric functions, can be computed (to a specified precision), using Taylor expansion or other numeric mathematic techniques.

Exercise: Design a TM that on input $\langle m, n \rangle$, halts with $\langle m + n \rangle$ on tape.
Computable Functions

A useful class of functions modifies TM descriptions. For example:

On input $w$:

- if $w = \langle M \rangle$ for some TM,
Computable Functions

A useful class of functions modifies TM descriptions. For example:

On input $w$:

- if $w = \langle M \rangle$ for some TM,
  - construct $\langle M' \rangle$, where
Computable Functions

A useful class of functions modifies TM descriptions. For example:

On input $w$:

- if $w = \langle M \rangle$ for some TM,
  - construct $\langle M' \rangle$, where
    - $L(M') = L(M)$, but
Computable Functions

A useful class of functions modifies TM descriptions. For example:

On input $w$:

- if $w = \langle M \rangle$ for some TM,
  - construct $\langle M' \rangle$, where
  - $L(M') = L(M)$, but
  - $M'$ never tries to move off LHS of tape.
Computable Functions

A useful class of functions modifies TM descriptions. For example:

On input $w$:

- if $w = \langle M \rangle$ for some TM,
  - construct $\langle M' \rangle$, where
  - $L(M') = L(M)$, but
  - $M'$ never tries to move off LHS of tape.
- otherwise write $\varepsilon$ and halt.
Computable Functions

A useful class of functions modifies TM descriptions. For example:

On input $w$:

- if $w = \langle M \rangle$ for some TM,
  - construct $\langle M' \rangle$, where
    - $L(M') = L(M)$, but
    - $M'$ never tries to move off LHS of tape.
- otherwise write $\varepsilon$ and halt.

Left as an exercise.
Mapping Reductions

**Definition:** Let $A$ and $B$ be two languages. We say that there is a **mapping reduction** from $A$ to $B$, and denote

$$A \leq_m B$$
Mapping Reductions

Definition: Let $A$ and $B$ be two languages. We say that there is a mapping reduction from $A$ to $B$, and denote

$$A \leq_m B$$

if there is a computable function

$$f : \Sigma^* \rightarrow \Sigma^*$$

such that, for every $w$, 
Mapping Reductions

**Definition:** Let $A$ and $B$ be two languages. We say that there is a mapping reduction from $A$ to $B$, and denote

$$A \leq_m B$$

if there is a computable function

$$f : \Sigma^* \rightarrow \Sigma^*$$

such that, for every $w$,

$$w \in A \iff f(w) \in B.$$ 

The function $f$ is called the reduction from $A$ to $B$. 
Mapping Reductions

A mapping reduction converts questions about membership in A to membership in B.
A mapping reduction converts questions about membership in \( A \) to membership in \( B \)
Mapping Reductions

**Theorem:** If $A \leq_m B$ and $B$ is decidable, then $A$ is decidable.
Mapping Reductions

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

**Proof:** Let

- $M$ be the decider for $B$, and
- $f$ the reduction from $A$ to $B$. 

Slides modified by Benny Chor, based on original slides by Maurice Herlihy, Brown University. – p.31
Mapping Reductions

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

**Proof:** Let

- $M$ be the decider for $B$, and
- $f$ the reduction from $A$ to $B$.

Define $N$: On input $w$

1. compute $f(w)$
2. run $M$ on input $f(w)$ and output whatever $M$ outputs.
Mapping Reductions

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

**Corollary**: If $A \leq_m B$ and $A$ is undecidable, then $B$ is undecidable.

In fact, this has been our principal tool for proving undecidability of languages other than $A_{TM}$.
Example: Halting

Recall that

\[ \mathcal{A}_{\text{TM}} = \{ \langle M, w \rangle | \text{TM } M \text{ accepts input } w \} \]
\[ \mathcal{H}_{\text{TM}} = \{ \langle M, w \rangle | \text{TM } M \text{ halts on input } w \} \]
Example: Halting

Recall that

\[ A_{TM} = \{ \langle M, w \rangle | \text{TM } M \text{ accepts input } w \} \]
\[ H_{TM} = \{ \langle M, w \rangle | \text{TM } M \text{ halts on input } w \} \]

Earlier we proved that

- \( H_{TM} \) is undecidable
- by (de facto) reduction from \( A_{TM} \).

