Reducibility

Contents

- Undecidable Problems from Language Theory
- Post Correspondence Problem
- Mapping Reducibility

Reducibility

• In this lecture we examine several new unsolvable problems; our main approach to demonstrating that a computational problem is unsolvable will be **reducibility**.

• A **reduction** is a way to convert one problem/domain to another problem/domain in such a way that a solution to the second problem can be applied to solve the first problem.

Reducibility

• Reducibility involves two problems: A and B; if **A reduces to B** ($A \rightarrow_{red} B$), we can use a solution to B to solve A.

• Notice that reducibility says nothing about solving A or B alone, but only about **the solvability of A in the presence of a solution to B**.

Reducibility

• Reducibility involves two problems: A and B; if **A reduces to B** $(A \rightarrow_{red} B)$, we can use a solution to B to solve A.

• Notice that reducibility says nothing about solving A or B alone, but only about **the solvability of A in the presence of a solution to B**.

For example: the problem of (uniquely) solving a square system of linear equations: $Ax = b$, reduces to finding A^{-1}; in other words if we have A^{-1} we can necessarily solve $Ax = b$.

Reducibility

• Reducibility plays an important role in classifying decidable and undecidable problems (and complexity theory).

• If $A \rightarrow_{red} B$, **solving *A* cannot be harder than solving *B* because a solution to *B* (always) gives a solution to *A***.

Reducibility

- In terms of computability, if $A \rightarrow_{red} B$ and B is decidable, **then *A* is also decidable**.

- Similarly, if $A \rightarrow_{red} B$ and A is undecidable, then ***B* is undecidable**.

Reducibility

• In terms of computability, if $A \rightarrow_{red} B$ and B is decidable, **then *A* is also decidable**.

• Similarly, if $A \rightarrow_{red} B$ and A is undecidable, then ***B* is undecidable**.

* In summary, our **general method** for proving that a problem is undecidable entails showing that some problem already known to be undecidable reduces to it.

Undecidable Problems

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

- Recall that we have already (using diagonalization) proved the undecidability of A_{TM}.

Undecidable Problems

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

• Recall that we have already (using diagonalization) proved the undecidability of A_{TM}.

• Consider now the problem $HALT_{TM}$, the problem of determining whether a TM halts (by accepting *or* rejecting) on a given input; this problem is widely known as the **Halting Problem** in computability theory.

Define:

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

Undecidable Problems

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

Theorem. $HALT_{TM}$ is undecidable.

Undecidable Problems

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

Theorem. $HALT_{TM}$ is undecidable.

Proof Sketch: (Contradiction) Assume $HALT_{TM}$ is decidable; use this assumption to show the implication that A_{TM} is consequently decidable, a contradiction.

(Proof logic: $A_{TM} \rightarrow_{red} HALT_{TM}$)

Undecidable Problems

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

Theorem. $HALT_{TM}$ is undecidable.

Proof Sketch: (Contradiction) Assume $HALT_{TM}$ is decidable; use this assumption to show the implication that A_{TM} is consequently decidable, a contradiction.

* The key step in the proof is to **explicitly use a TM (R) that decides $HALT_{TM}$ to render a TM deciding A_{TM}.**

Simply use R to decide whether M halts on input w. If it doesn't – reject; otherwise, simulate M on w and return the result of this simulation.

Undecidable Problems

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

Theorem. $HALT_{TM}$ is undecidable.

Proof. (Contradiction) Assume the TM R decides $HALT_{TM}$**.** We construct TM S to decide A_{TM}.

S = "On input $\langle M, w\rangle$, an encoding of a TM M and string w:

(1) Run TM R on input $\langle M, w\rangle$.

(2) If R rejects, **reject.**

(3) If R accepts, simulate M on w until it halts.

(4) If M has accepted, **accept**; if M has rejected, **reject**."

*Because A_{TM} is undecidable, $HALT_{TM}$ is also undecidable.

Undecidable Problems

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

Theorem. E_{TM} is undecidable.

(Proof logic: $A_{TM} \rightarrow_{red} E_{TM}$)

Undecidable Problems

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

Theorem. E_{TM} is undecidable.

Proof Sketch: (Contradiction) Assume E_{TM} is decidable; use this assumption to show the implication that A_{TM} is consequently decidable, a contradiction.

* Key step: run TM R deciding E_{TM} on a modification of $\langle M \rangle$.

We modify $\langle M \rangle$ so that M rejects all strings except w, but on input w it works as usual. Now use R on this modified TM; R will decide A_{TM}, a contradiction.

Undecidable Problems

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

Theorem. E_{TM} is undecidable.

Proof. Define the aforementioned, modified TM M_1**:**

M_1 = "On input x:

(1) If $x \neq w$, **reject**.

(2) If $x = w$, run M on input w and **accept** if M does.

Undecidable Problems

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

Theorem. E_{TM} is undecidable.

Proof. Now using R, the TM that decides E_{TM}, construct TM S that decides A_{TM} as follows:

S ="On input $\langle M, w \rangle$, an encoding of a TM M and a string w :

(1) Use the description of M and w to construct the TM M_1 just described.

(2) Run R on input $\langle M_1 \rangle$.

(3) If R accepts, **reject**; if R rejects, **accept**."

*In summary, if R were a decider for E_{TM}, then A_{TM} would be decidable, a contradiction.

Undecidable Problems

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

Theorem. $REGULAR_{TM}$ is undecidable.

* This computation problem relates to whether a given TM recognizes a language that can also be recognized by a simpler computational model.

(Proof logic: $A_{TM} \rightarrow_{red} REGULAR_{TM}$)

Undecidable Problems

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

Theorem. $REGULAR_{TM}$ is undecidable.

Proof Sketch: (again: contradiction, reduction from A_{TM}) We assume $REGULAR_{TM}$ is decidable by TM R, use this fact to show then A_{TM} would be decidable by some TM S.

Challenge: How to construct S from R? S takes input $\langle M, w \rangle$; we modify M (call it M_2) so that the resulting TM recognizes a regular language *iff* M accepts w.

Undecidable Problems

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

Theorem. $REGULAR_{TM}$ is undecidable.

Challenge: How to construct S from R? S takes input $\langle M, w \rangle$; we modify M (call it M_2) so that the resulting TM recognizes a regular language *iff* M accepts w.

We define M_2 to recognize the non-regular language $\{0^n 1^n | n \geq 0\}$ if M doesn't accept w; otherwise, M_2 accepts its input *iff* M accepts w.

In summary, M_2 works by automatically accepting all strings in $\{0^n 1^n | n \geq 0\}$. In addition, if M accepts w, M_2 accepts all other strings.

Undecidable Problems

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

Theorem. $REGULAR_{TM}$ is undecidable.

Proof. Let R be a TM that decides $REGULAR_{TM}$ and 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 the following TM M_2.

M_2 = "On input x:

(i) If x has the form $\{0^n 1^n | n \geq 0\}$, *accept*.

(ii) If x does not have this form, run M on input w and *accept* if M accepts w.

(2) Run R on input $\langle M_2 \rangle$.

(3) If R accepts, **accept**; if R rejects, **reject**.

Undecidable Problems

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

Theorem. EQ_{TM} is undecidable.

(Proof logic: $A_{TM} \rightarrow_{red} E_{TM} \rightarrow_{red} EQ_{TM}$)

Undecidable Problems

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

Theorem. EQ_{TM} is undecidable.

Proof Sketch: Recall the undecidable problem from previous slides:

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

This time, to show EQ_{TM} is undecidable, we reduce from E_{TM}. The idea is straightforward: use the fact that E_{TM} is a special case of EQ_{TM}.

Undecidable Problems

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

Theorem. EQ_{TM} is undecidable.

Proof. Suppose TM R decides EQ_{TM} **and construct TM S to decide** E_{TM} **as follows:**

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

(1) Run R on input $\langle M, M_1 \rangle$, where M_1 is the TM that rejects all inputs.

(2) If R accepts, **accept;** if R rejects, **reject.**"

Rice's Theorem

Theorem. Any nontrivial property about the language recognized by a Turing machine is undecidable.

Rice's Theorem

Theorem. Any nontrivial property about the language recognized by a Turing machine is undecidable.

• More concretely, Rice's Theorem says that **non-trivial semantic properties of Turing-recognizable languages are undecidable**.

• Here, **non-trivial** connotes the fact that the property is neither always true nor always false (for computable functions).

Put another way, for a set of languages S, then **S is non-trivial** if:

(1) There exists a TM that recognizes a language in S,

(2) There exists a TM that recognizes a language which is not in S.

• Semantic properties are properties about the behavior of a TM (*cf.* syntactic properties).

Reductions with Computation Histories

- Next, we will use the notion of a **computation history** to further prove undecidability for additional classes of computation problems.

Reductions with Computation Histories

• Next, we will use the notion of a **computation history** to further prove undecidability for additional classes of computation problems.

Def. Let M be a TM and w an input string. An accepting **computation history** for M on w is a sequence of configurations, $C_1, C_2, \ldots, C_L$, where C_1 is the *start configuration* of M, C_L is an *accepting configuration* of M, and each C_i legally follows from C_{i-1} according to the rules of M.

A **rejecting computation history** for M on w is defined analogously, except that C_L is a *rejecting configuration*.

Reductions with Computation Histories

• Next, we will use the notion of a **computation history** to further prove undecidability for additional classes of computation problems.

Def. Let M be a TM and w an input string. An accepting **computation history** for M on w is a sequence of configurations, $C_1, C_2, \ldots, C_L$, where C_1 is the *start configuration* of M, C_L is an *accepting configuration* of M, and each C_i legally follows from C_{i-1} according to the rules of M.

A **rejecting computation history** for M on w is defined analogously, except that C_L is a *rejecting configuration*.

*Note that computation histories are finite sequences; if M doesn't halt on w, no accepting or rejecting computation history exists for M on w.

*Deterministic machines have at most one computation history per input; non-deterministic machines may have many computation histories per input (corresponding with computation branches); for now, consider the machines to be deterministic for simplicity.

LBA

• We now consider TMs with a limited amount of memory; these machines can only solve problems requiring memory that can fit within the tape used for the input alphabet.

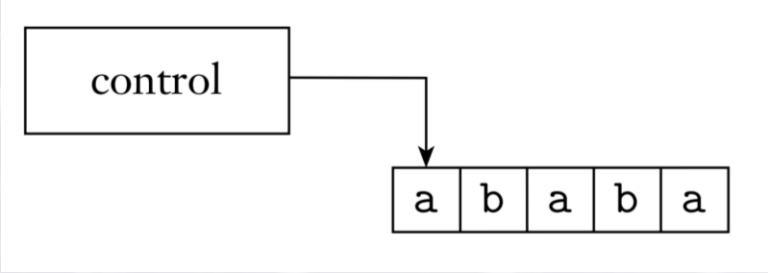
• Using a tape alphabet that is larger than the input alphabet allows the available memory to be increased up to a constant factor.

LBA

• We now consider TMs with a limited amount of memory; these machines can only solve problems requiring memory that can fit within the tape used for the input alphabet.

• Using a tape alphabet that is larger than the input alphabet allows the available memory to be increased up to a constant factor.

Def. A **linear bounded automaton (LBA)** is a restricted TM wherein the tape head isn't permitted to move off the portion of the tape containing the input (if the machine tries to move off on either end the tape head stays where it is).



LBA

• Despite their memory constraint, LBA are quite powerful.

* For example, the deciders for A_{DFA}, A_{CFG}, E_{DFA}, E_{CFG} are all LBA.

• In fact, it is not trivial to find a decidable language that can't be decided by an LBA.

• Before proving decidability/undecidability of computational problems related to LBA, we first devise a useful bound for the number of distinct configurations for LBA.

LBA

Lemma. Let M be an LBA with q states and g symbols in the tape alphabet. There are exactly qng^n distinct configurations of M for a tape of length n.

LBA

Lemma. Let M be an LBA with q states and g symbols in the tape alphabet. There are exactly qng^n distinct configurations of M for a tape of length n.

Proof. Remember that a "configuration" of a TM specifies three things: (1) current state, (2) location of tape head, and (3) contents of the tape.

Here, M has q states; the length of the tape is n, so the head can be in one of n positions; lastly, there are g^n possible tape contents. We multiply these quantities together (using the *multiplication principle* from combinatorics), yielding the desired result.

LBA Problems

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

Theorem. A_{LBA} is decidable.

LBA Problems

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

Theorem. A_{LBA} is decidable.

Proof Sketch: We simulate LBA M on w; if M halts and accepts or rejects, we accept or reject accordingly.

Otherwise, we need to detect when M is looping. Because M is an LBA, by the previous lemma, M can be in only a limited number of configurations for the input tape.

Detecting that M is looping is possible by simulating M for the number of steps prescribed by the previous lemma: qng^n.

LBA Problems

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

Theorem. A_{LBA} is decidable.

Proof. The algorithm decides A_{LBA} as follows.

$L =$ "On input $\langle M, w\rangle$, where M is an LBA and w a string:

(1) Simulate M on w for qng^n steps or until it halts.

(2) If M has halted, **accept** if it has accepted and **reject** if it has rejected. If it has not halted, **reject**.

*This result shows that LBA and TMs differ in one essential way: for LBA the acceptance problem is decidable, whereas with TMs the acceptance problem is undecidable.

*However, many other problems remain undecidable for LBA, as we show next.

Undecidable LBA Problems

$E_{LBA} = \{\langle M \rangle \mid M \text{ is an LBA where } L(M) = \emptyset\}$

Theorem. E_{LBA} is undecidable.

Undecidable LBA Problems

$$E_{LBA} = \{\langle M \rangle \mid M \text{ is an LBA where } L(M) = \emptyset\}$$

Theorem. E_{LBA} is undecidable.

Proof Sketch: The proof is by reduction from A_{TM}.

We suppose E_{LBA} is decidable. For a TM M and input w, we construct an LBA B and then test whether L(B) is empty.

The language that B recognizes comprises all accepting computation histories for M on w. If M accepts w, this language contains one string and is therefore non-empty. If M does not accept w, this language is empty.

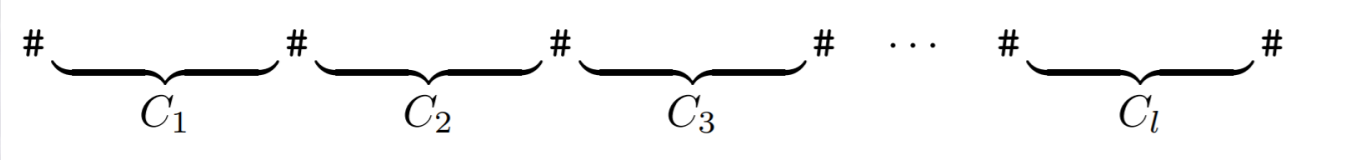
Undecidable LBA Problems

$E_{LBA} = \{\langle M \rangle \mid M \text{ is an LBA where } L(M) = \emptyset\}$

Theorem. E_{LBA} is undecidable.

Proof Sketch: We now describe how to construct B from M and w; in particular, we need to show how a TM can obtain a description of B from M and w.

Construct B to accept its input x, if x is an accepting computation history for M on w . For simplicity, assume the computation history consists of configurations delimited by #.



Undecidable LBA Problems

$$E_{LBA} = \{\langle M \rangle \mid M \text{ is an LBA where } L(M) = \emptyset\}$$

Theorem. E_{LBA} is undecidable.

Proof Sketch: The LBA B works as follows: given a computation history C_1, C_2,…, C_L, B checks whether:

(1) C_1 corresponds with the start configuration for M on w

(2) Each C_{i+1} legally follows from C_i, and

(3) C_L is the accepting configuration for M.

Undecidable LBA Problems

$E_{LBA} = \{\langle M \rangle \mid M \text{ is an LBA where } L(M) = \emptyset\}$

Theorem. E_{LBA} is undecidable.

Proof Sketch: The LBA B works as follows: given a computation history C_1, C_2,…, C_L, B checks whether:

(1) C_1 corresponds with the start configuration for M on w

(2) Each C_{i+1} legally follows from C_i, and

(3) C_L is the accepting configuration for M.

Regarding (1), the start configuration C_1 for M on w is the string $q_0 w_1 \ldots w_n$, where q_0 is the start state for M on w. B has the initial tape string, so this can be checked.

Regarding (3), B checks C_L to see if it is the accept state.

Undecidable LBA Problems

$E_{LBA} = \{\langle M \rangle \mid M \text{ is an LBA where } L(M) = \emptyset\}$

Theorem. E_{LBA} is undecidable.

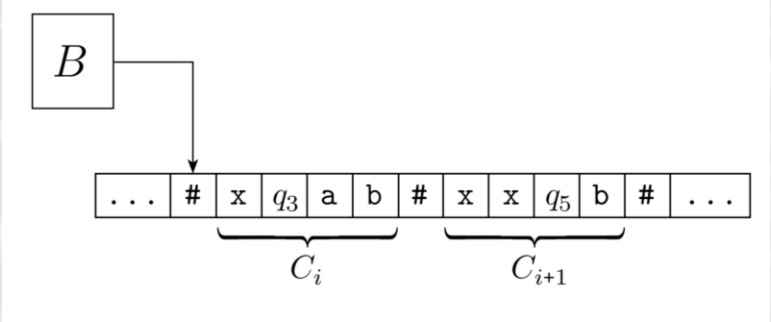
Proof Sketch: The LBA B works as follows: given a computation history C_1, C_2,…, C_L, B checks whether:

(1) C_1 corresponds with the start configuration for M on w

(2) Each C_{i+1} legally follows from C_i, and

(3) C_L is the accepting configuration for M.

Regarding (2), B checks whether C_i and C_{i+1} are identical except for the position under and adjacent to the tape head; this can be checked by following a zig-zagging procedure.



Undecidable LBA Problems

$E_{LBA} = \{\langle M \rangle \mid M \text{ is an LBA where } L(M) = \emptyset\}$

Theorem. E_{LBA} is undecidable.

Proof. Suppose that TM R decides E_{LBA}. **Construct TM S to decide A_{TM} as follows:**

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

(1) Construct LBA B from M and w as described previously.

(2) Run R on input $\langle B \rangle$.

(3) If R rejects, **accept**; if R accepts, **reject**."

*If R accepts $\langle B \rangle$, then L(B) = $\emptyset$. Thus, M has no accepting computation history on w and M doesn't accept w. Consequently, S rejects $\langle M, w \rangle$.

An Undecidable CFG Problem

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

Theorem. ALL_{CFG} is undecidable.

An Undecidable CFG Problem

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

Theorem. ALL_{CFG} is undecidable.

Proof Idea: By contradiction and reduction from A_{TM}; the proof technique is similar to that used for the previous theorem, except that we modify the representation of computation histories.

An Undecidable CFG Problem

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

Theorem. ALL_{CFG} is undecidable.

Proof Idea:

For a TM M and input w, we construct a CFG G that generates all strings *iff* M does not accept w.

So, if M does accept w, G does not generate some particular string – the string will be the accepting computation history for M on w.

* G is designed to generate all strings that are not accepting computation histories for M on w.

An Undecidable CFG Problem

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

Theorem. ALL_{CFG} is undecidable.

Proof Idea: An accepting configuration history for M on w appears as $\#C_1\# \ldots \#C_L\#$.

A string may fail to be an accepting computation history for several reasons:

(1) It doesn't start with C_1

(2) It doesn't end with an accepting configuration

(3) C_i doesn't properly yield C_{i+1} under the rules of M

If M does not accept w, no accepting computation history exists, so all strings fail in one way or another, i.e. G doesn't generate all strings.

An Undecidable CFG Problem

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

Theorem. ALL_{CFG} is undecidable.

Proof Idea: We **begin with a PDA D** (instead of directly constructing G); from previous work, we know that D can be converted into a CFG. We use a PDA to begin with, as it is easier than directly designing a CFG for this problem.

D begins **by non-deterministically guessing which of the (3) preceding conditions fail**. Conditions (1) and (2) are easy to check – where we start in C_1 and end in accept.

(1) It doesn't start with C_1

(2) It doesn't end with an accepting configuration

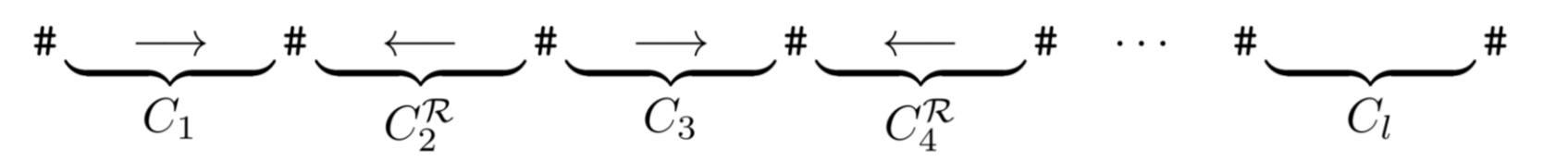
An Undecidable CFG Problem

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

Theorem. ALL_{CFG} is undecidable.

Proof Idea: For condition (3), we use the stack and compare C_i and C_{i+1} by pushing the contents of C_i until arriving at the delimiter #; next, the stack is popped and compared with C_{i+1} , a discrepancy with the location of the tape/tape-head yields an accept (recall consecutive legal states only differ by the location of the head and cells adjacent to the head).

One last detail – because the stack is processed LIFO, the computation history is written so that every other configuration is in reverse order.



Post Correspondence Problem

• Undecidability is not confined to problems concerning automata.

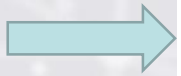
• The **Post Correspondence Problem** (after Emile Post, pictured) is another example of an undecidable problem that can be stated independent of automata.

• The problem can be described as a puzzle. Consider a collection of dominos (with a top/bottom portion), such as:

$$\left\{ \left[\frac{b}{ca}\right], \left[\frac{a}{ab}\right], \left[\frac{ca}{a}\right], \left[\frac{abc}{c}\right] \right\}.$$

• The task is to make a list of these dominos (repetition is allowed) so that the string yielded by reading off the "top" row is identical to that of the "bottom" row; such a list is called a **match**. Here is an example:

$$\left[\frac{a}{ab}\right]\left[\frac{b}{ca}\right]\left[\frac{ca}{a}\right]\left[\frac{a}{ab}\right]\left[\frac{abc}{c}\right]$$



a	b	c	a	a	a	b	c
a	b	c	a	a	a	b	c

Post Correspondence Problem

• For some collections of dominos, finding a match is impossible; here is one such example (why?):

$$\left\{\left[\frac{\texttt{abc}}{\texttt{ab}}\right], \left[\frac{\texttt{ca}}{\texttt{a}}\right], \left[\frac{\texttt{acc}}{\texttt{ba}}\right]\right\}$$

• Formally, an instance of the PCP is a collection P of dominos:

$$P = \left\{\left[\frac{t_1}{b_1}\right], \left[\frac{t_2}{b_2}\right], \cdots, \left[\frac{t_k}{b_k}\right]\right\}$$

And a match is a sequence $i_1, i_2, \ldots, i_l$ where $t_{i1}t_{i2}\ldots t_{il} = b_{i1}b_{i2}\ldots b_{il}$. The problem is to determine whether P has a match. Let:

$$PCP = \{\langle P\rangle | P \text{ is an instance of the PCP with a match}\}$$

Post Correspondence Problem

$$PCP = \{\langle P \rangle | P \text{ is an instance of the PCP with a match}\}$$

Theorem. PCP is undecidable.

Post Correspondence Problem

$$PCP = \{\langle P \rangle | P \text{ is an instance of the PCP with a match}\}$$

Theorem. PCP is undecidable.

The proof will be detailed in **(7) parts**, including a running example.

First, for convenience and simplification we apply (3) minor modifications to PCP*:

* Note that each of these requirements can be eliminated and thus PCP can be shown to be undecidable in its original form (this proof is however less concise and straightforward).

Post Correspondence Problem

$$PCP = \{\langle P \rangle | P \text{ is an instance of the PCP with a match}\}$$

Theorem. PCP is undecidable.

The proof will be detailed in **(7) parts**, including a running example.

First, for convenience and simplification we apply (3) minor modifications to PCP:

(1) We assume M on w never attempts to move the tape head off the left-hand end of the tape.

(2) If $w = \varepsilon$, we use the blank symbol ⊔ in place of w in the cor$\left[\frac{t_1}{b_1}\right]$ion.

(3) We modify PCP to require that a match starts with the first domino:

Call this modified problem the **Modified Post Correspondence Problem** (MPCP):

$$MPCP = \left\{ \begin{matrix} \langle P \rangle | P \text{ is an instance of the PCP} \\ \text{with a match that stars with the first domino} \end{matrix} \right\}$$

Post Correspondence Problem

Theorem. PCP is undecidable.

Proof. (Contradiction/reduction) We let TM R decide the PCP and construct S deciding A_{TM}.

Let M = $(Q, \sum, \Gamma, \delta, q_0, q_{accept}, q_{reject})$.

Key insight: S will construct an instance of the PCP P that has a match iff M accepts w.

To this end, S first constructs an instance P' of MPCP; next, we describe the construction in (7) parts, each of which accomplishes a particular aspect of simulating M on w.

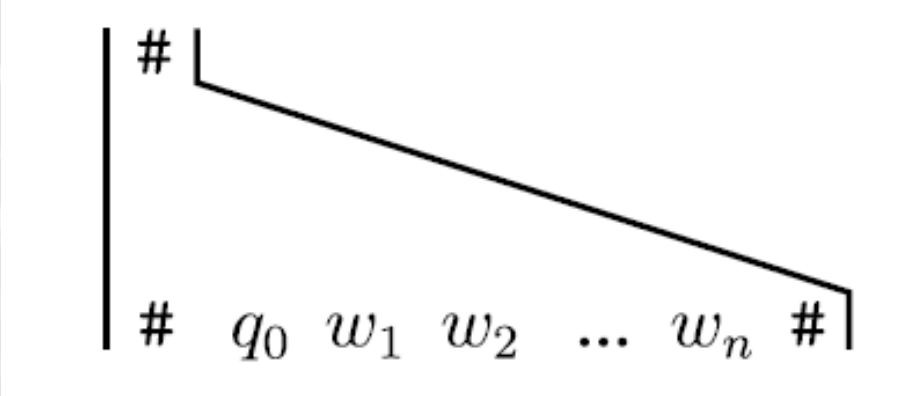
Post Correspondence Problem

Theorem. PCP is undecidable.

Proof. Part 1: The construction begins as follows:

Put $\left[\frac{\#}{\#q_0 w_1 w_2 \cdots w_n \#}\right]$ into P' as the first domino $\left[\frac{t_1}{b_1}\right]$

Recall that we are constructing the computation history on M; accordingly, C_1 is given by:



Post Correspondence Problem

Theorem. PCP is undecidable.

Proof. (Part 2 handles head motions to the right)

Part 2: For every $a, b \in \Gamma$ and every $q, r \in Q$ where $q \neq q_{reject}$,

$$\text{If } \delta(q,a) = (r,b,R), put \left[\frac{qa}{br}\right] \text{ into } P'$$

:

Post Correspondence Problem

Theorem. PCP is undecidable.

Proof. (Part 2 handles head motions to the right; Part 3 to the left)

Part 2: For every $a, b \in \Gamma$ and every $q, r \in Q$ where $q \neq q_{reject}$,

$$\text{If } \delta(q,a) = (r,b,R), put \left[\frac{qa}{br}\right] \text{ into } P'$$

Part 3: For every $a, b, c \in \Gamma$ and every $q, r \in Q$ where $q \neq q_{reject}$,

$$\text{If } \delta(q,a) = (r,b,L), put \left[\frac{cqa}{rcb}\right] \text{ into } P'$$

Post Correspondence Problem

Theorem. PCP is undecidable.

Proof. (Part 4 handles adding any tape symbol)

Part 4: For every $a \in \Gamma$,

$$put \left[\frac{a}{a}\right] \text{ into } P'$$

Post Correspondence Problem

Theorem. PCP is undecidable.

Proof. (Part 4 handles adding any tape symbol)

Part 4: For every $a \in \Gamma$,

$$put \left[\frac{a}{a}\right] \text{ into } P'$$

Part 5:

$$put \left[\frac{\#}{\#}\right] and \left[\frac{\#}{\sqcup\#}\right] \text{ into } P'$$

*Note that the first domino allows us to copy the # symbol that marks the separation of the configurations; the second domino allows us to add a blank symbol at the end of the configuration to simulate the end of the input tape.

Post Correspondence Problem

Proof. Recapping, Put $\left[\frac{\#}{\#q_0 w_1 w_2 \cdots w_n \#}\right]$ into P' as the first domino $\left[\frac{t_1}{b_1}\right]$

If $\delta(q,a) = (r,b,R), put\ \left[\frac{qa}{br}\right]$ into P' $\ put\ \left[\frac{a}{a}\right]$ into P'

If $\delta(q,a) = (r,b,L), put\ \left[\frac{cqa}{rcb}\right]$ into P' $\ put\ \left[\frac{\#}{\#}\right]\ and\ \left[\frac{\#}{\sqcup\#}\right]$ into P'

Example: Let Γ={0,1,2,⊔}; say $w = 0100$ and that the start state of M is q0. In state q_0 upon reading a 0, suppose the transition dictates that M enter state q_7, writes a 2 on the tape, and moves to the right.

Part 1: The match begins…

$$\left[\frac{\#}{\#q_0 0100\#}\right]$$

Post Correspondence Problem

Proof. Recapping,

If $\delta(q,a) = (r,b,R), put \left[\frac{qa}{br}\right]$ into P' $put \left[\frac{a}{a}\right]$ into P'

If $\delta(q,a) = (r,b,L), put \left[\frac{cqa}{rcb}\right]$ into P' $put \left[\frac{\#}{\#}\right]$ $and \left[\frac{\#}{\sqcup\#}\right]$ into P'

Example: Let Γ={0,1,2,⊔}; say $w = 0100$ and that the start state of M is q0. In state q_0 upon reading a 0, suppose the transition dictates that M enter state q_7, writes a 2 on the tape, and moves to the right.

Part 2: Since $\delta(q_0, 0) = (q_7, 2, R)$, we place the domino

$$\left[\frac{q_0 0}{2 q_7}\right]$$

Part 4: Place the dominos

$$\left[\frac{0}{0}\right], \left[\frac{1}{1}\right], \left[\frac{2}{2}\right] \text{ and } \left[\frac{\sqcup}{\sqcup}\right]$$

Post Correspondence Problem

Proof. Example: Let Γ={0,1,2,⊔}; say $w = 0100$ and that the start state of M is q0.In state q_0 upon reading a 0, suppose the transition dictates that M enter state q_7, writes a 2 on the tape, and moves to the right.

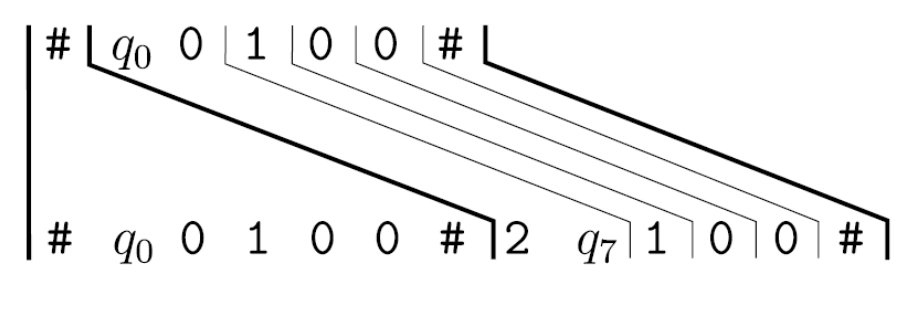
Part 2: Since $\delta(q_0, 0) = (q_7, 2, R)$, we place the domino

$$\left[\frac{q_0 0}{2 q_7}\right]$$

Part 4: Place the dominos

$$\left[\frac{0}{0}\right], \left[\frac{1}{1}\right], \left[\frac{2}{2}\right] \text{ and } \left[\frac{\sqcup}{\sqcup}\right]$$

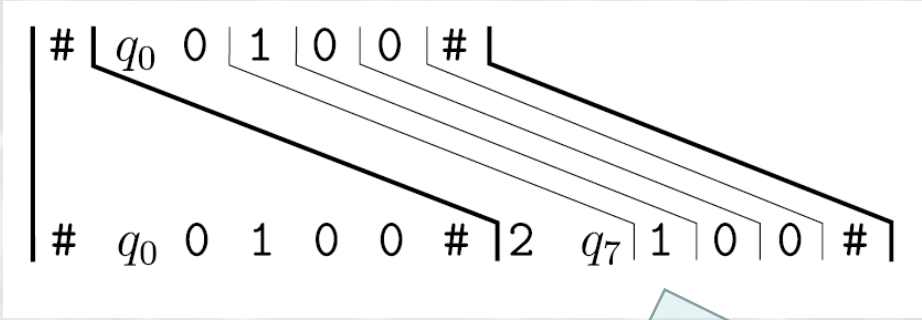
Together, with Part 5 we can extend the match as follows:



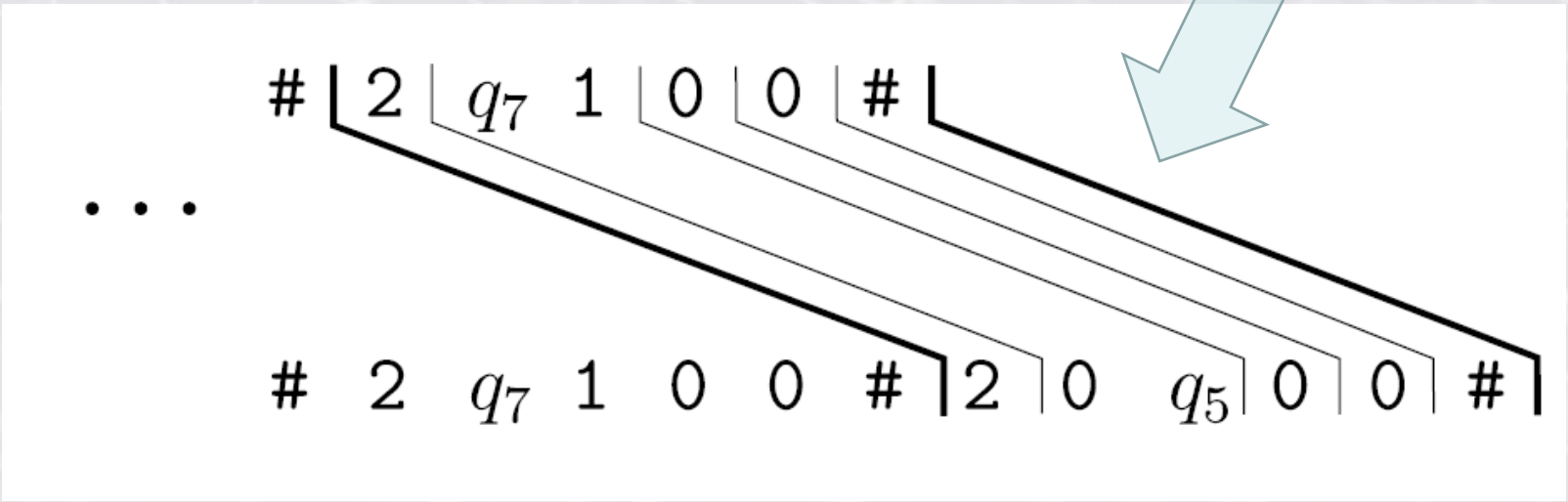
* Notice that the dominos in parts 2, 3, and 4 allow us to extend the match by adding a configuration.

Post Correspondence Problem

Proof. Continuing the example…

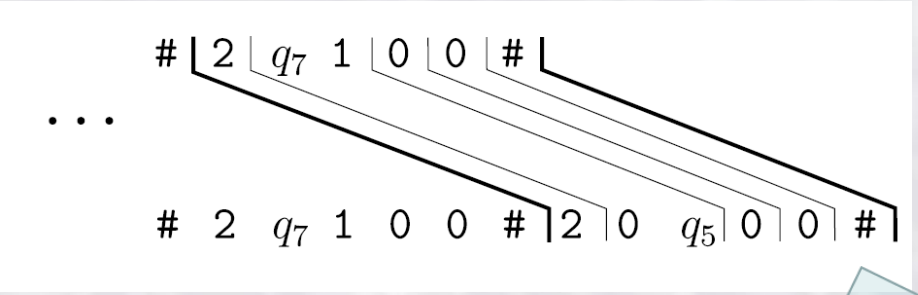


Suppose that $\delta(q_7, 1) = (q_5, 0, R)$, also. This allows us to extend the partial match as follows:

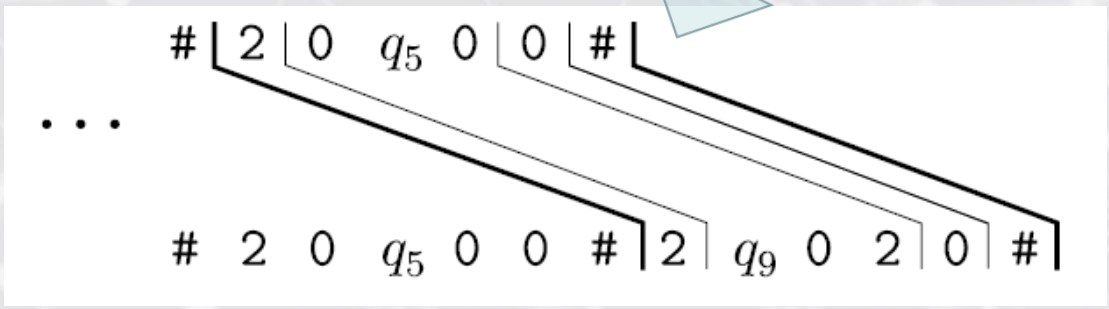


Post Correspondence Problem

Proof. Further continuing the example…



Suppose that in state q_5, upon reading a 0, M goes to state q_9, writes a 2, and moves its head to the left. So $\delta(q_5, 0) = (q_9, 2, L)$, also. Then we have the dominos: $\left[\frac{0q_50}{q_902}\right]$, $\left[\frac{010}{q_912}\right]$, $\left[\frac{2q_50}{q_922}\right]$ and $\left[\frac{\sqcup q_50}{q_9\sqcup 2}\right]$



*Notice that as we construct a match, we are forced to simulate M on input w. This process continues until M reaches a halting state.

Mapping Reducibility

- We now formalize the notion of **reducibility**.

- Roughly speaking, being able to reduce problem A to problem B means that a *computable function* exists that converts instances of problem A to instances of problem B.

Mapping Reducibility

- We now formalize the notion of **reducibility**.

- Roughly speaking, being able to reduce problem A to problem B means that a *computable function* exists that converts instances of problem A to instances of problem B.

Def. Computable Functions

A function $f: \Sigma^* \rightarrow \Sigma^*$ is a computable function if some Turing machine M, on every input w, halts with just $f(w)$ on its tape.

Mapping Reducibility

A function $f: \Sigma^* \rightarrow \Sigma^*$ is a **computable function** if some Turing machine M, on every input w, halts with just $f(w)$ on its tape.

Example: For arithmetic operations, the input $\langle m, n \rangle$ would map to $\mathrm{m} + \mathrm{n}$.

Example: A computable function may serve to generation transformations of machine descriptions.

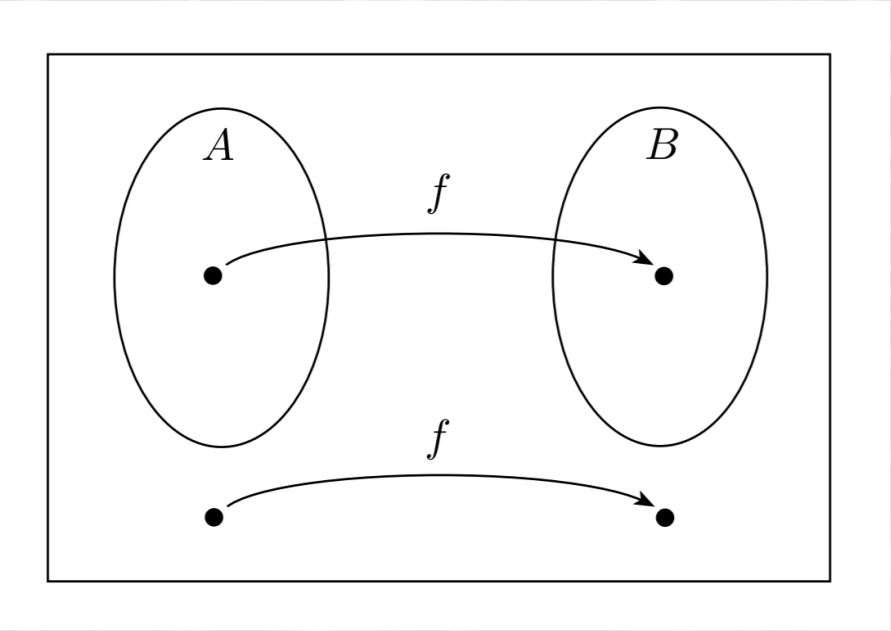
For instance, the computable function f could take input w and return the description of a TM $\langle M \rangle$ if $\mathrm{w} = \langle M \rangle$ is an encoding of a TM M.

Mapping Reducibility

Language A is **mapping reducible** to language B, written $A \leq_m B$, if there is a computable function $f: \Sigma^* \to \Sigma^*$, where for every w,

$$w \in A \leftrightarrow f(w) \in B$$

The function f is called the **reduction** from A to B.



Mapping Reducibility

Language A is **mapping reducible** to language B, written $A \leq_m B$, if there is a computable function $f: \Sigma^* \to \Sigma^*$, where for every w,

$$w \in A \leftrightarrow f(w) \in B$$

The function f is called the **reduction** from A to B.

- A mapping reduction of A to B provides a way to convert questions about membership testing in A to membership testing in B.
- To test whether $w \in A$, we use the reduction f to map w to $f(w)$ and test whether $f(w) \in B$.
- **Key point**: If one problem is mapping reducible to a second, previously solved problem, we can thereby obtain a solution to the original problem.

Mapping Reducibility

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

Proof.

Let M be the decider for B and f be the reduction from A to B. We describe a decider N for A as follows:

N ="On input w:

(1) Compute $f(w)$

(2) Run M on input $f(w)$ and output whatever M outputs."

Note: Clearly, if $w \in A$, then $f(w)$ in B because f is a reduction from A to B. Thus M accepts $f(w)$ whenever $w \in A$, so N is a decider for A.

Mapping Reducibility

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

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

How to prove the corollary from the theorem?

Mapping Reducibility

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

• Previously, we showed that $HALT_{TM}$ is undecidable (from an informal reduction from A_{TM}); we now prove the result anew using mapping reducibility.

Theorem. $HALT_{TM}$ is undecidable.

Mapping Reducibility

Theorem. $HALT_{TM}$ is undecidable.

Proof. We define a *mapping reducibility* from A_{TM} to $HALT_{TM}$ below; to do so, we provide a computable function f that takes input of the form $\langle M, w \rangle$ and returns output of the form $\langle M', w' \rangle$, where:

$$\langle M, w \rangle \in A_{TM} \textit{ iff } \langle M', w' \rangle \in HALT_{TM}$$

Mapping Reducibility

• Note that mapping reducibility is said to be **sensitive to complementation**. This means that there exist cases for which a mapping reducibility: $A \leq_m B$ exists, but no such mapping reducibility exists for A to $\bar{B}$.

• One such example of this sensitivity arises in the case of A_{TM} reduced to $\overline{E_{TM}}$ (which **does** have a mapping reduction). However, **no such** mapping reduction of A_{TM} to E_{TM} exists! (Notice that both A_{TM} and E_{TM} are nevertheless undecidable).

Mapping Reducibility

- We can use mapping reducibility to show that problems are not Turing-recognizable.

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

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

* Note that these proofs are analogous to the earlier proofs for decidability/undecidability with mapping reducibility.

Mapping Reducibility

Theorem. EQ_{TM} is **neither** Turing-recognizable **nor** co-Turing-recognizable.

Idea: We know that A_{TM} is not Turing-recognizable, as shown in a previous lecture.

Notice that the definition of mapping reducibility implies that $A \leq_m B$ *iff* $\bar{A} \leq_m \bar{B}$.

To prove that B is not Turing-recognizable, we may show that $A_{TM} \leq_m \bar{B}$, as this implies $\overline{A_{TM}} \leq_m B$.

Recalling (from previous slide):

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

Mapping Reducibility

Theorem. EQ_{TM} is **neither** Turing-recognizable **nor** co-Turing-recognizable.

Proof. First, we show that EQ_{TM} is not Turing-recognizable; we use a reduction from A_{TM} to $\overline{EQ_{TM}}$.

Mapping Reducibility

Theorem. EQ_{TM} is **neither** Turing-recognizable **nor** co-Turing-recognizable.

Proof. First, we show that EQ_{TM} is not Turing-recognizable; we use a reduction from A_{TM} to $\overline{EQ_{TM}}$.

The reducing function f works as follows:

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

(1) Construct the following two machines, M_1 and M_2.

$M_1 =$ "On any input: **reject**.

$M_2 =$ On any input: Run M on w. If it accepts, **accept**."

(2) Output $\langle M_1, M_2 \rangle$.

In summary: M_1 accepts nothing; if M accepts w, M_2 accepts everything, and so the two machines are not equivalent. Conversely, if M doesn't accept w, M_2 accepts nothing, and they are equivalent. Thus f reduced A_{TM} to $\overline{EQ_{TM}}$, as desired.

Mapping Reducibility

Theorem. EQ_{TM} is **neither** Turing-recognizable **nor** co-Turing-recognizable.

Proof. Second, we show that $\overline{EQ_{TM}}$ is not Turing-recognizable; we use a reduction from A_{TM} to the **complement** of $\overline{EQ_{TM}}$, *viz.*, we show: $A_{TM} \leq_m EQ_{TM}$.

Mapping Reducibility

Theorem. EQ_{TM} is **neither** Turing-recognizable **nor** co-Turing-recognizable.

Proof. Second, we show that $\overline{EQ_{TM}}$ is not Turing-recognizable; we use a reduction from A_{TM} to the **complement** of $\overline{EQ_{TM}}$, *viz.*, we show: $A_{TM} \leq_m EQ_{TM}$.

The following TM G computed the reducing function g:

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

(1) Construct the following two machines, M_1 and M_2.

M_1 = "On any input: **accept**.

M_2 = On any input: Run M on w. If it accepts, **accept**."

Only change from proof of part 1

(2) Output $\langle M_1, M_2 \rangle$.

In summary: In g, M accepts w *iff* M_1 and M_2 are equivalent, so g is a reduction from A_{TM} to EQ_{TM}.

Fin