

CS 341: Foundations of CS II

Marvin K. Nakayama
Computer Science Department
New Jersey Institute of Technology
Newark, NJ 07102

Chapter 5 Reducibility

Contents

- Reducing One Problem to Another
- Examples of Undecidable Problems
- Mapping Reducibility

Introduction

- Previously, we saw
 - Church-Turing Thesis
 - Many problems are solvable using TMs
 - One problem, A_{TM} , is unsolvable by TMs, where
$$A_{TM} = \{ \langle M, w \rangle \mid M \text{ is a TM that accepts string } w \}$$
- We now will see many other computationally unsolvable problems.
- We will do this by using **reductions**.
- **Example:**

Finding your way around a city
reduces to
obtaining a city map.

Reducibility

- Reduction always involves two problems, A and B .
- **Definition:** If A **reduces** to B , then any solution of B solves A .
- **Remarks:**
 - We showed that A_{NFA} is decidable by reducing A_{NFA} to A_{DFA} .
 - Definition of reduction says nothing about solving A or B alone.
 - If A is reducible to B , then A cannot be harder than B .
 - The statement " $p \Rightarrow q$ " is equivalent to " $\neg q \Rightarrow \neg p$ ".
 - Suppose A reduces to B . Then
 - ▲ If I can solve B , then I can solve A .
 - ▲ Equivalently, if I can't solve A , then I can't solve B .
 - ▲ Equivalently, if A is undecidable, then B is undecidable.

Reducibility

- It required some effort to prove that A_{TM} is not decidable.
- But now we can build on this result as follows:
 - To show another language L is undecidable, we typically show A_{TM} reduces to L .
 - If “language L is decidable” implies “ A_{TM} is decidable,” then L is not decidable.
- Typical approach to show L is undecidable via reduction from A_{TM} to L
 - Suppose L is decidable.
 - Let R be a TM that decides L .
 - Using R as subroutine,
 - ▲ can construct another TM S that decides A_{TM} .
 - But A_{TM} is not decidable.
 - Conclusion: L is not decidable.

Halting Problem for TMs is Undecidable

- Recall that A_{TM} (acceptance problem for TMs) is undecidable, where

$$A_{TM} = \{ \langle M, w \rangle \mid M \text{ is a TM and } M \text{ accepts string } w \}.$$
- Define related problem:

$$HALT_{TM} = \{ \langle M, w \rangle \mid M \text{ is a TM and } M \text{ halts on string } w \}.$$
- We are considering the universe of all possible Turing machines M and strings w .
- Given a particular Turing machine M and string w ,
 - if M halts on input w , then $\langle M, w \rangle \in HALT_{TM}$,
 - if M doesn't halt on input w , then $\langle M, w \rangle \notin HALT_{TM}$.
- How does $HALT_{TM}$ differ from A_{TM} ?

Theorem 5.1

$HALT_{TM}$ is undecidable.

Basic Idea of Proof that $HALT_{TM}$ is Undecidable

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

$$HALT_{TM} = \{ \langle M, w \rangle \mid M \text{ is a TM and } M \text{ halts on string } w \}.$$

Basic idea of proof by contradiction is to reduce A_{TM} to $HALT_{TM}$:

- Suppose \exists TM R that decides $HALT_{TM}$.
- How could we use R to construct TM to decide A_{TM} ?
- Recall universal TM U recognizes A_{TM} :

$$U = \text{“On input } \langle M, w \rangle, \text{ where } M \text{ is a TM and } w \text{ is a string:}$$
 1. Simulate M on input w .
 2. If M ever enters its accept state, *accept*;
if M ever enters its reject state, *reject*.”
- U doesn't decide A_{TM} since M may loop on w in stage 1.
- Solution: first run R on $\langle M, w \rangle$ to see if it's safe to run M on w .

Proof that $HALT_{TM}$ is Undecidable

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

$$HALT_{TM} = \{ \langle M, w \rangle \mid M \text{ is a TM and } M \text{ halts on string } w \}.$$

- Assume \exists TM R that decides $HALT_{TM}$.
- Define TM S to decide A_{TM} using TM R as follows:

$$S = \text{“On input } \langle M, w \rangle, \text{ where } M \text{ is a TM and } w \text{ a string:}$$
 1. Run R on input $\langle M, w \rangle$.
 2. If R rejects, *reject*.
 3. If R accepts, simulate M on input w until it halts.
 4. If M accepts, *accept*; otherwise, *reject*.”
- TM S always halts and decides A_{TM}
 - S accepts $\langle M, w \rangle \in A_{TM}$, and S rejects $\langle M, w \rangle \notin A_{TM}$.
- Thus, deciding A_{TM} is reduced to deciding $HALT_{TM}$.
- However, A_{TM} is undecidable, so $HALT_{TM}$ must also be undecidable.

Emptiness Problem for TMs is Undecidable

- Does a TM M accept no strings at all?

$$E_{\text{TM}} = \{ \langle M \rangle \mid M \text{ is a TM and } L(M) = \emptyset \}.$$
- Considering universe of all Turing machines. For a specific TM M ,
 - if M accepts at least one string, then $\langle M \rangle \notin E_{\text{TM}}$,
 - if M accepts no strings, then $\langle M \rangle \in E_{\text{TM}}$.

Theorem 5.2

E_{TM} is undecidable.

Proof Idea: Reduce A_{TM} to E_{TM} .

- Suppose E_{TM} is decidable.
- Let R be a TM that decides E_{TM} .
- Use TM R to construct another TM S that decides A_{TM} .
- But since A_{TM} is undecidable, E_{TM} must also be.

Constructing Decider S for A_{TM} From Decider R for E_{TM}

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

- **Bad Idea:** When S receives input $\langle M, w \rangle$, it calls R with input $\langle M \rangle$.
 - If R accepts, then $L(M) = \emptyset$.
 - ▲ In particular, M does not accept w , so S *rejects*.
 - If R rejects, then $L(M) \neq \emptyset$, so M accepts at least one string.
 - ▲ But don't know if M accepts w , so TM S can't decide A_{TM} .
- **Fix:** Create another TM M_1 from TM M and w as follows:
 $M_1 =$ "On input x :
 1. If $x \neq w$, *reject*.
 2. If $x = w$, run M on input w , and *accept* if M accepts."
 - w is the only string M_1 might accept, so one of 2 cases occurs:
 - ▲ $L(M_1) = \{w\}$, which occurs when M accepts w ; or
 - ▲ $L(M_1) = \emptyset$, which occurs when M does not accept w .

Proving E_{TM} is Undecidable

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

- Suppose \exists TM R that decides E_{TM} .
- Define TM S using TM R as follows:
 $S =$ "On input $\langle M, w \rangle$, where M is a TM and w is a string:
 1. Construct TM M_1 from M and w (as described before).
 2. Run R on input $\langle M_1 \rangle$.
 3. If R accepts, *reject*; if R rejects, *accept*."

- Note that

$$\begin{aligned} \langle M_1 \rangle \notin E_{\text{TM}} &\iff L(M_1) \neq \emptyset \\ &\iff M \text{ accepts } w \\ &\iff \langle M, w \rangle \in A_{\text{TM}}. \end{aligned}$$

- But then TM S decides A_{TM} , which is undecidable.
- Therefore, TM R cannot exist.

TM Recognizing Regular Language is Undecidable

- Does a TM M recognize a regular language?

$$REG_{\text{TM}} = \{ \langle M \rangle \mid M \text{ is a TM and } L(M) \text{ is a regular language} \}$$
- Considering universe of all Turing machines. For a specific TM M ,
 - if $L(M)$ is regular, then $\langle M \rangle \in REG_{\text{TM}}$,
 - if $L(M)$ is nonregular, then $\langle M \rangle \notin REG_{\text{TM}}$.

Theorem 5.3

REG_{TM} is undecidable.

Proof Idea: Reduce A_{TM} to REG_{TM} .

- Assume REG_{TM} is decidable.
- Let R be a TM that decides REG_{TM} .
- Use TM R to construct TM S that decides A_{TM} .
- But how do we do this?

Constructing Decider S for A_{TM} from Decider R for REG_{TM}

$REG_{TM} = \{ \langle M \rangle \mid M \text{ is a TM and } L(M) \text{ is a regular language} \}$.

- TM S is given input $\langle M, w \rangle$.
- TM S first constructs a TM M' using $\langle M, w \rangle$ so that TM M' recognizes a regular language if and only if M accepts w .
 - If M does not accept w , then M' recognizes language

$$\{ 0^n 1^n \mid n \geq 0 \},$$
 which is nonregular.
 - If M accepts w , then M' recognizes language Σ^* , which is regular.
- We construct M' as follows:
 - M' automatically accepts all strings in $\{ 0^n 1^n \mid n \geq 0 \}$.
 - In addition, if M accepts w , then M' accepts all other strings.

Proof that REG_{TM} is Undecidable

- Suppose that REG_{TM} is decidable.
- Let R be a TM that decides REG_{TM} .
- Use R to construct TM S to decide A_{TM} :

$S =$ "On input $\langle M, w \rangle$, where M is a TM and w is a string:

 1. Construct following TM M' from M and w :

$$M' = \text{"On input } x:$$
 1. If $x \in \{ 0^n 1^n \mid n \geq 0 \}$, *accept*.
 2. If $x \notin \{ 0^n 1^n \mid n \geq 0 \}$, run M on input w and *accept* if M accepts w ."
 2. Run R on input $\langle M' \rangle$.
 3. If R accepts, *accept*; if R rejects, *reject*."
- $\langle M' \rangle \in REG_{TM} \iff \langle M, w \rangle \in A_{TM}$, so S decides A_{TM} , which is impossible.

Equivalence of 2 TMs is Undecidable

- Do 2 TMs recognize the same language?

$$EQ_{TM} = \{ \langle M_1, M_2 \rangle \mid M_1, M_2 \text{ are TMs and } L(M_1) = L(M_2) \}.$$
- We are considering the universe of all possible pairs of Turing machines.
- For any particular pair M_1 and M_2 ,
 - if $L(M_1) = L(M_2)$, then $\langle M_1, M_2 \rangle \in EQ_{TM}$,
 - if $L(M_1) \neq L(M_2)$, then $\langle M_1, M_2 \rangle \notin EQ_{TM}$.

Theorem 5.4

EQ_{TM} is undecidable.

Proof that EQ_{TM} is Undecidable

- Recall

$$EQ_{TM} = \{ \langle M_1, M_2 \rangle \mid M_1, M_2 \text{ are TMs and } L(M_1) = L(M_2) \}.$$
- Reduce E_{TM} to EQ_{TM} as follows:
 - Let $M_2 = M_\emptyset$ be a TM with $L(M_\emptyset) = \emptyset$.
 - A TM that decides EQ_{TM} can also decide E_{TM} by deciding if $\langle M_1, M_\emptyset \rangle \in EQ_{TM}$.

$$\blacktriangle \langle M_1 \rangle \in E_{TM} \iff \langle M_1, M_\emptyset \rangle \in EQ_{TM}$$
- Since E_{TM} is undecidable (Theorem 5.2), EQ_{TM} must be undecidable.
- We'll see later that EQ_{TM} is
 - not Turing-recognizable
 - not co-Turing-recognizable

Other Undecidable Problems

- Does a TM recognize a finite language?
- Does a TM recognize a context-free language?
- Does a TM recognize a decidable language?
- Does a TM halt on all inputs?
- Does a TM have a state that is never entered on any input string?

Rice's Theorem.

- *Informally:* Every non-trivial property \mathcal{P} of languages of Turing machines is undecidable.
- *Formally:* Let \mathcal{P} be a language consisting of TM descriptions such that
 1. \mathcal{P} contains some, but not all, TM descriptions, and
 2. whenever $L(M_1) = L(M_2)$, we have $\langle M_1 \rangle \in \mathcal{P}$ iff $\langle M_2 \rangle \in \mathcal{P}$.
 Then \mathcal{P} is undecidable.

Proof of Rice's Theorem: Reduce A_{TM} to \mathcal{P}

- Suppose \mathcal{P} is decided by TM $R_{\mathcal{P}}$.
- Let T_{\emptyset} be a TM that always rejects, so $L(T_{\emptyset}) = \emptyset$.
- Without loss of generality, assume $\langle T_{\emptyset} \rangle \notin \mathcal{P}$. (Otherwise, consider $\overline{\mathcal{P}}$.)
- Because we assumed \mathcal{P} is nontrivial, \exists TM T with $\langle T \rangle \in \mathcal{P}$.
- Now design TM S to decide A_{TM} using $R_{\mathcal{P}}$'s ability to distinguish between T_{\emptyset} and T .

$S =$ "On input $\langle M, w \rangle$, where M is a TM and w a string:

1. Use M and w to construct the following TM M_w :

$M_w =$ "On input x :

1. Simulate M on input w . If it halts and rejects, *reject*.
 2. Simulate T on input x . If it accepts, *accept*."
2. Use TM $R_{\mathcal{P}}$ to determine whether $\langle M_w \rangle \in \mathcal{P}$.
If YES, *accept*. If NO, *reject*."

Proof of Rice's Theorem: Reduce A_{TM} to \mathcal{P} (cont.)

- Note that TM M_w simulates T if M accepts w .
- Hence,
 - $L(M_w) = L(T)$ if M accepts w ,
 - $L(M_w) = \emptyset$ if M does not accept w .
- Therefore, $\langle M_w \rangle \in \mathcal{P}$ iff M accepts w .
- Hence, S decides A_{TM} , which is impossible since A_{TM} is undecidable.
- Thus, \mathcal{P} is undecidable.

Limited Success Thus Far

- Our reductions have been straightforward:
 - Transform TM for some language into a similar TM that decides another language
- As a result, the languages we proved are undecidable are similar:
 - A_{TM} , EQ_{TM} , $HALT_{\text{TM}}$, etc.
- For languages concerning questions not about TMs, we have to use a different approach.
 - e.g., Hilbert's 10th problem
- Recall interpretation of TM configuration:

1011 q_7 01

 - current state is q_7
 - LHS of tape is 1011, and RHS of tape is 01
 - tape head is on RHS 0

Computation Histories

Definition: An **accepting computation history** for a TM M on a string w is a sequence of configurations

$$C_1, C_2, \dots, C_k$$

for some $k \geq 1$ such that the following properties hold:

1. C_1 is the start configuration of M on w .
2. Each C_j yields C_{j+1} .
3. C_k is an accepting configuration.

Definition: A **rejecting computation history** for M on w is the same except last configuration C_k is a rejecting configuration of M .

Remarks About Computation Histories

- Computation histories are finite.
- If M does not halt on w ,
 - then no accepting or rejecting computation history exists.
- Useful for both
 - deterministic TMs (one history)
 - nondeterministic TMs (many histories).
- “ $\langle M, w \rangle \notin A_{\text{TM}}$ ” is equivalent to
 - “ \nexists accepting computation history C_1, \dots, C_k for M on w ”
 - “All histories C_1, \dots, C_k are non-accepting ones for M on w ”.

Context-Free Languages

Does a CFG generate all strings over Σ ?

$$ALL_{\text{CFG}} = \{ \langle G \rangle \mid G \text{ is a CFG with } L(G) = \Sigma^* \}.$$

Theorem 5.13

ALL_{CFG} is undecidable.

Proof Idea: (see Sipser for full proof)

- Approach: Reduce A_{TM} to ALL_{CFG} .
- Construct a CFG G from TM M and input w .
 - If M does not accept w , then G generates all strings.
 - If M accepts w , then G generates all strings **except** the accepting computation histories for M on w .
- CFG G generates all strings iff TM M does not accept w .

Mapping Reducibility

- Thus far, we have seen several ways to reduce one problem to another.
- Reductions appear in
 - decidability theory
 - complexity theory (as we'll see later).
- Now we want to formalize the notion of reducibility.

Computable Functions

- Suppose we have 2 languages A and B , where
 - A is defined over alphabet Σ_1
 - B is defined over alphabet Σ_2
- Informally speaking, A is reducible to B if we can use a “black box” for B to build an algorithm for A .

- **Definition:** A function

$$f : \Sigma_1^* \rightarrow \Sigma_2^*$$

is a **computable function** if some TM M , on every input $w \in \Sigma_1^*$, halts with just $f(w) \in \Sigma_2^*$ on its tape.

- All the usual computations are all computable:
 - Addition, multiplication, sorting, minimization, etc.

Computable Functions

One useful class of computable functions transforms one TM into another.

Example:

$T =$ “On input w :

1. If $w = \langle M \rangle$, where M is some TM,
 - Construct $\langle M' \rangle$, where M' is a TM such that
 - $L(M') = L(M)$, but
 - M' never tries to move tape head off LHS of tape.”

Mapping Reducibility

Definition: Suppose

- A is language over alphabet Σ_1
- B is language over alphabet Σ_2 .

Then A is **mapping reducible** to B , written

$$A \leq_m B$$

if there is a computable function

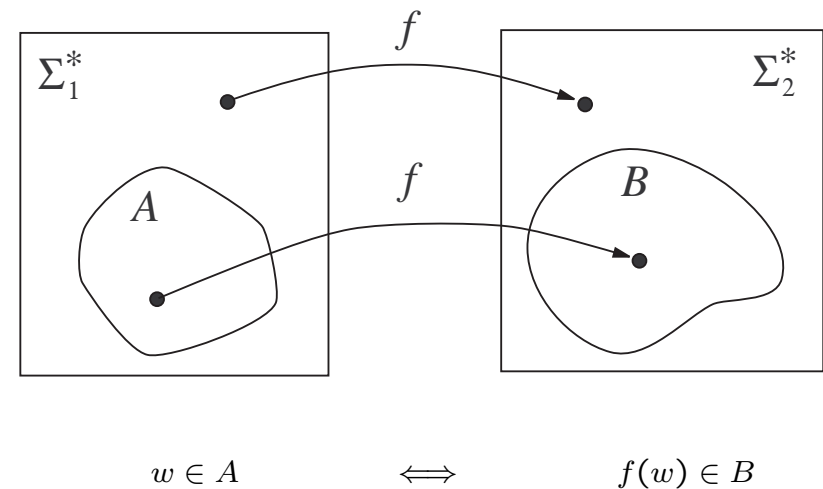
$$f : \Sigma_1^* \rightarrow \Sigma_2^*$$

such that, for every $w \in \Sigma_1^*$,

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

The function f is called a **reduction** of A to B .
(f is also called a **many-one reduction**.)

Language A is Mapping Reducible to B



Example: Formally Reducing A_{TM} to $HALT_{TM}$

- Recall that

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

$$HALT_{TM} = \{ \langle M, w \rangle \mid M \text{ is a TM that halts on string } w \}.$$

- We previously proved that
 - $HALT_{TM}$ is undecidable
 - by using reduction from A_{TM} .
- Let's now explicitly show the reduction by specifying a computable function f that reduces A_{TM} to $HALT_{TM}$.
- Want computable function f with
 - input $\langle M, w \rangle$
 - output $\langle M', w' \rangle$
 - where $\langle M, w \rangle \in A_{TM} \iff \langle M', w' \rangle \in HALT_{TM}$.

Example: Formally Reducing A_{TM} to $HALT_{TM}$

The following TM F computes this function f .

$F =$ "On input $\langle M, w \rangle$, where M is a TM and w is a string:

- Construct the following TM M' :

$M' =$ "On input x :

- Run M on input x .
- If M accepts, *accept*.
- If M rejects, enter a loop."

- Output $\langle M', w \rangle$."

Note that $\langle M, w \rangle \in A_{TM} \iff \langle M', w \rangle \in HALT_{TM}$.

Decidability obeys \leq_m Ordering**Theorem 5.22**

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

Proof.

- Let M be a TM that decides B .
- Let f be a reducing function from A to B .
- Consider the following TM:

$M' =$ "On input w :

 - Compute $f(w)$.
 - Run M on input $f(w)$ and give the same result."
- Since f is a reducing function, $w \in A \iff f(w) \in B$.
 - If $w \in A$, then $f(w) \in B$, so M and M' accept.
 - If $w \notin A$, then $f(w) \notin B$, so M and M' reject.
- Thus, M' decides A .

Undecidability obeys \leq_m Ordering**Corollary 5.23**

If $A \leq_m B$ and A is undecidable, then B is undecidable also.

Proof. Language A undecidable and B decidable contradicts the previous theorem.

Recall: Complements $\bar{A} = \Sigma_1^* - A$ and $\bar{B} = \Sigma_2^* - B$.

Fact: If $A \leq_m B$, then $\bar{A} \leq_m \bar{B}$.

Proof.

- Let f be the reducing function of A to B with

$$w \in A \iff f(w) \in B.$$
- This same function f shows $\bar{A} \leq_m \bar{B}$ since

$$w \in \bar{A} \iff f(w) \in \bar{B}.$$

Recognizability and \leq_m

Theorem 5.28

If $A \leq_m B$ and B is Turing-recognizable, then A is Turing-recognizable.

Proof.

- Let M be a TM that recognizes B
- Let f be a reducing function from A to B .
- Define a new TM as follows:

$M' =$ "On input w :

 1. Compute $f(w)$.
 2. Run M on input $f(w)$ and give the same result."
- Since f is a reducing function, $w \in A \iff f(w) \in B$.
 - If $w \in A$, then $f(w) \in B$, so M and M' accept.
 - If $w \notin A$, then $f(w) \notin B$, so M and M' reject or loop.
- Thus, M' recognizes A .

Unrecognizability and \leq_m

Corollary 5.29

If $A \leq_m B$ and A is not Turing-recognizable, then B is not Turing-recognizable.

Proof. Language A not Turing-recognizable and B Turing-recognizable contradicts the previous theorem.

Fact: If $A \leq_m B$ and A is not co-Turing-recognizable, then B is not co-Turing-recognizable.

Proof.

- If A is not co-Turing-recognizable, then its complement \bar{A} is not Turing-recognizable.
- $A \leq_m B$ implies $\bar{A} \leq_m \bar{B}$ (see slide 5-33).
- Previous corollary implies \bar{B} is not Turing-recognizable.
- Hence, B is not co-Turing-recognizable.

E_{TM} is not Turing-recognizable

Recall: the emptiness problem for TMs

$$E_{\text{TM}} = \{ \langle M \rangle \mid M \text{ is a TM with } L(M) = \emptyset \}.$$

Proof (E_{TM} is not Turing-recognizable):

- Use reduction $\overline{A_{\text{TM}}} \leq_m E_{\text{TM}}$ as follows.
- Define reducing function $f(\langle M, w \rangle) = \langle M' \rangle$, where M' is following TM:

$M' =$ "On input x :

 1. Ignore input x , and run M on input w .
 2. If M accepts w , then *accept*."
- Note that $L(M') = \Sigma^*$ if M accepts w , and $L(M') = \emptyset$ if M does not accept w .
- Thus, $\langle M, w \rangle \in \overline{A_{\text{TM}}} \iff f(\langle M, w \rangle) = \langle M' \rangle \in E_{\text{TM}}$.
- Corollary 5.29 implies E_{TM} not TM-recognizable since $\overline{A_{\text{TM}}}$ also isn't.

Theorem 5.30: EQ_{TM} is not Turing-recognizable

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

Proof. Reduce $\overline{A_{\text{TM}}} \leq_m EQ_{\text{TM}}$.

- Define reducing function $f(\langle M, w \rangle) = \langle M_1, M_2 \rangle$, where
 - $M_1 =$ "reject on all inputs."
 - $M_2 =$ "On input x :
 1. Ignore input x , and run M on w .
 2. If M accepts w , *accept*."
- We see that with this Turing-computable f :

$$\langle M, w \rangle \in \overline{A_{\text{TM}}} \iff f(\langle M, w \rangle) = \langle M_1, M_2 \rangle \in EQ_{\text{TM}}.$$
- $\overline{A_{\text{TM}}}$ not TM-recognizable (Cor. 4.23), so EQ_{TM} not TM-recognizable by Corollary 5.29.

Theorem 5.30: EQ_{TM} is not co-Turing-recognizable

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

Proof. Reduce $A_{TM} \leq_m EQ_{TM}$.

- Define reducing function $f(\langle M, w \rangle) = \langle M_1, M_2 \rangle$, where
 - $M_1 =$ “accept on all inputs.”
 - $M_2 =$ “On input x :
 1. Ignore input, and run M on w .
 2. If M accepts w , accept.”
- $\langle M, w \rangle \in A_{TM} \iff f(\langle M, w \rangle) = \langle M_1, M_2 \rangle \in EQ_{TM}$.
- Because A_{TM} is not co-Turing-recognizable, EQ_{TM} is not co-Turing-recognizable by Fact on slide 5-35.

Summary of Chapter 5

- Computable function f maps one language (one problem) to another.
- Reductions $A \leq_m B$: $w \in A \iff f(w) \in B$, for some computable function f .
 - If I can solve B , then I can solve A .
 - If I can't solve A , then I can't solve B .
- Undecidable problems: A_{TM} , $HALT_{TM}$, E_{TM} , REG_{TM} , EQ_{TM} , ALL_{CFG}
- Rice's Theorem: any nontrivial property of Turing-recognizable languages is undecidable.
- E_{TM} is not Turing-recognizable.
- EQ_{TM} is neither Turing-recognizable nor co-Turing-recognizable.