Let’s reformulate this.
Example: Halting

Define a **computable function**, \( f \):

- input of form \( \langle M, w \rangle \)
Example: Halting

Define a **computable function**, $f$:

- input of form $\langle M, w \rangle$
- output of form $\langle M', w' \rangle$
Example: Halting

Define a **computable function**, \( f \):

- input of form \( \langle M, w \rangle \)
- output of form \( \langle M', w' \rangle \)
- where \( \langle M, w \rangle \in A_{TM} \iff \langle M', w' \rangle \in H_{TM} \).
Example: Halting

The following machine computes this function $f$.

$F = \text{on input } \langle M, w \rangle:$

Construct the following machine $M'$. $M'$: on input $x$
Example: Halting

The following machine computes this function $f$.

$F = \text{on input } \langle M, w \rangle$: 

- Construct the following machine $M'$.
  - $M'$: on input $x$
    - run $M$ on $x$
Example: Halting

The following machine computes this function \( f \).
\[
F = \text{on input } \langle M, w \rangle:
\]

- Construct the following machine \( M' \).

\( M' \): on input \( x \)
- run \( M \) on \( x \)
- If \( M \) accepts, \( \text{accept} \).
Example: Halting

The following machine computes this function $f$. $F =$ on input $\langle M, w \rangle$:

- Construct the following machine $M'$.
  
  $M'$: on input $x$
  
  - run $M$ on $x$
  
  - If $M$ accepts, accept.
  
  - if $M$ rejects, enter a loop.
Example: Halting

The following machine computes this function $f$. $F = \text{on input } \langle M, w \rangle$:

- Construct the following machine $M'$. $M'$: on input $x$
  - run $M$ on $x$
  - If $M$ accepts, accept.
  - if $M$ rejects, enter a loop.
- output $\langle M', w \rangle$
Enumerability

**Theorem:** If $A \leq_m B$ and $B$ is enumerable, then $A$ is enumerable.

Proof is same as before, using accepters instead of deciders.
Enumerability

**Corollary:** If $A \leq_m B$ and $A$ is not enumerable, then $B$ is not enumerable.
TM Equality

**Theorem:** Both $\text{EQ}_\text{TM}$ and its complement, $\overline{\text{EQ}_\text{TM}}$, are not enumerable. Stated differently, $\text{EQ}_\text{TM}$ is neither enumerable nor co-enumerable.
TM Equality

**Theorem:** Both $\text{EQ}_{\text{TM}}$ and its complement, $\overline{\text{EQ}_{\text{TM}}}$, are not enumerable. Stated differently, $\text{EQ}_{\text{TM}}$ is neither enumerable nor co-enumerable.

We show that $A_{\text{TM}}$ is reducible to $\text{EQ}_{\text{TM}}$. The same function is also a mapping reduction from $A_{\text{TM}}$ to $\overline{\text{EQ}_{\text{TM}}}$, and thus $\overline{\text{EQ}_{\text{TM}}}$ is not enumerable.
TM Equality

**Theorem:** Both $\text{EQ}_{\text{TM}}$ and its complement, $\overline{\text{EQ}_{\text{TM}}}$, are not enumerable. Stated differently, $\text{EQ}_{\text{TM}}$ is neither enumerable nor co-enumerable.

- We show that $A_{\text{TM}}$ is reducible to $\text{EQ}_{\text{TM}}$. The *same function* is also a mapping reduction from $A_{\text{TM}}$ to $\overline{\text{EQ}_{\text{TM}}}$, and thus $\overline{\text{EQ}_{\text{TM}}}$ is *not* enumerable.

- We then show that $A_{\text{TM}}$ is reducible to $\overline{\text{EQ}_{\text{TM}}}$. The *new function* is also a mapping reduction from $A_{\text{TM}}$ to $\text{EQ}_{\text{TM}}$, and thus $\text{EQ}_{\text{TM}}$ is *not* enumerable.
TM Equality

Claim: \( A_{TM} \) is reducible to \( EQ_{TM} \).

\( f : A_{TM} \rightarrow EQ_{TM} \) works as follows:

\( F: \) On input \( \langle M, w \rangle \)

- Construct machine \( M_1 \): on any input, reject.
TM Equality

Claim: $A_{TM}$ is reducible to $EQ_{TM}$.

$f : A_{TM} \longrightarrow EQ_{TM}$ works as follows:

$F$: On input $\langle M, w \rangle$

- Construct machine $M_1$: on any input, reject.
- Construct machine $M_2$: on input $x$, run $M$ on $w$. If it accepts, accept.
TM Equality

Claim: $A_{TM}$ is reducible to $\overline{EQ_{TM}}$.

$f : A_{TM} \rightarrow \overline{EQ_{TM}}$ works as follows:

$F$: On input $\langle M, w \rangle$

- Construct machine $M_1$: on any input, reject.
- Construct machine $M_2$: on input $x$, run $M$ on $w$. If it accepts, accept.
- Output $\langle M_1, M_2 \rangle$. 
TM Equality

$F$: On input $\langle M, w \rangle$

- Construct machine $M_1$: on any input, reject.
- Construct machine $M_2$: on any input $x$, run $M$ on $w$.
  If it accepts, accept $x$.
- Output $\langle M_1, M_2 \rangle$.

Note

- $M_1$ accepts nothing
TM Equality

\[ F: \text{On input } \langle M, w \rangle \]

- Construct machine \( M_1 \): on any input, reject.
- Construct machine \( M_2 \): on any input \( x \), run \( M \) on \( w \).
  
  If it accepts, accept \( x \).

- Output \( \langle M_1, M_2 \rangle \).

Note

- \( M_1 \) accepts nothing
- if \( M \) accepts \( w \) then \( M_2 \) accepts everything, and otherwise nothing.
**TM Equality**

$F$: On input $\langle M, w \rangle$

- Construct machine $M_1$: on any input, reject.
- Construct machine $M_2$: on any input $x$, run $M$ on $w$.
  
  If it accepts, accept $x$.

- Output $\langle M_1, M_2 \rangle$.

**Note**

- $M_1$ accepts nothing
- if $M$ accepts $w$ then $M_2$ accepts everything, and otherwise nothing.

- so $\langle M, w \rangle \in A_{TM} \iff \langle M_1, M_2 \rangle \in \overline{EQ_{TM}}$
TM Equality

Claim: $A_{TM}$ is reducible to $EQ_{TM}$.

$f : A_{TM} \longrightarrow EQ_{TM}$ works as follows:

$F$: On input $\langle M, w \rangle$

- Construct machine $M_1$: on any input, *accept*.
TM Equality

Claim: $A_{TM}$ is reducible to $EQ_{TM}$.

$f : A_{TM} \longrightarrow EQ_{TM}$ works as follows:

$F$: On input $\langle M, w \rangle$
  
  - Construct machine $M_1$: on any input, accept.
  - Construct machine $M_2$: on any input $x$, run $M$ on $w$.

  If it accepts, accept.
**TM Equality**

Claim: $A_{TM}$ is reducible to $EQ_{TM}$.

$f : A_{TM} \rightarrow EQ_{TM}$ works as follows:

$F$: On input $\langle M, w \rangle$

- Construct machine $M_1$: on any input, *accept*.
- Construct machine $M_2$: on any input $x$, run $M$ on $w$. If it accepts, *accept*.
- Output $\langle M_1, M_2 \rangle$. 
TM Equality

\( F \): On input \( \langle M, w \rangle \)

- Construct machine \( M_1 \): on any input, accept.
- Construct machine \( M_2 \): on any input \( x \), run \( M \) on \( w \).
  
  If it accepts, accept.

- Output \( \langle M_1, M_2 \rangle \).

Note

- \( M_1 \) accepts everything
TM Equality

\( F \): On input \( \langle M, w \rangle \)
- Construct machine \( M_1 \): on any input, accept.
- Construct machine \( M_2 \): on any input \( x \), run \( M \) on \( w \).
  - If it accepts, accept.
- Output \( \langle M_1, M_2 \rangle \).  

Note
- \( M_1 \) accepts everything
- if \( M \) accepts \( w \), then \( M_2 \) accepts everything, and otherwise nothing.
TM Equality

**F**: On input \( \langle M, w \rangle \)

- Construct machine \( M_1 \): on any input, accept.
- Construct machine \( M_2 \): on any input \( x \), run \( M \) on \( w \).
  
