FORMAL LANGUAGES, AUTOMATA AND COMPUTATION

POST CORRESPONDENCE PROBLEM
REVIEW OF DECIDABILITY AND REDUCTIONS

A DFA  A CFG

A TM

DEC  RL  CFL  T-REC  T-UNREC
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 $A$ cannot 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.
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.
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 $B$. 
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 $A$
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**

\[ E_{TM} = \{ \langle M \rangle \mid M \text{ is a TM and } L(M) = \Phi \} \text{ is undecidable.} \]
Proving Undecidability via Reductions

**Theorem 5.2**

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

- Suppose \( R \) decides \( E_{TM} \). We try to construct \( S \) to decide \( A_{TM} \) using \( R \).
**Theorem 5.2**

\[ E_{TM} = \{ \langle M \rangle \mid M \text{ is a TM and } L(M) = \emptyset \} \] 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.
Theorem 5.2

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

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 does not tell us if \( M \) accepts \( w \).
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 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.
**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 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 \( L(M_1) = \Phi \).
PROVING UNDECIDABILITY VIA REDUCTIONS

**Theorem 5.2**

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

---

For any $w$ define $M_1$ as:

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

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!
Theorem 5.2

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

Proof

For any \( w \) define \( M_1 \) as

\[ M_1 = \text{“On input } x:\]
THEOREM 5.2

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

PROOF

- For any \( w \) define \( M_1 \) as
  - \( M_1 = \text{"On input } x:\) 
  - 1. If \( x \neq w \), reject.
**Theorem 5.2**

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

**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.”
Theorem 5.2

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

Proof

For any $w$ define $M_1$ as

$M_1 = \text{"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!
Assume \( R \) decides \( E_{TM} \) defines below uses \( R \) to decide on \( A_{TM} \)

\[
S = \begin{cases} 
\text{On input } \langle M, w \rangle \\
1 \quad \text{Use } \langle M, w \rangle \text{ to construct } M_1 \text{ above.} \\
2 \quad \text{Run } R \text{ on input } \langle M_1 \rangle \\
3 \quad \text{If } R \text{ accepts, reject, if } R \text{ rejects, accept.} 
\end{cases}
\]

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.
PROOF CONTINUED

- Assume $R$ decides $E_{TM}$
Proof continued

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

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

$S$ defines below uses $R$ to decide on $A_{TM}$

$S = \text{“On input } \langle M, w \rangle \text{”}$

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.
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,
Assume $R$ decides $E_{TM}$

$S$ defines below uses $R$ to decide on $A_{TM}$

$S = \text{“On input } \langle M, w \rangle \text{”}$

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 $w$, if $R$ decides $E_{TM}$ then $S$ decides $A_{TM}$ – Contradiction.
Assume $R$ decides $E_{TM}$

$S$ defines below uses $R$ to decide on $A_{TM}$

$S = \text{“On input } \langle M, w \rangle \text{”}$

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.
Assume $R$ decides $E_{TM}$

$S$ defines below uses $R$ to decide on $A_{TM}$

$S = \text{"On input } \langle M, w \rangle$

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
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
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.

If \( M \) does not halt on \( w \), there is no computation history.
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.
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.
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)
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 $qgn^2$ 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 $qgn^n$ distinct configurations for a tape of length $n$.

**Proof.**

- The machine can be in one of $q$ states.
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.
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**

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

$\#C_1\#C_2\#C_3\#\cdots\#C_l\#$

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 $C_1 = q_0w_1w_2\cdots w_n$. Check if $C_l = \cdots q_{\text{accept}}\cdots$. 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 $L(B) \neq \emptyset$ then $M$ accepts $w$. (Lecture 17)
Now for a really wild and crazy idea!
Consider an accepting computation history of a TM $M$, $C_1, C_2, \ldots, C_l$
Now for a really wild and crazy idea!
Consider an accepting computation history of a TM $M$, $C_1$, $C_2$, ... , $C_l$
Note that each $C_i$ is a string.
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

```
#  C_1  #  C_2  #  C_3  #  \cdots  #  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 $C_l = \cdots q_{\text{accept}} \cdots$
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 $L(B) \neq \emptyset$ then $M$ accepts $w$.
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

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

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

$$\# C_1 \# C_2 \# C_3 \# \cdots \# C_l \#$$

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 $w$. 
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

\[
\# \ C_1 \ # \ C_2 \ # \ C_3 \ # \ \cdots \ # \ C_l \ #
\]

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 $C_1 = q_0 w_1 w_2 \cdots w_n$
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

$$\# C_1 \# C_2 \# C_3 \# \cdots \# C_l \#$$

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 $C_l = \cdots q_{accept} \cdots$
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

$\# C_1 \# C_2 \# C_3 \# \cdots \# C_l \#$

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

$$\# \\ C_1 \# \\ C_2 \# \\ C_3 \# \cdots \# \\ C_l \#$$

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 $C_l = \cdots q_{\text{accept}} \cdots$
- 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

$$
\# \underline{C_1} \# \underline{C_2} \# \underline{C_3} \# \cdots \# \underline{C_l} \#
$$

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 $C_l = \cdots q_{\text{accept}} \cdots$
- 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.
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.
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., $\left[ \begin{array}{c} ca \\ a \end{array} \right]$.

Suppose we have dominos

\[ \left\{ \left[ \begin{array}{c} b \\ ca \end{array} \right], \left[ \begin{array}{c} a \\ ab \end{array} \right], \left[ \begin{array}{c} ca \\ a \end{array} \right], \left[ \begin{array}{c} abc \\ c \end{array} \right] \right\} \]
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

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

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

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

\[
\left[ \frac{a}{ab} \right] \left[ \frac{b}{ca} \right] \left[ \frac{ca}{ab} \right] \left[ \frac{a}{c} \right] = \frac{abcaaaabc}{abcaaaabc}
\]

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.
An instance of the PCP

A PCP instance over $\Sigma$ is a finite collection $P$ of dominos

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

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

AN INSTANCE OF THE PCP

A PCP instance over $\Sigma$ 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

\[ t_{i_1} t_{i_2} \cdots t_{i_\ell} = b_{i_1} b_{i_2} \cdots b_{i_\ell} \]
**Question:**

Does a given PCP instance $P$ have a match?

**Language formulation:**

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

**Theorem 5.15**

PCP is undecidable.

Proof: By reduction using computation histories. If PCP is decidable then so is $\text{A TM}$. That is, if PCP has a match, then $M$ accepts $w$. (Lecture 17)
**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 = $\{\langle P \rangle \mid P$ 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 ATM. That is, if PCP has a match, then $M$ accepts $w$. 

(Lecture 17)

Slides for 15-453

Spring 2011
**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 $w$. 
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:

\[ \text{MPCP} = \{ \langle P \rangle | P \text{ is a PCP instance and it has a match which starts with index } 1 \} \]

So the solution to MPCP starts with the domino $[t_1 b_1]$. We later remove this restriction in the second part of the proof.

We also assume that the decider for MPCP never moves its head to the left of the input $w$. 
The reduction works in two steps:

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

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

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

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.
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**

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

- 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 $w$. 
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.
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 $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$. 
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 $w$. 
The first domino kicks of the computation history

\[
\begin{bmatrix}
  t_1 \\
  b_1
\end{bmatrix} = \begin{bmatrix}
  \# \\
  \# q_0 w_1 w_2 \cdots w_n \#
\end{bmatrix},
\]
The first domino kicks of the computation history

\[ \begin{bmatrix} t_1 \\ b_1 \end{bmatrix} = \begin{bmatrix} \# \\ \# q_0 w_1 w_2 \cdots w_n \# \end{bmatrix}, \]

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

if \( \delta(q, a) = (r, b, R) \), put \[ \begin{bmatrix} qa \\ br \end{bmatrix} \] into \( P' \)
The first domino kicks of the computation history

\[
\begin{bmatrix}
    t_1 \\
    b_1
\end{bmatrix} = \begin{bmatrix}
    \# \\
    \# q_0 w_1 w_2 \cdots w_n \#
\end{bmatrix},
\]

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

if \( \delta(q, a) = (r, b, R) \), put \( \begin{bmatrix} qa \\ br \end{bmatrix} \) into \( P' \)

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

if \( \delta(q, a) = (r, b, L) \), put \( \begin{bmatrix} cqa \\ rcb \end{bmatrix} \) into \( P' \)
The first domino kicks of the computation history

\[
\begin{bmatrix}
  t_1 \\
  b_1
\end{bmatrix} = \begin{bmatrix}
  \# \\
  \# q_0 w_1 w_2 \cdots w_n \#
\end{bmatrix},
\]

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

if \( \delta(q, a) = (r, b, R) \), put \( \begin{bmatrix} qa \\ br \end{bmatrix} \) into \( P' \)

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

if \( \delta(q, a) = (r, b, L) \), put \( \begin{bmatrix} cqa \\ rcb \end{bmatrix} \) into \( P' \)

For every \( a \in \Gamma \) put \( \begin{bmatrix} a \\ a \end{bmatrix} \) into \( P' \)
The first domino kicks of the computation history

\[
\begin{bmatrix}
  t_1 \\
  b_1
\end{bmatrix}
= \begin{bmatrix}
  # \\
  \# q_0 w_1 w_2 \cdots w_n #
\end{bmatrix},
\]

1. Handle right moving transitions. For every \( a, b \in \Gamma \) and every \( q, r \in Q \) where \( q \neq q_{\text{reject}} \)

   \[
   \text{if} \ \delta(q, a) = (r, b, R), \ \text{put} \ \begin{bmatrix}
   qa \\
   br
\end{bmatrix} \text{into } P'
   \]

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

   \[
   \text{if} \ \delta(q, a) = (r, b, L), \ \text{put} \ \begin{bmatrix}
   cqa \\
   rcb
\end{bmatrix} \text{into } P'
   \]

3. For every \( a \in \Gamma \) put \[ \begin{bmatrix}
   a \\
   a
\end{bmatrix} \text{into } P'
   \]

4. Put \[ \begin{bmatrix}
   # \\
   #
\end{bmatrix} \text{ and } \begin{bmatrix}
   # \\
   \square #
\end{bmatrix} \text{ into } P'. \]

( Lecture 17)
Let us assume $\Gamma = \{0, 1, 2, \Box\}$, $w = 0100$ and that $\delta(q_0, 0) = (q_7, 2, R)$. 

Part 1 places the first domino and the match begins. 

Part 2 places the domino $[q_0 0 2q_7]$. 

Part 4 places the dominos $[0 0][1 1][2 2]$ and $[\Box \Box]$ into $P'$ so we can extend the match. 

Part 5 puts in the domino $[# #]$. 

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 - 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

```
#  q_0  0 1 0 0  #
```

Part 2 places the domino
```
#  q_0  2q_7  #
```

Part 4 places the dominos
```
[0 0][1 1][2 2][\sqcup \sqcup]
```

and we can extend the match.

Part 5 puts in the domino
```
#  #
```

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.

( Lecture 17)

Slides for 15-453

Spring 2011 16 / 28
Let us assume $\Gamma = \{0, 1, 2, \Box\}$, $w = 0100$ and that $\delta(q_0, 0) = (q_7, 2, R)$.

Part 1 places the first domino and the match begins

\[
\begin{array}{c}
# & q_0 & 0 \\
# & q_0 & 0 & 1 & 0 & 0 & # & 2 & q_7 \\
\end{array}
\]

Part 2 places the domino

\[
\begin{array}{c}
q_00 \\
2q_7
\end{array}
\]
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.

<table>
<thead>
<tr>
<th>#</th>
<th>$q_0$</th>
<th>0</th>
<th>1</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>#</td>
<td>$q_0$</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Part 2 places the domino $\left[ \begin{array}{c} q_0 \\ 0 \\ 2q_7 \end{array} \right]$.

Part 4 places the dominos $\left[ \begin{array}{c} 0 \\ 1 \\ \frac{2}{2} \end{array} \right]$ and $\left[ \begin{array}{c} \sqcup \\ \sqcup \end{array} \right]$ into $P'$ so we can extend the match.

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.
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

\[
\begin{array}{c}
\# & q_0 & 0 & 1 & 0 & 0 & \# \\
\# & q_0 & 0 & 1 & 0 & 0 & \# & 2 & q_7 & 1 & 0 & 0 & \#
\end{array}
\]

- