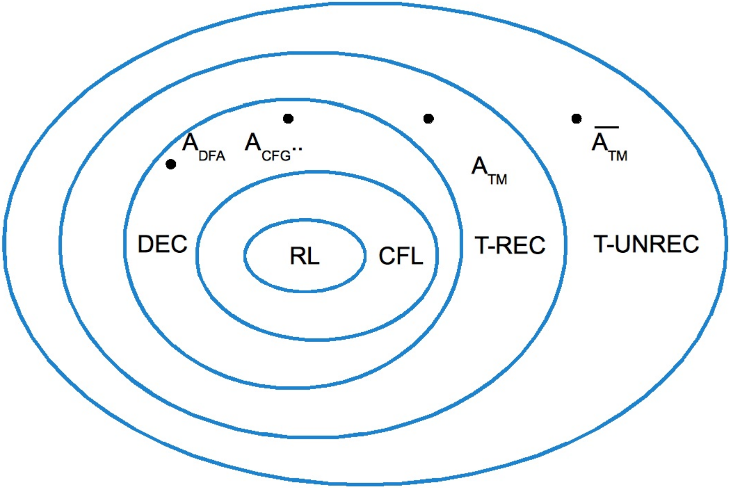
Formal Languages, Automata and Computation

Post Correspondence Problem

Review of Decidability and Reductions



REDUCIBILITY

- A reduction is a way of converting one problem to another problem, so that the solution to the second problem can be used to solve the first problem.

- A reduction is a way of converting one problem to another problem, so that the solution to the second problem can be used to solve the first problem.
  - Finding the area of a rectangle, reduces to measuring its width and height

REDUCIBILITY

- A reduction is a way of converting one problem to another problem, so that the solution to the second problem can be used to solve the first problem.
  - Finding the area of a rectangle, reduces to measuring its width and height
  - Solving a set of linear equations, reduces to inverting a matrix.

Reducibility

- A reduction is a way of converting one problem to another problem, so that the solution to the second problem can be used to solve the first problem.
  - Finding the area of a rectangle, reduces to measuring its width and height
  - Solving a set of linear equations, reduces to inverting a matrix.
- Reducibility involves two problems A and B.

Reducibility

- A reduction is a way of converting one problem to another problem, so that the solution to the second problem can be used to solve the first problem.
  - Finding the area of a rectangle, reduces to measuring its width and height
  - Solving a set of linear equations, reduces to inverting a matrix.
- Reducibility involves two problems A and B.
  - If A reduces to B, you can use a solution to B to solve A

Reducibility

- A reduction is a way of converting one problem to another problem, so that the solution to the second problem can be used to solve the first problem.
  - Finding the area of a rectangle, reduces to measuring its width and height
  - Solving a set of linear equations, reduces to inverting a matrix.
- Reducibility involves two problems A and B.
  - If A reduces to B, you can use a solution to B to solve A
- When A is reducible to B, solving A can not be "harder" than solving B.

REDUCIBILITY

- A reduction is a way of converting one problem to another problem, so that the solution to the second problem can be used to solve the first problem.
  - Finding the area of a rectangle, reduces to measuring its width and height
  - Solving a set of linear equations, reduces to inverting a matrix.
- Reducibility involves two problems A and B.
  - If A reduces to B, you can use a solution to B to solve A
- When A is reducible to B, solving A can not be "harder" than solving B.
- If A is reducible to B and B is decidable, then A is also decidable.

REDUCIBILITY

- A reduction is a way of converting one problem to another problem, so that the solution to the second problem can be used to solve the first problem.
  - Finding the area of a rectangle, reduces to measuring its width and height
  - Solving a set of linear equations, reduces to inverting a matrix.
- Reducibility involves two problems A and B.
  - If A reduces to B, you can use a solution to B to solve A
- When A is reducible to B, solving A can not be "harder" than solving B.
- If A is reducible to B and B is decidable, then A is also decidable.
- If A is undecidable and reducible to B, then B is undecidable.

THEOREM 5.2

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

Theorem 5.2

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

- Suppose R decides E_{TM}. We try to construct S to decide A_{TM} using R.

THEOREM 5.2

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

- Suppose R decides E_{TM}. We try to construct S to decide A_{TM} using R.
  - Note that S takes $\langle M, w \rangle$ as input.

PROVING UNDECIDABILITY VIA REDUCTIONS

THEOREM 5.2

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

- Suppose R decides E_{TM}. We try to construct S to decide A_{TM} using R.
  - Note that S takes $\langle M, w \rangle$ as input.
- One idea is to run R on $\langle M \rangle$ to check if M accepts some string or not – but that that does not tell us if M accepts w.

Proving Undecidability via Reductions

Theorem 5.2

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

- Suppose R decides E_{TM}. We try to construct S to decide A_{TM} using R.
  - Note that S takes $\langle M, w \rangle$ as input.
- One idea is to run R on $\langle M \rangle$ to check if M accepts some string or not – but that that does not tell us if M accepts w.
- Instead we modify M to M_1. M_1 rejects all strings other than w but on w, it does what M does.

Proving Undecidability via Reductions

Theorem 5.2

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

- Suppose R decides E_{TM}. We try to construct S to decide A_{TM} using R.
  - Note that S takes $\langle M, w \rangle$ as input.
- One idea is to run R on $\langle M \rangle$ to check if M accepts some string or not – but that that does not tell us if M accepts w.
- Instead we modify M to M_1. M_1 rejects all strings other than w but on w, it does what M does.
- Now we can check if $L(M_1) = \Phi$.

THEOREM 5.2

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

Proving Undecidability via Reductions

Theorem 5.2

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

Proof

THEOREM 5.2

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

PROOF

- For any w define M_1 as
  M_1 = “On input x:

THEOREM 5.2

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

PROOF

- For any w define M_1 as
  $M_1 =$ "On input x:
  1. If $x \neq w$, *reject*.

THEOREM 5.2

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

PROOF

- For any w define M_1 as
  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."

Proving Undecidability via Reductions

Theorem 5.2

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

Proof

- For any w define M_1 as
  $M_1 =$ "On input x:
  1. If $x \neq w$, *reject*.
  2. If $x = w$, run M on input w and *accept* if M does."
- Note that M_1 either accepts w only or nothing!

PROOF CONTINUED

PROOF CONTINUED

- Assume R decides E_{TM}

PROOF CONTINUED

- Assume R decides E_{TM}
- S defines below uses R to decide on A_{TM}
  $S =$ "On input $\langle M, w \rangle$

PROOF CONTINUED

- Assume R decides E_{TM}
- S defines below uses R to decide on A_{TM}
  $S =$ "On input $\langle M, w \rangle$
  1. Use $\langle M, w \rangle$ to construct M_1 above.

PROOF CONTINUED

- Assume R decides E_{TM}
- S defines below uses R to decide on A_{TM}
  $S =$ "On input $\langle M, w \rangle$
  1. Use $\langle M, w \rangle$ to construct M_1 above.
  2. Run R on input $\langle M_1 \rangle$

PROOF CONTINUED

- Assume R decides E_{TM}
- S defines below uses R to decide on A_{TM}
  $S =$ "On input $\langle M, w \rangle$
  1. Use $\langle M, w \rangle$ to construct M_1 above.
  2. Run R on input $\langle M_1 \rangle$
  3. If R accepts, *reject*, if R rejects, *accept*.

PROOF CONTINUED

- Assume R decides E_{TM}
- S defines below uses R to decide on A_{TM}
  $S =$ "On input $\langle M, w\rangle$
  1. Use $\langle M, w\rangle$ to construct M_1 above.
  2. Run R on input $\langle M_1\rangle$
  3. If R accepts, *reject*, if R rejects, *accept*.
- So, if R decides $L(M_1)$ is empty,

Proof continued

- Assume R decides E_{TM}
- S defines below uses R to decide on A_{TM}
  $S =$ "On input $\langle M, w\rangle$
  1. Use $\langle M, w\rangle$ to construct M_1 above.
  2. Run R on input $\langle M_1\rangle$
  3. If R accepts, *reject*, if R rejects, *accept*.
- So, if R decides $L(M_1)$ is empty,
  - then M does NOT accept w,

PROOF CONTINUED

- Assume R decides E_{TM}
- S defines below uses R to decide on A_{TM}
  $S =$ "On input $\langle M, w \rangle$
  1. Use $\langle M, w \rangle$ to construct M_1 above.
  2. Run R on input $\langle M_1 \rangle$
  3. If R accepts, *reject*, if R rejects, *accept*.
- So, if R decides $L(M_1)$ is empty,
  - then M does NOT accept w,
  - else M accepts w.