  If it accepts, accept.

- Output \( \langle M_1, M_2 \rangle \).

**Note**

- \( M_1 \) accepts everything
- If \( M \) accepts \( w \), then \( M_2 \) accepts everything, and otherwise nothing.

\[ \langle M, w \rangle \in A_{TM} \iff \langle M_1, M_2 \rangle \in EQ_{TM}. \]
Recursive Inseparability

Two disjoint languages $L_1$ and $L_2$ are recursively inseparable if there is no decidable language $D$ such that

$$L_1 \cap D = \emptyset,$$

and
Recursive Inseparability

Two disjoint languages $L_1$ and $L_2$ are recursively inseparable if there is no decidable language $D$ such that

1. $L_1 \cap D = \emptyset$, and
2. $L_2 \subset D$. 
Recursive Inseparability

Two disjoint languages $L_1$ and $L_2$ are recursively inseparable if there is no decidable language $D$ such that

- $L_1 \cap D = \emptyset$, and
- $L_2 \subset D$.

Example of recursively separable languages:
Recursive Inseparability

$A_{TM}$ and $\overline{A}_{TM}$ are a trivial example.
Recursive Inseparability

$A_{TM}$ and $\overline{A}_{TM}$ are a trivial example.

Why?
Recursive Inseparability

$A_{TM}$ and $\overline{A}_{TM}$ are a trivial example.

Why?

Are there non-trivial examples?
Recursive Inseparability

Define

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

and

\[ A_{\text{no}} = \{ \langle M \rangle | M \text{ is a TM that halts and rejects } \langle M \rangle \} \]

**Theorem:** \(A_{\text{yes}}\) and \(A_{\text{no}}\) are recursively inseparable.
Proof by Contradiction

Let $D$ be a decidable language that separates them.
Proof by Contradiction

- Let $D$ be a decidable language that separates them.
- Assume $A_{\text{no}} \subset D$ and $D \cap A_{\text{yes}} = \emptyset$.

Let $M_D$ be the TM that decides $D$.

What does $M_D$ do with input $\langle M_D \rangle$?

- It must halt.
  - If $M_D$ accepts $\langle M_D \rangle$: $\langle M_D \rangle \in A_{\text{yes}} \land \langle M_D \rangle \notin D$ so $M_D$ rejects $\langle M_D \rangle$.
  - If $M_D$ rejects $\langle M_D \rangle$: $\langle M_D \rangle \in A_{\text{no}} \land \langle M_D \rangle \in D$ so $M_D$ accepts $\langle M_D \rangle$.
Proof by Contradiction

Let $D$ be a decidable language that separates them. Assume $A_{\text{no}} \subset D$ and $D \cap A_{\text{yes}} = \emptyset$. Let $M_D$ be the TM that decides $D$. 

\begin{itemize}
  \item If $M_D$ accepts $\langle M_D \rangle$: $\langle M_D \rangle \in A_{\text{yes}} \land \langle M_D \rangle \notin D$ so $M_D$ rejects $\langle M_D \rangle$.
  \item If $M_D$ rejects $\langle M_D \rangle$: $\langle M_D \rangle \in A_{\text{no}} \land \langle M_D \rangle \in D$ so $M_D$ accepts $\langle M_D \rangle$.
\end{itemize}

♣

Slides modified by Benny Chor, based on original slides by Maurice Herlihy, Brown University.
Proof by Contradiction

- Let $D$ be a decidable language that separates them.
- Assume $A_{\text{no}} \subset D$ and $D \cap A_{\text{yes}} = \emptyset$.
- Let $M_D$ be the TM that decides $D$
- What does $M_D$ do with input $\langle M_D \rangle$?

[Proof details and logic follow here]
Proof by Contradiction

- Let $D$ be a decidable language that separates them.
- Assume $A_{\text{no}} \subset D$ and $D \cap A_{\text{yes}} = \emptyset$.
- Let $M_D$ be the TM that decides $D$.
- What does $M_D$ do with input $\langle M_D \rangle$?
- It must halt. (why?)
Proof by Contradiction

- Let $D$ be a decidable language that separates them.
- Assume $A_{\text{no}} \subset D$ and $D \cap A_{\text{yes}} = \emptyset$.
- Let $M_D$ be the TM that decides $D$
- What does $M_D$ do with input $\langle M_D \rangle$?
- It must halt. (why?)
- If $M_D$ accepts $\langle M_D \rangle$:
Proof by Contradiction

- Let $D$ be a decidable language that separates them.
- Assume $A_{no} \subset D$ and $D \cap A_{yes} = \emptyset$.
- Let $M_D$ be the TM that decides $D$
- What does $M_D$ do with input $\langle M_D \rangle$?
- It must halt. (why?)
- If $M_D$ accepts $\langle M_D \rangle$:
  - $\langle M_D \rangle \in A_{yes}$
Proof by Contradiction

Let $D$ be a decidable language that separates them.
Assume $A_{\text{no}} \subset D$ and $D \cap A_{\text{yes}} = \emptyset$.
Let $M_D$ be the TM that decides $D$.
What does $M_D$ do with input $\langle M_D \rangle$?
It must halt. (why?)
If $M_D$ accepts $\langle M_D \rangle$:
- $\langle M_D \rangle \in A_{\text{yes}}$
- $\langle M_D \rangle \notin D$
Proof by Contradiction

- Let $D$ be a decidable language that separates them.
- Assume $A_{\text{no}} \subset D$ and $D \cap A_{\text{yes}} = \emptyset$.
- Let $M_D$ be the TM that decides $D$.
- What does $M_D$ do with input $\langle M_D \rangle$?
  - It must halt. (why?)
- If $M_D$ accepts $\langle M_D \rangle$:
  - $\langle M_D \rangle \in A_{\text{yes}}$
  - $\langle M_D \rangle \notin D$
  - so $M_D$ rejects $\langle M_D \rangle$.
Proof by Contradiction

- Let $D$ be a decidable language that separates them.
- Assume $A_{\text{no}} \subset D$ and $D \cap A_{\text{yes}} = \emptyset$.
- Let $M_D$ be the TM that decides $D$
- What does $M_D$ do with input $\langle M_D \rangle$?
- It must halt. (why?)
- If $M_D$ accepts $\langle M_D \rangle$:
  - $\langle M_D \rangle \in A_{\text{yes}}$
  - $\langle M_D \rangle \not\in D$
  - so $M_D$ rejects $\langle M_D \rangle$.
- If $M_D$ rejects $\langle M_D \rangle$:
Proof by Contradiction

Let $D$ be a decidable language that separates them.

Assume $A_{\text{no}} \subset D$ and $D \cap A_{\text{yes}} = \emptyset$.

Let $M_D$ be the TM that decides $D$

What does $M_D$ do with input $\langle M_D \rangle$?

It must halt. (why?)

If $M_D$ accepts $\langle M_D \rangle$:

$\langle M_D \rangle \in A_{\text{yes}}$

$\langle M_D \rangle \notin D$

so $M_D$ rejects $\langle M_D \rangle$.

If $M_D$ rejects $\langle M_D \rangle$:

$\langle M_D \rangle \in A_{\text{no}}$
Proof by Contradiction

- Let $D$ be a decidable language that separates them.
- Assume $A_{no} \subset D$ and $D \cap A_{yes} = \emptyset$.
- Let $M_D$ be the TM that decides $D$.
- What does $M_D$ do with input $\langle M_D \rangle$?
- It must halt. (why?)
- If $M_D$ accepts $\langle M_D \rangle$:
  - $\langle M_D \rangle \in A_{yes}$
  - $\langle M_D \rangle \notin D$
  - so $M_D$ rejects $\langle M_D \rangle$.
- If $M_D$ rejects $\langle M_D \rangle$:
  - $\langle M_D \rangle \in A_{no}$
  - $\langle M_D \rangle \in D$
Proof by Contradiction

- Let $D$ be a decidable language that separates them.
- Assume $A_{no} \subset D$ and $D \cap A_{yes} = \emptyset$.
- Let $M_D$ be the TM that decides $D$.
- What does $M_D$ do with input $\langle M_D \rangle$?
- It must halt. (why?)
- If $M_D$ accepts $\langle M_D \rangle$:
  - $\langle M_D \rangle \in A_{yes}$
  - $\langle M_D \rangle \notin D$
  - so $M_D$ rejects $\langle M_D \rangle$.
- If $M_D$ rejects $\langle M_D \rangle$:
  - $\langle M_D \rangle \in A_{no}$
  - $\langle M_D \rangle \in D$
  - so $M_D$ accepts $\langle M_D \rangle$.  

Slides modified by Benny Chor, based on original slides by Maurice Herlihy, Brown University.
Recursive Inseparability

Define

\[ B_{yes} = \{ \langle M \rangle | M \text{ is a TM that accepts } \varepsilon \} \]

and

\[ B_{no} = \{ \langle M \rangle | M \text{ is a TM that halts and rejects } \varepsilon \} \]

Theorem: \( B_{yes} \) and \( B_{no} \) are recursively inseparable.

Proof by reduction and contradiction.
Recursive Inseparability

**Theorem:** $B_{yes}$ and $B_{no}$ are recursively inseparable.
By reduction and contradiction.

Assume $B_{yes}$ and $B_{no}$ can be separated by $E$, decided by TM $M_E$. 
Recursive Inseparability

**Theorem:** \( B_{yes} \) and \( B_{no} \) are recursively inseparable.

By reduction and contradiction.

- Assume \( B_{yes} \) and \( B_{no} \) can be separated by \( E \), decided by TM \( M_E \).

- For TM \( M \), define \( M' \): On any input,
Recursive Inseparability

**Theorem:** $B_{\text{yes}}$ and $B_{\text{no}}$ are recursively inseparable.

By reduction and contradiction.

- Assume $B_{\text{yes}}$ and $B_{\text{no}}$ can be separated by $E$, decided by TM $M_E$.
- For TM $M$, define $M'$: On any input,
  1. run $M$ on input $\langle M \rangle$. 
Recursive Inseparability

Theorem: \( B_{\text{yes}} \) and \( B_{\text{no}} \) are recursively inseparable.

By reduction and contradiction.

- Assume \( B_{\text{yes}} \) and \( B_{\text{no}} \) can be separated by \( E \), decided by TM \( M_E \).
- For TM \( M \), define \( M' \): On any input,
  1. run \( M \) on input \( \langle M \rangle \).
  2. if \( M \) accepts, accept; if \( M \) rejects, reject;
Proof (Concluded)

- Define $N$: On input $\langle M \rangle$,
Proof (Concluded)

Define $N$: On input $\langle M \rangle$,
1. construct description of $M'$. 
Proof (Concluded)

Define $N$: On input $\langle M \rangle$,
1. construct description of $M'$.
2. run $M_E$ on $\langle M' \rangle$.
Proof (Concluded)

Define $N$: On input $\langle M \rangle$,
1. construct description of $M'$.
2. run $M_E$ on $\langle M' \rangle$.
3. if $M_E$ accepts, accept; if $M_E$ rejects, reject;

Claim: $N$ is a decider. (why?)

So $N$ decides a language $D$ separates $A_{\text{yes}}$ and $A_{\text{no}}$, contradiction.

♣

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

- Define $N$: On input $\langle M \rangle$,
  1. construct description of $M'$.
  2. run $M_E$ on $\langle M' \rangle$.
  3. if $M_E$ accepts, accept; if $M_E$ rejects, reject;

- Claim:
Proof (Concluded)

Define $N$: On input $\langle M \rangle$,
1. construct description of $M'$.
2. run $M_E$ on $\langle M' \rangle$.
3. if $M_E$ accepts, accept; if $M_E$ rejects, reject;

Claim:
$N$ is a decider. (why?)
Proof (Concluded)

Define $N$: On input $\langle M \rangle$,
1. construct description of $M'$.
2. run $M_E$ on $\langle M' \rangle$.
3. if $M_E$ accepts, accept; if $M_E$ rejects, reject;

Claim:
- $N$ is a decider. (why?)
- So $N$ decides a language $D$. 
Proof (Concluded)

- Define $N$: On input $\langle M \rangle$,
  1. construct description of $M'$.
  2. run $M_E$ on $\langle M' \rangle$.
  3. if $M_E$ accepts, accept; if $M_E$ rejects, reject;

- Claim:
  - $N$ is a decider. (why?)
  - So $N$ decides a language $D$.
  - $D$ separates $A_{\text{yes}}$ and $A_{\text{no}}$, contradiction. ♣