Proving Undecidability via Reductions

Proof continued

- Assume R decides E_{TM}
- S defines below uses R to decide on A_{TM}
  $S =$ "On input $\langle M, w\rangle$
  1. Use $\langle M, w\rangle$ to construct M_1 above.
  2. Run R on input $\langle M_1\rangle$
  3. If R accepts, *reject*, if R rejects, *accept*.
- So, if R decides $L(M_1)$ is empty,
  - then M does NOT accept w,
  - else M accepts w.
- If R decides E_{TM} then S decides A_{TM} – Contradiction.

- An accepting computation history for a TM is a sequence of configurations

$$C_1, C_2, \ldots, C_l$$

  such that

REDUCTIONS VIA COMPUTATION HISTORIES

- An accepting computation history for a TM is a sequence of configurations

$$C_1, C_2, \ldots, C_l$$

  such that
  - C_1 is the start configuration for input w

- An accepting computation history for a TM is a sequence of configurations

$$C_1, C_2, \ldots, C_l$$

  such that
  - C_1 is the start configuration for input w
  - C_l is an accepting configuration, and

Reductions via Computation Histories

- An accepting computation history for a TM is a sequence of configurations
$$C_1, C_2, \ldots, C_l$$
such that
  - C_1 is the start configuration for input w
  - C_l is an accepting configuration, and
  - each C_i follows legally from the preceding configuration.

Reductions via Computation Histories

- An accepting computation history for a TM is a sequence of configurations

$$C_1, C_2, \ldots, C_l$$

  such that

  - C_1 is the start configuration for input w
  - C_l is an accepting configuration, and
  - each C_i follows legally from the preceding configuration.
- A rejecting computation history is defined similarly.

Reductions via Computation Histories

- An accepting computation history for a TM is a sequence of configurations

$$C_1, C_2, \ldots, C_l$$

  such that
  - C_1 is the start configuration for input w
  - C_l is an accepting configuration, and
  - each C_i follows legally from the preceding configuration.
- A rejecting computation history is defined similarly.
- Computation histories are finite sequences – if M does not halt on M, there is no computation history.

Reductions via Computation Histories

- An accepting computation history for a TM is a sequence of configurations

$$C_1, C_2, \ldots, C_l$$

  such that
  - C_1 is the start configuration for input w
  - C_l is an accepting configuration, and
  - each C_i follows legally from the preceding configuration.
- A rejecting computation history is defined similarly.
- Computation histories are finite sequences – if M does not halt on M, there is no computation history.
- Deterministic v.s nondeterministic computation histories.

Linear Bounded Automaton

- Suppose we cripple a TM so that the head never moves outside the boundaries of the input string.

LINEAR BOUNDED AUTOMATON

- Suppose we cripple a TM so that the head never moves outside the boundaries of the input string.
- Such a TM is called a linear bounded automaton (LBA)

Linear Bounded Automaton

- Suppose we cripple a TM so that the head never moves outside the boundaries of the input string.
- Such a TM is called a linear bounded automaton (LBA)
- Despite their memory limitation, LBAs are quite powerful.

- Suppose we cripple a TM so that the head never moves outside the boundaries of the input string.
- Such a TM is called a linear bounded automaton (LBA)
- Despite their memory limitation, LBAs are quite powerful.

Lemma

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

Linear Bounded Automaton

- Suppose we cripple a TM so that the head never moves outside the boundaries of the input string.
- Such a TM is called a linear bounded automaton (LBA)
- Despite their memory limitation, LBAs are quite powerful.

Lemma

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

Proof.

□

Linear Bounded Automaton

- Suppose we cripple a TM so that the head never moves outside the boundaries of the input string.
- Such a TM is called a linear bounded automaton (LBA)
- Despite their memory limitation, LBAs are quite powerful.

Lemma

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

Proof.

- The machine can be in one of q states.

□

Linear Bounded Automaton

- Suppose we cripple a TM so that the head never moves outside the boundaries of the input string.
- Such a TM is called a linear bounded automaton (LBA)
- Despite their memory limitation, LBAs are quite powerful.

Lemma

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

Proof.

- The machine can be in one of q states.
- The head can be on one of the n cells.

□

Linear Bounded Automaton

- Suppose we cripple a TM so that the head never moves outside the boundaries of the input string.
- Such a TM is called a linear bounded automaton (LBA)
- Despite their memory limitation, LBAs are quite powerful.

Lemma

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

Proof.

- The machine can be in one of q states.
- The head can be on one of the n cells.
- At most g^n distinct strings can occur on the tape.

□

LINEAR BOUNDED AUTOMATON

- Suppose we cripple a TM so that the head never moves outside the boundaries of the input string.
- Such a TM is called a linear bounded automaton (LBA)
- Despite their memory limitation, LBAs are quite powerful.

LEMMA

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

PROOF.

- The machine can be in one of q states.
- The head can be on one of the n cells.
- At most g^n distinct strings can occur on the tape.

□

THEOREM 5.9

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

Computation over "Computation Histories"

- Now for a really wild and crazy idea!

Computation over "Computation Histories"

- Now for a really wild and crazy idea!
- Consider an accepting computation history of a TM M, $C_1, C_2, \ldots, C_l$

Computation over "Computation Histories"

- Now for a really wild and crazy idea!
- Consider an accepting computation history of a TM M, $C_1, C_2, \ldots, C_l$
- Note that each C_i is a string.

Computation over "Computation Histories"

- Now for a really wild and crazy idea!
- Consider an accepting computation history of a TM M, $C_1, C_2, \ldots, C_l$
- Note that each C_i is a string.
- Consider the string

$$\# \underbrace{\qquad\qquad}_{C_1} \# \underbrace{\qquad\qquad}_{C_2} \# \underbrace{\qquad\qquad}_{C_3} \# \cdots \# \underbrace{\qquad\qquad}_{C_l} \#$$

Computation over "Computation Histories"

- Now for a really wild and crazy idea!
- Consider an accepting computation history of a TM M, $C_1, C_2, \ldots, C_l$
- Note that each C_i is a string.
- Consider the string

$$\#\underbrace{\qquad\qquad}_{C_1}\#\underbrace{\qquad\qquad}_{C_2}\#\underbrace{\qquad\qquad}_{C_3}\#\cdots\#\underbrace{\qquad\qquad}_{C_l}\#$$

- The set of all valid accepting histories is also a language!!

- Now for a really wild and crazy idea!
- Consider an accepting computation history of a TM M, $C_1, C_2, \ldots, C_l$
- Note that each C_i is a string.
- Consider the string

$$\# \underbrace{\qquad\qquad}_{C_1} \# \underbrace{\qquad\qquad}_{C_2} \# \underbrace{\qquad\qquad}_{C_3} \# \cdots \# \underbrace{\qquad\qquad}_{C_l} \#$$

- The set of all valid accepting histories is also a language!!
- This string has length m and an LBA B can check if this is a valid computation history for a TM M accepting w.

Computation over "Computation Histories"

- Now for a really wild and crazy idea!
- Consider an accepting computation history of a TM M, $C_1, C_2, \ldots, C_l$
- Note that each C_i is a string.
- Consider the string

$$\# \underbrace{\qquad\qquad}_{C_1} \# \underbrace{\qquad\qquad}_{C_2} \# \underbrace{\qquad\qquad}_{C_3} \# \cdots \# \underbrace{\qquad\qquad}_{C_l} \#$$

- The set of all valid accepting histories is also a language!!
- This string has length m and an LBA B can check if this is a valid computation history for a TM M accepting w.
  - Check if $C_1 = q_0 w_1 w_2 \cdots w_n$

Computation over "Computation Histories"

- Now for a really wild and crazy idea!
- Consider an accepting computation history of a TM M, $C_1, C_2, \ldots, C_l$
- Note that each C_i is a string.
- Consider the string

$$\# \underbrace{\qquad\qquad}_{C_1} \# \underbrace{\qquad\qquad}_{C_2} \# \underbrace{\qquad\qquad}_{C_3} \# \cdots \# \underbrace{\qquad\qquad}_{C_l} \#$$

- The set of all valid accepting histories is also a language!!
- This string has length m and an LBA B can check if this is a valid computation history for a TM M accepting w.
  - Check if $C_1 = q_0 w_1 w_2 \cdots w_n$
  - Check if $C_l = \cdots q_{accept} \cdots$

Computation over "Computation Histories"

- Now for a really wild and crazy idea!
- Consider an accepting computation history of a TM M, $C_1, C_2, \ldots, C_l$
- Note that each C_i is a string.
- Consider the string

$$\# \underbrace{\qquad\qquad}_{C_1} \# \underbrace{\qquad\qquad}_{C_2} \# \underbrace{\qquad\qquad}_{C_3} \# \cdots \# \underbrace{\qquad\qquad}_{C_l} \#$$

- The set of all valid accepting histories is also a language!!
- This string has length m and an LBA B can check if this is a valid computation history for a TM M accepting w.
  - Check if $C_1 = q_0 w_1 w_2 \cdots w_n$
  - Check if $C_l = \cdots q_{accept} \cdots$
  - Check if each C_{i+1} follows from C_i legally.

- Now for a really wild and crazy idea!
- Consider an accepting computation history of a TM M, $C_1, C_2, \ldots, C_l$
- Note that each C_i is a string.
- Consider the string

$$\# \underbrace{\qquad\qquad}_{C_1} \# \underbrace{\qquad\qquad}_{C_2} \# \underbrace{\qquad\qquad}_{C_3} \# \cdots \# \underbrace{\qquad\qquad}_{C_l} \#$$

- The set of all valid accepting histories is also a language!!
- This string has length m and an LBA B can check if this is a valid computation history for a TM M accepting w.
  - Check if $C_1 = q_0 w_1 w_2 \cdots w_n$
  - Check if $C_l = \cdots q_{accept} \cdots$
  - Check if each C_{i+1} follows from C_i legally.
- Note that B is not constructed for the purpose of running it on any input!

- Now for a really wild and crazy idea!
- Consider an accepting computation history of a TM M, $C_1, C_2, \ldots, C_l$
- Note that each C_i is a string.
- Consider the string

$$\#\underbrace{\qquad\qquad}_{C_1}\#\underbrace{\qquad\qquad}_{C_2}\#\underbrace{\qquad\qquad}_{C_3}\#\cdots\#\underbrace{\qquad\qquad}_{C_l}\#$$

- The set of all valid accepting histories is also a language!!
- This string has length m and an LBA B can check if this is a valid computation history for a TM M accepting w.
  - Check if $C_1 = q_0 w_1 w_2 \cdots w_n$
  - Check if $C_l = \cdots q_{accept} \cdots$
  - Check if each C_{i+1} follows from C_i legally.
- Note that B is not constructed for the purpose of running it on any input!
- If $L(B) \neq \Phi$ then M accepts w

Post Correspondence Problem

- Undecidability is not just confined to problems concerning automata and languages.

Post Correspondence Problem

- Undecidability is not just confined to problems concerning automata and languages.
- There are other "natural" problems which can be proved undecidable.

Post Correspondence Problem

- Undecidability is not just confined to problems concerning automata and languages.
- There are other "natural" problems which can be proved undecidable.
- The Post correspondence problem (PCP) is a tiling problem over strings.

Post Correspondence Problem

- Undecidability is not just confined to problems concerning automata and languages.
- There are other "natural" problems which can be proved undecidable.
- The Post correspondence problem (PCP) is a tiling problem over strings.
- A tile or a domino contains two strings, t and b; e.g., $\left[\frac{ca}{a}\right]$.

- Undecidability is not just confined to problems concerning automata and languages.
- There are other “natural” problems which can be proved undecidable.
- The Post correspondence problem (PCP) is a tiling problem over strings.
- A tile or a domino contains two strings, t and b; e.g., $\left[\frac{ca}{a}\right]$.
- Suppose we have dominos

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

Post Correspondence Problem

- Undecidability is not just confined to problems concerning automata and languages.
- There are other "natural" problems which can be proved undecidable.
- The Post correspondence problem (PCP) is a tiling problem over strings.
- A tile or a domino contains two strings, t and b; e.g., $\left[\frac{ca}{a}\right]$.
- Suppose we have dominos

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

- A match is a list of these dominos so that when concatenated the top and the bottom strings are identical. For example,

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

POST CORRESPONDENCE PROBLEM

- Undecidability is not just confined to problems concerning automata and languages.
- There are other "natural" problems which can be proved undecidable.
- The Post correspondence problem (PCP) is a tiling problem over strings.
- A tile or a domino contains two strings, t and b; e.g., $\left[\frac{ca}{a}\right]$.
- Suppose we have dominos

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

- A match is a list of these dominos so that when concatenated the top and the bottom strings are identical. For example,

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

- The set of dominos $\left\{ \left[\frac{abc}{ab}\right], \left[\frac{ca}{a}\right], \left[\frac{acc}{ba}\right], \right\}$ does not have a solution.

Post Correspondence Problem

An instance of the PCP

A PCP instance over Σ is a finite collection P of dominos

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

where for all $i, 1 \leq i \leq k, t_i, b_i \in \Sigma^*$.

An instance of the PCP

A PCP instance over Σ is a finite collection P of dominos

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

where for all i, $1 \leq i \leq k$, $t_i, b_i \in \Sigma^*$.

Match

Given a PCP instance P, a match is a nonempty sequence

$$i_1, i_2, \ldots, i_\ell$$

of numbers from $\{1, 2, \ldots, k\}$ (with repetition) such that
$t_{i_1} t_{i_2} \cdots t_{i_\ell} = b_{i_1} b_{i_2} \cdots b_{i_\ell}$

Post Correspondence Problem

Question:

Does a given PCP instance P have a match?

Post Correspondence Problem

Question:

Does a given PCP instance P have a match?

Language formulation:

$PCP = \{\langle P \rangle \mid P \text{ is a PCP instance and it has a match}\}$

Post Correspondence Problem

Question:

Does a given PCP instance P have a match?

Language formulation:

$PCP = \{\langle P \rangle \mid P \text{ is a PCP instance and it has a match}\}$

Theorem 5.15

PCP is undecidable.

POST CORRESPONDENCE PROBLEM

QUESTION:

Does a given PCP instance P have a match?

LANGUAGE FORMULATION:

$PCP = \{\langle P \rangle \mid P \text{ is a PCP instance and it has a match}\}$

THEOREM 5.15

PCP is undecidable.

Proof: By reduction using computation histories. If PCP is decidable then so is A_{TM}. That is, if PCP has a match, then M accepts w.

PCP – The Structure of the Undecidability Proof

The reduction works in two steps:

1. We reduce A_{TM} to Modified PCP (MPCP).

PCP – The Structure of the Undecidability Proof

The reduction works in two steps:

1. We reduce A_{TM} to Modified PCP (MPCP).
2. We reduce MPCP to PCP.

PCP – The Structure of the Undecidability Proof

The reduction works in two steps:

1. We reduce A_{TM} to Modified PCP (MPCP).
2. We reduce MPCP to PCP.

MPCP as a language problem

$MPCP = \{\langle P\rangle \mid P$ is a PCP instance and it has a match which starts with index 1$\}$

PCP – The Structure of the Undecidability Proof

The reduction works in two steps:

1. We reduce A_{TM} to Modified PCP (MPCP).
2. We reduce MPCP to PCP.

MPCP as a language problem

$MPCP = \{\langle P \rangle \mid P$ is a PCP instance and it has a match which starts with index 1$\}$

- So the solution to MPCP starts with the domino $\left[\frac{t_1}{b_1}\right]$. We later remove this restriction in the second part of the proof.

PCP – The Structure of the Undecidability Proof

The reduction works in two steps:

1. We reduce A_{TM} to Modified PCP (MPCP).
2. We reduce MPCP to PCP.

MPCP as a language problem

$MPCP = \{\langle P\rangle \mid P$ is a PCP instance and it has a match which starts with index 1$\}$

- So the solution to MPCP starts with the domino $\left[\frac{t_1}{b_1}\right]$. We later remove this restriction in the second part of the proof.
- We also assume that the decider for M never moves its head to the left of the input w.

PCP – The Proof

For input $\langle M, w \rangle$ of A_{TM}, construct an MPCP instance such that M accepts w iff P' has a match starting with domino 1

PCP – THE PROOF

For input $\langle M, w\rangle$ of A_{TM}, construct an MPCP instance such that M accepts w iff P' has a match starting with domino 1

- The first part of the proof proceeds in 7 stages where we add different types of dominos to P' depending on the TM $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$.

PCP – The Proof

For input $\langle M, w \rangle$ of A_{TM}, construct an MPCP instance such that M accepts w iff P' has a match starting with domino 1

- The first part of the proof proceeds in 7 stages where we add different types of dominos to P' depending on the TM $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$.
- Using the dominos, we try to construct an accepting computation history for M accepting w.

PCP – ADDING THE RIGHT KIND OF DOMINOS

1. The first domino kicks of the computation history

$$\left[\frac{t_1}{b_1}\right] = \left[\frac{\#}{\#q_0 w_1 w_2 \cdots w_n \#}\right],$$

1. The first domino kicks of the computation history

$$\left[\frac{t_1}{b_1}\right] = \left[\frac{\#}{\#q_0 w_1 w_2 \cdots w_n \#}\right],$$

2. Handle right moving transitions. For every $a, b \in \Gamma$ and every $q, r \in Q$ where $q \neq q_{reject}$

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

1. The first domino kicks of the computation history

$$\left[\frac{t_1}{b_1}\right] = \left[\frac{\#}{\#q_0 w_1 w_2 \cdots w_n\#}\right],$$

2. Handle right moving transitions. For every $a, b \in \Gamma$ and every $q, r \in Q$ where $q \neq q_{reject}$

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

3. Handle left moving transitions. For every $a, b, c \in \Gamma$ and every $q, r \in Q$ where $q \neq q_{reject}$

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

1. The first domino kicks of the computation history

$$\left[\frac{t_1}{b_1}\right] = \left[\frac{\#}{\# q_0 w_1 w_2 \cdots w_n \#}\right],$$

2. Handle right moving transitions. For every $a, b \in \Gamma$ and every $q, r \in Q$ where $q \neq q_{reject}$

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

3. Handle left moving transitions. For every $a, b, c \in \Gamma$ and every $q, r \in Q$ where $q \neq q_{reject}$

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

4. For every $a \in \Gamma$ put $\left[\frac{\mathbf{a}}{\mathbf{a}}\right]$ into P'

PCP – Adding the right kind of dominos

1. The first domino kicks of the computation history
$$\left[\frac{t_1}{b_1}\right] = \left[\frac{\#}{\#q_0 w_1 w_2 \cdots w_n\#}\right],$$
2. Handle right moving transitions. For every $a, b \in \Gamma$ and every $q, r \in Q$ where $q \neq q_{reject}$
$$\text{if } \delta(q, a) = (r, b, R), \text{ put } \left[\frac{qa}{br}\right] \text{into } P'$$
3. Handle left moving transitions. For every $a, b, c \in \Gamma$ and every $q, r \in Q$ where $q \neq q_{reject}$
$$\text{if } \delta(q, a) = (r, b, L), \text{ put } \left[\frac{cqa}{rcb}\right] \text{into } P'$$
4. For every $a \in \Gamma$ put $\left[\frac{\mathbf{a}}{\mathbf{a}}\right]$ into P'
5. Put $\left[\frac{\#}{\#}\right]$ and $\left[\frac{\#}{\sqcup\#}\right]$ into P'.

PCP - How the dominos work

- Let us assume $\Gamma = \{0, 1, 2, \sqcup\}$, $w = 0100$ and that $\delta(q_0, 0) = (q_7, 2, R)$

- Let us assume $\Gamma = \{0, 1, 2, \sqcup\}$, $w = 0100$ and that $\delta(q_0, 0) = (q_7, 2, R)$
- Part 1 places the first domino and the match begins

$$\left[\frac{\#}{\#\ q_0\ 0\ 1\ 0\ 0\ \#}\right]$$

- Let us assume $\Gamma = \{0, 1, 2, \sqcup\}$, $w = 0100$ and that $\delta(q_0, 0) = (q_7, 2, R)$
- Part 1 places the first domino and the match begins

$$\begin{array}{ccccccccc} \# & q_0 & 0 & & & & & & \\ \# & q_0 & 0 & 1 & 0 & 0 & \# & 2 & q_7 \end{array}$$

- Part 2 places the domino $\left[\frac{q_0 0}{2 q_7}\right]$

PCP - How the dominos work

- Let us assume $\Gamma = \{0, 1, 2, \sqcup\}$, $w = 0100$ and that $\delta(q_0, 0) = (q_7, 2, R)$
- Part 1 places the first domino and the match begins

$$\begin{array}{cccccccccccc} \# & q_0 & 0 & 1 & 0 & 0 & & & & & & \\ \# & q_0 & 0 & 1 & 0 & 0 & \# & 2 & q_7 & 1 & 0 & 0 \end{array}$$

- Part 2 places the domino $\left[\frac{q_0 0}{2 q_7}\right]$
- Part 4 places the dominos $\left[\frac{0}{0}\right]$ $\left[\frac{1}{1}\right]$ $\left[\frac{2}{2}\right]$ and $\left[\frac{\sqcup}{\sqcup}\right]$ into P' so we can extend the match.

PCP - How the dominos work

- Let us assume $\Gamma = \{0, 1, 2, \sqcup\}$, $w = 0100$ and that $\delta(q_0, 0) = (q_7, 2, R)$
- Part 1 places the first domino and the match begins

$$\begin{array}{ccccccccccccc} \# & q_0 & 0 & 1 & 0 & 0 & \# & & & & & & \\ \# & q_0 & 0 & 1 & 0 & 0 & \# & 2 & q_7 & 1 & 0 & 0 & \# \end{array}$$

- Part 2 places the domino $\left[\frac{q_0 0}{2 q_7}\right]$
- Part 4 places the dominos $\left[\frac{0}{0}\right]$ $\left[\frac{1}{1}\right]$ $\left[\frac{2}{2}\right]$ and $\left[\frac{\sqcup}{\sqcup}\right]$ into P' so we can extend the match.
- Part 5 puts in the domino $\left[\frac{\#}{\#}\right]$

- Let us assume $\Gamma = \{0, 1, 2, \sqcup\}$, $w = 0100$ and that $\delta(q_0, 0) = (q_7, 2, R)$
- Part 1 places the first domino and the match begins

$$\begin{array}{ccccccccccccc} \# & q_0 & 0 & 1 & 0 & 0 & \# & & & & & & \\ \# & q_0 & 0 & 1 & 0 & 0 & \# & 2 & q_7 & 1 & 0 & 0 & \# \end{array}$$

- Part 2 places the domino $\left[\frac{q_0 0}{2 q_7}\right]$
- Part 4 places the dominos $\left[\frac{0}{0}\right]$ $\left[\frac{1}{1}\right]$ $\left[\frac{2}{2}\right]$ and $\left[\frac{\sqcup}{\sqcup}\right]$ into P' so we can extend the match.
- Part 5 puts in the domino $\left[\frac{\#}{\#}\right]$
- What exactly is going on ?

PCP - How the dominos work

- Let us assume $\Gamma = \{0, 1, 2, \sqcup\}$, $w = 0100$ and that $\delta(q_0, 0) = (q_7, 2, R)$
- Part 1 places the first domino and the match begins

$$\begin{array}{ccccccccccccc} \# & q_0 & 0 & 1 & 0 & 0 & \# & & & & & & \\ \# & q_0 & 0 & 1 & 0 & 0 & \# & 2 & q_7 & 1 & 0 & 0 & \# \end{array}$$

- Part 2 places the domino $\left[\frac{q_0 0}{2 q_7}\right]$
- Part 4 places the dominos $\left[\frac{0}{0}\right]$ $\left[\frac{1}{1}\right]$ $\left[\frac{2}{2}\right]$ and $\left[\frac{\sqcup}{\sqcup}\right]$ into P' so we can extend the match.
- Part 5 puts in the domino $\left[\frac{\#}{\#}\right]$
- What exactly is going on ?
- We force the bottom string to create a copy on the top which is forced to generate the next configuration on the bottom – We are simulating M on w!

PCP - How the dominos work

- Let us assume $\Gamma = \{0, 1, 2, \sqcup\}$, $w = 0100$ and that $\delta(q_0, 0) = (q_7, 2, R)$
- Part 1 places the first domino and the match begins

$$\begin{array}{ccccccccccccc} \# & q_0 & 0 & 1 & 0 & 0 & \# & & & & & & \\ \# & q_0 & 0 & 1 & 0 & 0 & \# & 2 & q_7 & 1 & 0 & 0 & \# \end{array}$$

- Part 2 places the domino $\left[\frac{q_0 0}{2 q_7}\right]$
- Part 4 places the dominos $\left[\frac{0}{0}\right]$ $\left[\frac{1}{1}\right]$ $\left[\frac{2}{2}\right]$ and $\left[\frac{\sqcup}{\sqcup}\right]$ into P' so we can extend the match.
- Part 5 puts in the domino $\left[\frac{\#}{\#}\right]$
- What exactly is going on ?
- We force the bottom string to create a copy on the top which is forced to generate the next configuration on the bottom – We are simulating M on w!
- The process continues until M reaches a halting state and we then pad the upper string.

PCP – More domino types

6. For every $a \in \Gamma$,

$$\text{put } \left[\frac{\mathbf{a}\mathbf{q}_{\mathbf{accept}}}{\mathbf{q}_{\mathbf{accept}}}\right] \text{ and } \left[\frac{\mathbf{q}_{\mathbf{accept}}\mathbf{a}}{\mathbf{q}_{\mathbf{accept}}}\right] \text{into } P'$$

These dominos “clean-up” by adding any symbols to the top string while adding just the state symbol to the lower string.

6. For every $a \in \Gamma$,

$$\text{put } \left[\frac{\mathbf{a}\mathbf{q}_{\mathbf{accept}}}{\mathbf{q}_{\mathbf{accept}}}\right] \text{ and } \left[\frac{\mathbf{q}_{\mathbf{accept}}\mathbf{a}}{\mathbf{q}_{\mathbf{accept}}}\right] \text{into } P'$$

These dominos “clean-up” by adding any symbols to the top string while adding just the state symbol to the lower string.

Just before these apply the upper and lower strings are like

$$\begin{array}{l} \ldots\ \mathbf{\#} \\ \ldots\ \mathbf{\#\,2\,1\,q_{accept}\,0\,2\,\#} \end{array}$$

6. For every $a \in \Gamma$,

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

These dominos "clean-up" by adding any symbols to the top string while adding just the state symbol to the lower string.

Just before these apply the upper and lower strings are like

$$\begin{array}{l} \ldots\ \# \\ \ldots\ \#\ 2\ 1\ q_{accept}\ 0\ 2\ \# \end{array}$$

After using these dominos, we end up with

$$\begin{array}{l} \ldots\ \# \\ \ldots\ \#\ q_{accept}\ \# \end{array}$$

6. For every $a \in \Gamma$,

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

These dominos "clean-up" by adding any symbols to the top string while adding just the state symbol to the lower string.

Just before these apply the upper and lower strings are like

$$\begin{array}{l} \ldots \# \\ \ldots\ \# 2\ 1\ q_{accept}\ 0\ 2\ \# \end{array}$$

After using these dominos, we end up with

$$\begin{array}{l} \ldots \# \\ \ldots \# q_{accept}\ \# \end{array}$$

7. Finally we add the domino

$$\left[\frac{q_{accept}\#\#}{\#}\right]$$

to complete the match.

PCP PROOF – SUMMARY OF PART 1

- This concluses the construction of P'.

PCP Proof – Summary of Part 1

- This concludes the construction of P'.
- Thus if M accepts w, the set of MPCP dominos constructed have a solution to the MPCP problem.

PCP Proof – Summary of Part 1

- This concludes the construction of P'.
- Thus if M accepts w, the set of MPCP dominos constructed have a solution to the MPCP problem.
- But not yet to the PCP problem.

PCP Proof – Part 2

- Suppose we have the MPCP instance

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

PCP Proof – Part 2

- Suppose we have the MPCP instance

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

- We let P be the collection

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

PCP Proof – Part 2

- Suppose we have the MPCP instance

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

- We let P be the collection

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

- The only domino that could possibly start a match is the first one!

PCP PROOF – PART 2

- Suppose we have the MPCP instance

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

- We let P be the collection

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

- The only domino that could possibly start a match is the first one!
- The last domino just adds the missing $\star$ at the end of the match.

PCP Proof – Part 2

- Suppose we have the MPCP instance

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

- We let P be the collection

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

- The only domino that could possibly start a match is the first one!
- The last domino just adds the missing $\star$ at the end of the match.

Conclusion

PCP is undecidable!

Summary of Reducibility

We know that language A is undecidable. By reducing A to B we want to show that the language B is also undecidable.

Summary of Reducibility

We know that language A is undecidable. By reducing A to B we want to show that the language B is also undecidable.

1. Assume that we have a decider M_B for B.

Summary of Reducibility

We know that language A is undecidable. By reducing A to B we want to show that the language B is also undecidable.

1. Assume that we have a decider M_B for B.
2. Using M_B we construct a decider M_A for the language A:

Summary of Reducibility

We know that language A is undecidable. By reducing A to B we want to show that the language B is also undecidable.

1. Assume that we have a decider M_B for B.
2. Using M_B we construct a decider M_A for the language A:

$M_A =$ "On input $\langle I_A \rangle$

We know that language A is undecidable. By reducing A to B we want to show that the language B is also undecidable.

1. Assume that we have a decider M_B for B.
2. Using M_B we construct a decider M_A for the language A:

$M_A =$ "On input $\langle I_A \rangle$

1. Algorithmically construct an input $\langle I_B \rangle$ for M_B, such that

Summary of Reducibility

We know that language A is undecidable. By reducing A to B we want to show that the language B is also undecidable.

1. Assume that we have a decider M_B for B.
2. Using M_B we construct a decider M_A for the language A:

$M_A =$ "On input $\langle I_A \rangle$

1. Algorithmically construct an input $\langle I_B \rangle$ for M_B, such that

We know that language A is undecidable. By reducing A to B we want to show that the language B is also undecidable.

1. Assume that we have a decider M_B for B.
2. Using M_B we construct a decider M_A for the language A:

$M_A =$ "On input $\langle I_A \rangle$

1. Algorithmically construct an input $\langle I_B \rangle$ for M_B, such that
   a) Either

Summary of Reducibility

We know that language A is undecidable. By reducing A to B we want to show that the language B is also undecidable.

1. Assume that we have a decider M_B for B.
2. Using M_B we construct a decider M_A for the language A:

$M_A =$ "On input $\langle I_A \rangle$

1. Algorithmically construct an input $\langle I_B \rangle$ for M_B, such that
   a) Either

   If $\langle I_A \rangle \in A$ then $\langle I_B \rangle \in B$
   If $\langle I_A \rangle \notin A$ then $\langle I_B \rangle \notin B$

We know that language A is undecidable. By reducing A to B we want to show that the language B is also undecidable.

1. Assume that we have a decider M_B for B.
2. Using M_B we construct a decider M_A for the language A:

$M_A =$ "On input $\langle I_A \rangle$

1. Algorithmically construct an input $\langle I_B \rangle$ for M_B, such that

   a) Either

   If $\langle I_A \rangle \in A$ then $\langle I_B \rangle \in B$
   If $\langle I_A \rangle \notin A$ then $\langle I_B \rangle \notin B$

   b) or

SUMMARY OF REDUCIBILITY

We know that language A is undecidable. By reducing A to B we want to show that the language B is also undecidable.

1. Assume that we have a decider M_B for B.
2. Using M_B we construct a decider M_A for the language A:

$M_A =$ "On input $\langle I_A \rangle$

1. Algorithmically construct an input $\langle I_B \rangle$ for M_B, such that

a) Either

If $\langle I_A \rangle \in A$ then $\langle I_B \rangle \in B$
If $\langle I_A \rangle \notin A$ then $\langle I_B \rangle \notin B$

b) or

If $\langle I_A \rangle \in A$ then $\langle I_B \rangle \notin B$
If $\langle I_A \rangle \notin A$ then $\langle I_B \rangle \in B$

Summary of Reducibility

We know that language A is undecidable. By reducing A to B we want to show that the language B is also undecidable.

1. Assume that we have a decider M_B for B.
2. Using M_B we construct a decider M_A for the language A:

$M_A =$ "On input $\langle I_A \rangle$

1. Algorithmically construct an input $\langle I_B \rangle$ for M_B, such that

   a) Either

   If $\langle I_A \rangle \in A$ then $\langle I_B \rangle \in B$
   If $\langle I_A \rangle \notin A$ then $\langle I_B \rangle \notin B$

   b) or

   If $\langle I_A \rangle \in A$ then $\langle I_B \rangle \notin B$
   If $\langle I_A \rangle \notin A$ then $\langle I_B \rangle \in B$

2. Run the decider M_B on $\langle I_B \rangle$ for M_B
   Case a): M_A accepts if M_B accepts, and rejects if M_B rejects
   Case b): M_A rejects if M_B accepts, and accepts if M_B reject.

SUMMARY OF REDUCIBILITY

We know that language A is undecidable. By reducing A to B we want to show that the language B is also undecidable.

1. Assume that we have a decider M_B for B.
2. Using M_B we construct a decider M_A for the language A:

$M_A =$ "On input $\langle I_A \rangle$

1. Algorithmically construct an input $\langle I_B \rangle$ for M_B, such that

   a) Either

   If $\langle I_A \rangle \in A$ then $\langle I_B \rangle \in B$
   If $\langle I_A \rangle \notin A$ then $\langle I_B \rangle \notin B$

   b) or

   If $\langle I_A \rangle \in A$ then $\langle I_B \rangle \notin B$
   If $\langle I_A \rangle \notin A$ then $\langle I_B \rangle \in B$

2. Run the decider M_B on $\langle I_B \rangle$ for M_B
   Case a): M_A accepts if M_B accepts, and rejects if M_B rejects
   Case b): M_A rejects if M_B accepts, and accepts if M_B reject.

3. We know M_A can not exist so M_B can not exist.

Summary of Reducibility

We know that language A is undecidable. By reducing A to B we want to show that the language B is also undecidable.

1. Assume that we have a decider M_B for B.
2. Using M_B we construct a decider M_A for the language A:

$M_A =$ "On input $\langle I_A \rangle$

1. Algorithmically construct an input $\langle I_B \rangle$ for M_B, such that

   a) Either

   If $\langle I_A \rangle \in A$ then $\langle I_B \rangle \in B$
   If $\langle I_A \rangle \notin A$ then $\langle I_B \rangle \notin B$

   b) or

   If $\langle I_A \rangle \in A$ then $\langle I_B \rangle \notin B$
   If $\langle I_A \rangle \notin A$ then $\langle I_B \rangle \in B$

2. Run the decider M_B on $\langle I_B \rangle$ for M_B
   Case a): M_A accepts if M_B accepts, and rejects if M_B rejects
   Case b): M_A rejects if M_B accepts, and accepts if M_B reject.

3. We know M_A can not exist so M_B can not exist.
4. B is undecidable.

Computable Functions

Idea

Turing Machines can also compute function $f : \Sigma^* \longrightarrow \Sigma^*$.

COMPUTABLE FUNCTIONS

IDEA

Turing Machines can also compute function $f : \Sigma^* \longrightarrow \Sigma^*$.

COMPUTABLE FUNCTION

A function $f : \Sigma^* \longrightarrow \Sigma^*$ is a computable function if and only if there exists a TM M_f, which on any given input $w \in \Sigma^*$

IDEA

Turing Machines can also compute function $f : \Sigma^* \longrightarrow \Sigma^*$.

COMPUTABLE FUNCTION

A function $f : \Sigma^* \longrightarrow \Sigma^*$ is a computable function if and only if there exists a TM M_f, which on any given input $w \in \Sigma^*$

- always halts, and

Computable Functions

Idea

Turing Machines can also compute function $f : \Sigma^* \longrightarrow \Sigma^*$.

Computable Function

A function $f : \Sigma^* \longrightarrow \Sigma^*$ is a computable function if and only if there exists a TM M_f, which on any given input $w \in \Sigma^*$

- always halts, and
- leaves just $f(w)$ on its tape.

Computable Functions

Idea

Turing Machines can also compute function $f : \Sigma^* \longrightarrow \Sigma^*$.

Computable Function

A function $f : \Sigma^* \longrightarrow \Sigma^*$ is a computable function if and only if there exists a TM M_f, which on any given input $w \in \Sigma^*$

- always halts, and
- leaves just $f(w)$ on its tape.

Examples:

- Let $f(w) \stackrel{def}{=} ww$ be a function. Then f is computable.

Computable Functions

Idea

Turing Machines can also compute function $f : \Sigma^* \longrightarrow \Sigma^*$.

Computable Function

A function $f : \Sigma^* \longrightarrow \Sigma^*$ is a computable function if and only if there exists a TM M_f, which on any given input $w \in \Sigma^*$

- always halts, and
- leaves just $f(w)$ on its tape.

Examples:

- Let $f(w) \stackrel{def}{=} ww$ be a function. Then f is computable.
- Let $f(\langle n_1, n_2 \rangle) \stackrel{def}{=} \langle n \rangle$ where n_1 and n_2 are integers and $n = n_1 * n_2$. Then f is computable.

Mapping Reducibility

Definition

Let $A, B \subseteq \Sigma^*$. We say that language A is mapping reducible to language B, written $A <_m B$, if and only if

Mapping Reducibility

Definition

Let $A, B \subseteq \Sigma^*$. We say that language A is mapping reducible to language B, written $A <_m B$, if and only if

1. There is a computable function $f : \Sigma^* \longrightarrow \Sigma^*$ such that

Mapping Reducibility

Definition

Let $A, B \subseteq \Sigma^*$. We say that language A is mapping reducible to language B, written $A <_m B$, if and only if

1. There is a computable function $f : \Sigma^* \longrightarrow \Sigma^*$ such that
2. For every $w \in \Sigma^*$, $w \in A$ if and only if $f(w) \in B$.

Mapping Reducibility

Definition

Let $A, B \subseteq \Sigma^*$. We say that language A is mapping reducible to language B, written $A <_m B$, if and only if

1. There is a computable function $f : \Sigma^* \longrightarrow \Sigma^*$ such that
2. For every $w \in \Sigma^*$, $w \in A$ if and only if $f(w) \in B$.

The function f is called a reduction of A to B.

Theorem 5.22

If $A <_m B$ and B is decidable, then A is decidable.

Mapping Reducibility

Definition

Let $A, B \subseteq \Sigma^*$. We say that language A is mapping reducible to language B, written $A <_m B$, if and only if

1. There is a computable function $f : \Sigma^* \longrightarrow \Sigma^*$ such that
2. For every $w \in \Sigma^*$, $w \in A$ if and only if $f(w) \in B$.

The function f is called a reduction of A to B.

Theorem 5.22

If $A <_m B$ and B is decidable, then A is decidable.

Proof

Let M be a decider for B and f be a mapping from A to B. Then N decides A.
$N =$ "On input w

Definition

Let $A, B \subseteq \Sigma^*$. We say that language A is mapping reducible to language B, written $A <_m B$, if and only if

1. There is a computable function $f : \Sigma^* \longrightarrow \Sigma^*$ such that
2. For every $w \in \Sigma^*$, $w \in A$ if and only if $f(w) \in B$.

The function f is called a reduction of A to B.

Theorem 5.22

If $A <_m B$ and B is decidable, then A is decidable.

Proof

Let M be a decider for B and f be a mapping from A to B. Then N decides A.
$N =$ "On input w

1. Compute $f(w)$

Definition

Let $A, B \subseteq \Sigma^*$. We say that language A is mapping reducible to language B, written $A <_m B$, if and only if

1. There is a computable function $f : \Sigma^* \longrightarrow \Sigma^*$ such that
2. For every $w \in \Sigma^*$, $w \in A$ if and only if $f(w) \in B$.

The function f is called a reduction of A to B.

Theorem 5.22

If $A <_m B$ and B is decidable, then A is decidable.

Proof

Let M be a decider for B and f be a mapping from A to B. Then N decides A.
$N =$ "On input w

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

MAPPING REDUCIBILITY

DEFINITION

Let $A, B \subseteq \Sigma^*$. We say that language A is mapping reducible to language B, written $A <_m B$, if and only if

1. There is a computable function $f : \Sigma^* \longrightarrow \Sigma^*$ such that
2. For every $w \in \Sigma^*$, $w \in A$ if and only if $f(w) \in B$.

The function f is called a reduction of A to B.

THEOREM 5.22

If $A <_m B$ and B is decidable, then A is decidable.

PROOF

Let M be a decider for B and f be a mapping from A to B. Then N decides A.
$N =$ "On input w

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

Mapping Reducibility

Definition

Let $A, B \subseteq \Sigma^*$. We say that language A is mapping reducible to language B, written $A <_m B$, if and only if

1. There is a computable function $f : \Sigma^* \longrightarrow \Sigma^*$ such that
2. For every $w \in \Sigma^*$, $w \in A$ if and only if $f(w) \in B$.

The function f is called a reduction of A to B.

Theorem 5.22

If $A <_m B$ and B is decidable, then A is decidable.

Proof

Let M be a decider for B and f be a mapping from A to B. Then N decides A.
$N =$ "On input w

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

If $A <_m B$ and A is undecidable, then B is undecidable.

Mapping Reducibility

Theorem

$A_{TM} <_m HALT_{TM}$

Mapping Reducibility

Theorem

$A_{TM} <_m HALT_{TM}$

Proof.

Construct a computable function f which maps $\langle M, w \rangle$ to $\langle M', w' \rangle$ such that

$$\langle M, w \rangle \in A_{TM} \text{ if and only if } \langle M', w' \rangle \in HALT_{TM}$$

$M_f =$ “On input $\langle M, w \rangle$

□

Theorem

$A_{TM} <_m HALT_{TM}$

Proof.

Construct a computable function f which maps $\langle M, w\rangle$ to $\langle M', w'\rangle$ such that

$$\langle M, w\rangle \in A_{TM} \text{ if and only if } \langle M', w'\rangle \in HALT_{TM}$$

M_f = "On input $\langle M, w\rangle$

1. Construct the following machine M':
   M' = "On input x

□

Mapping Reducibility

Theorem

$A_{TM} <_m HALT_{TM}$

Proof.

Construct a computable function f which maps $\langle M, w\rangle$ to $\langle M', w'\rangle$ such that

$$\langle M, w\rangle \in A_{TM} \text{ if and only if } \langle M', w'\rangle \in HALT_{TM}$$

$M_f =$ "On input $\langle M, w\rangle$

1. Construct the following machine M':
   $M' =$ "On input x
   1. Run M on x.

$\square$

Mapping Reducibility

Theorem

$A_{TM} <_m HALT_{TM}$

Proof.

Construct a computable function f which maps $\langle M, w \rangle$ to $\langle M', w' \rangle$ such that

$$\langle M, w \rangle \in A_{TM} \text{ if and only if } \langle M', w' \rangle \in HALT_{TM}$$

$M_f =$ "On input $\langle M, w \rangle$

1. Construct the following machine M':
   $M' =$ "On input x
   1. Run M on x.
   2. If M accepts *accept*

□

Mapping Reducibility

Theorem

$A_{TM} <_m HALT_{TM}$

Proof.

Construct a computable function f which maps $\langle M, w \rangle$ to $\langle M', w' \rangle$ such that

$$\langle M, w \rangle \in A_{TM} \text{ if and only if } \langle M', w' \rangle \in HALT_{TM}$$

$M_f =$ "On input $\langle M, w \rangle$

1. Construct the following machine M':
   $M' =$ "On input x
   1. Run M on x.
   2. If M accepts *accept*
   3. If M rejects *enter a loop*."

□

THEOREM

$A_{TM} <_m HALT_{TM}$

PROOF.

Construct a computable function f which maps $\langle M, w\rangle$ to $\langle M', w'\rangle$ such that

$$\langle M, w\rangle \in A_{TM} \text{ if and only if } \langle M', w'\rangle \in HALT_{TM}$$

$M_f =$ "On input $\langle M, w\rangle$

1. Construct the following machine M':
   $M' =$ "On input x
   1. Run M on x.
   2. If M accepts *accept*
   3. If M rejects *enter a loop*."
2. Output $\langle M', w\rangle$."

□

- Earlier we showed

More examples of Mapping Reducibility

- Earlier we showed
  - $A_{TM} <_m MPCP$

More examples of Mapping Reducibility

- Earlier we showed
  - $A_{TM} <_m MPCP$
  - $MPCP <_m PCP$

More examples of Mapping Reducibility

- Earlier we showed
  - $A_{TM} <_m MPCP$
  - $MPCP <_m PCP$
- In Theorem 5.4 we showed $E_{TM} <_m EQ_{TM}$. The reduction f maps from $\langle M \rangle$ to the output $\langle M, M_1 \rangle$ where M_1 is the machine that rejects all inputs.

More examples of Mapping Reducibility

- Earlier we showed
  - $A_{TM} <_m MPCP$
  - $MPCP <_m PCP$
- In Theorem 5.4 we showed $E_{TM} <_m EQ_{TM}$. The reduction f maps from $\langle M \rangle$ to the output $\langle M, M_1 \rangle$ where M_1 is the machine that rejects all inputs.

Theorem 5.24

If $A <_m B$ and B is Turing-recognizable, then A is Turing-recognizable.

- Earlier we showed
  - $A_{TM} <_m MPCP$
  - $MPCP <_m PCP$
- In Theorem 5.4 we showed $E_{TM} <_m EQ_{TM}$. The reduction f maps from $\langle M \rangle$ to the output $\langle M, M_1 \rangle$ where M_1 is the machine that rejects all inputs.

THEOREM 5.24

If $A <_m B$ and B is Turing-recognizable, then A is Turing-recognizable.

PROOF

Essentially the same as the previous proof.

Summary of Mapping Reducibility Results

Summary of Theorems

Assume that $A <_m B$. Then

SUMMARY OF THEOREMS

Assume that $A <_m B$. Then

1. If B is decidable then A is decidable.

Summary of Mapping Reducibility Results

Summary of Theorems

Assume that $A <_m B$. Then

1. If B is decidable then A is decidable.
2. If A is undecidable then B is undecidable.

Summary of Mapping Reducibility Results

Summary of Theorems

Assume that $A <_m B$. Then

1. If B is decidable then A is decidable.
2. If A is undecidable then B is undecidable.
3. If B is Turing-recognizable then A is Turing-recognizable.

Summary of Mapping Reducibility Results

Summary of Theorems

Assume that $A <_m B$. Then

1. If B is decidable then A is decidable.
2. If A is undecidable then B is undecidable.
3. If B is Turing-recognizable then A is Turing-recognizable.
4. If A is not Turing-recognizable then B is not Turing-recognizable.

Summary of Mapping Reducibility Results

Summary of Theorems

Assume that $A <_m B$. Then

1. If B is decidable then A is decidable.
2. If A is undecidable then B is undecidable.
3. If B is Turing-recognizable then A is Turing-recognizable.
4. If A is not Turing-recognizable then B is not Turing-recognizable.
5. $\overline{A} <_m \overline{B}$

Summary of Mapping Reducibility Results

Summary of Theorems

Assume that $A <_m B$. Then

1. If B is decidable then A is decidable.
2. If A is undecidable then B is undecidable.
3. If B is Turing-recognizable then A is Turing-recognizable.
4. If A is not Turing-recognizable then B is not Turing-recognizable.
5. $\overline{A} <_m \overline{B}$

Useful observation:

- Suppose you can show $A_{TM} <_m \overline{B}$

Summary of Theorems

Assume that $A <_m B$. Then

1. If B is decidable then A is decidable.
2. If A is undecidable then B is undecidable.
3. If B is Turing-recognizable then A is Turing-recognizable.
4. If A is not Turing-recognizable then B is not Turing-recognizable.
5. $\overline{A} <_m \overline{B}$

Useful observation:

- Suppose you can show $A_{TM} <_m \overline{B}$
- This means $\overline{A_{TM}} <_m B$

Summary of Theorems

Assume that $A <_m B$. Then

1. If B is decidable then A is decidable.
2. If A is undecidable then B is undecidable.
3. If B is Turing-recognizable then A is Turing-recognizable.
4. If A is not Turing-recognizable then B is not Turing-recognizable.
5. $\overline{A} <_m \overline{B}$

Useful observation:

- Suppose you can show $A_{TM} <_m \overline{B}$
- This means $\overline{A_{TM}} <_m B$
- Since $\overline{A_{TM}}$ is Turing-unrecognizable then B is Turing-unrecognizable.

Example of Use

Theorem 5.30

$EQ_{TM} = \{\langle M_1, M_2\rangle \mid M_1$ and M_2 are TMs and $L(M_1) = L(M_2)\}$ is neither Turing recognizable nor co-Turing-recognizable.

Example of Use

Theorem 5.30

$EQ_{TM} = \{\langle M_1, M_2\rangle \mid M_1$ and M_2 are TMs and $L(M_1) = L(M_2)\}$ is neither Turing recognizable nor co-Turing-recognizable.

Proof Idea

We show

Example of Use

Theorem 5.30

$EQ_{TM} = \{\langle M_1, M_2\rangle \mid M_1$ and M_2 are TMs and $L(M_1) = L(M_2)\}$ is neither Turing recognizable nor co-Turing-recognizable.

Proof Idea

We show

- $\overline{A_{TM}} <_m EQ_{TM}$

Example of Use

Theorem 5.30

$EQ_{TM} = \{\langle M_1, M_2\rangle \mid M_1$ and M_2 are TMs and $L(M_1) = L(M_2)\}$ is neither Turing recognizable nor co-Turing-recognizable.

Proof Idea

We show

- $\overline{A_{TM}} <_m EQ_{TM}$
- $\overline{A_{TM}} <_m \overline{EQ_{TM}}$

EXAMPLE OF USE

THEOREM 5.30

$EQ_{TM} = \{\langle M_1, M_2\rangle \mid M_1$ and M_2 are TMs and $L(M_1) = L(M_2)\}$ is neither Turing recognizable nor co-Turing-recognizable.

PROOF IDEA

We show

- $\overline{A_{TM}} <_m EQ_{TM}$
- $\overline{A_{TM}} <_m \overline{EQ_{TM}}$
- These then imply the theorem.

Proof for $\overline{A_{TM}} <_m EQ_{TM}$

We show $A_{TM} <_m \overline{EQ_{TM}}$ (and hence $\overline{A_{TM}} <_m EQ_{TM}$) with the following f:

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

Proof for $\overline{A_{TM}} <_m EQ_{TM}$

We show $A_{TM} <_m \overline{EQ_{TM}}$ (and hence $\overline{A_{TM}} <_m EQ_{TM}$) with the following f:

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

1. Construct the following two machines M_1 and M_2

PROOF FOR $\overline{A_{TM}} <_m EQ_{TM}$

We show $A_{TM} <_m \overline{EQ_{TM}}$ (and hence $\overline{A_{TM}} <_m EQ_{TM}$) with the following f:

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

1. Construct the following two machines M_1 and M_2
   $M_1 =$ "On any input:
   1. Reject"

PROOF FOR $\overline{A_{TM}} <_m EQ_{TM}$

We show $A_{TM} <_m \overline{EQ_{TM}}$ (and hence $\overline{A_{TM}} <_m EQ_{TM}$) with the following f:

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

1. Construct the following two machines M_1 and M_2
   $M_1 =$ "On any input:
   1. Reject"

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

Proof for $\overline{A_{TM}} <_m EQ_{TM}$

We show $A_{TM} <_m \overline{EQ_{TM}}$ (and hence $\overline{A_{TM}} <_m EQ_{TM}$) with the following f:

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

1. Construct the following two machines M_1 and M_2
   $M_1 =$ "On any input:
   1. Reject"

   $M_2 =$ "On any input:
   1. Run M on w. If it accepts, *accept.*"
2. Output $\langle M_1, M_2\rangle$."

Example of Use

Proof for $\overline{A_{TM}} <_m EQ_{TM}$

We show $A_{TM} <_m \overline{EQ_{TM}}$ (and hence $\overline{A_{TM}} <_m EQ_{TM}$) with the following f:

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

1. Construct the following two machines M_1 and M_2
   $M_1 =$ "On any input:
   1. Reject"

   $M_2 =$ "On any input:
   1. Run M on w. If it accepts, *accept.*"
2. Output $\langle M_1, M_2\rangle$."

- M_1 accepts nothing.

PROOF FOR $\overline{A_{TM}} <_m EQ_{TM}$

We show $A_{TM} <_m \overline{EQ_{TM}}$ (and hence $\overline{A_{TM}} <_m EQ_{TM}$) with the following f:

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

1. Construct the following two machines M_1 and M_2
   $M_1 =$ "On any input:
   1. Reject"

   $M_2 =$ "On any input:
   1. Run M on w. If it accepts, *accept.*"
2. Output $\langle M_1, M_2 \rangle$."

- M_1 accepts nothing.
  - If M accepts w then M_2 accepts everything. So M_1 and M_2 are not equivalent.

PROOF FOR $\overline{A_{TM}} <_m EQ_{TM}$

We show $A_{TM} <_m \overline{EQ_{TM}}$ (and hence $\overline{A_{TM}} <_m EQ_{TM}$) with the following f:

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

1. Construct the following two machines M_1 and M_2
   $M_1 =$ "On any input:
   1. Reject"

   $M_2 =$ "On any input:
   1. Run M on w. If it accepts, *accept.*"
2. Output $\langle M_1, M_2 \rangle$."

- M_1 accepts nothing.
  - If M accepts w then M_2 accepts everything. So M_1 and M_2 are not equivalent.
  - If M does not accept w then M_2 accepts nothing. So M_1 and M_2 are equivalent.

EXAMPLE OF USE

PROOF FOR $\overline{A_{TM}} <_m EQ_{TM}$

We show $A_{TM} <_m \overline{EQ_{TM}}$ (and hence $\overline{A_{TM}} <_m EQ_{TM}$) with the following f:

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

1. Construct the following two machines M_1 and M_2
   $M_1 =$ "On any input:
   1. Reject"

   $M_2 =$ "On any input:
   1. Run M on w. If it accepts, *accept.*"
2. Output $\langle M_1, M_2 \rangle$."

- M_1 accepts nothing.
  - If M accepts w then M_2 accepts everything. So M_1 and M_2 are not equivalent.
  - If M does not accept w then M_2 accepts nothing. So M_1 and M_2 are equivalent.
- So $A_{TM} <_m \overline{EQ_{TM}}$ (and hence $\overline{A_{TM}} <_m EQ_{TM}$)

Proof for $\overline{A_{TM}} <_m \overline{EQ_{TM}}$

We show $A_{TM} <_m EQ_{TM}$ (and hence $\overline{A_{TM}} <_m \overline{EQ_{TM}}$) with the following g:

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

Proof for $\overline{A_{TM}} <_m \overline{EQ_{TM}}$

We show $A_{TM} <_m EQ_{TM}$ (and hence $\overline{A_{TM}} <_m \overline{EQ_{TM}}$) with the following g:

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

1. Construct the following two machines M_1 and M_2

PROOF FOR $\overline{A_{TM}} <_m \overline{EQ_{TM}}$

We show $A_{TM} <_m EQ_{TM}$ (and hence $\overline{A_{TM}} <_m \overline{EQ_{TM}}$) with the following g:

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

1. Construct the following two machines M_1 and M_2
   $M_1 =$ "On any input:
   1. Accept"

Example of Use

Proof for $\overline{A_{TM}} <_m \overline{EQ_{TM}}$

We show $A_{TM} <_m EQ_{TM}$ (and hence $\overline{A_{TM}} <_m \overline{EQ_{TM}}$) with the following g:

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

1. Construct the following two machines M_1 and M_2
   $M_1 =$ "On any input:
   1. Accept"

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

Example of Use

Proof for $\overline{A_{TM}} <_m \overline{EQ_{TM}}$

We show $A_{TM} <_m EQ_{TM}$ (and hence $\overline{A_{TM}} <_m \overline{EQ_{TM}}$) with the following g:

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

1. Construct the following two machines M_1 and M_2
   $M_1 =$ "On any input:
   1. Accept"

   $M_2 =$ "On any input:
   1. Run M on w. If it accepts, *accept.*"
2. Output $\langle M_1, M_2\rangle$."

Proof for $\overline{A_{TM}} <_m \overline{EQ_{TM}}$

We show $A_{TM} <_m EQ_{TM}$ (and hence $\overline{A_{TM}} <_m \overline{EQ_{TM}}$) with the following g:

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

1. Construct the following two machines M_1 and M_2
   $M_1 =$ "On any input:
   1. Accept"

   $M_2 =$ "On any input:
   1. Run M on w. If it accepts, *accept.*"
2. Output $\langle M_1, M_2\rangle$."

- M_1 accepts everything.

PROOF FOR $\overline{A_{TM}} <_m \overline{EQ_{TM}}$

We show $A_{TM} <_m EQ_{TM}$ (and hence $\overline{A_{TM}} <_m \overline{EQ_{TM}}$) with the following g:

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

1. Construct the following two machines M_1 and M_2
   $M_1 =$ "On any input:
   1. Accept"

   $M_2 =$ "On any input:
   1. Run M on w. If it accepts, *accept.*"
2. Output $\langle M_1, M_2\rangle$."

- M_1 accepts everything.
  - If M accepts w then M_2 accepts everything. So M_1 and M_2 are equivalent.

PROOF FOR $\overline{A_{TM}} <_m \overline{EQ_{TM}}$

We show $A_{TM} <_m EQ_{TM}$ (and hence $\overline{A_{TM}} <_m \overline{EQ_{TM}}$) with the following g:

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

1. Construct the following two machines M_1 and M_2
   $M_1 =$ "On any input:
   1. Accept"

   $M_2 =$ "On any input:
   1. Run M on w. If it accepts, *accept.*"
2. Output $\langle M_1, M_2\rangle$."

- M_1 accepts everything.
  - If M accepts w then M_2 accepts everything. So M_1 and M_2 are equivalent.
  - If M does not accept w then M_2 accepts nothing. So M_1 and M_2 are not equivalent.

Example of Use

Proof for $\overline{A_{TM}} <_m \overline{EQ_{TM}}$

We show $A_{TM} <_m EQ_{TM}$ (and hence $\overline{A_{TM}} <_m \overline{EQ_{TM}}$) with the following g:

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

1. Construct the following two machines M_1 and M_2
   $M_1 =$ "On any input:
   1. Accept"

   $M_2 =$ "On any input:
   1. Run M on w. If it accepts, *accept.*"
2. Output $\langle M_1, M_2\rangle$."

- M_1 accepts everything.
  - If M accepts w then M_2 accepts everything. So M_1 and M_2 are equivalent.
  - If M does not accept w then M_2 accepts nothing. So M_1 and M_2 are not equivalent.
- So $A_{TM} <_m EQ_{TM}$ (and hence $\overline{A_{TM}} <_m \overline{EQ_{TM}}$)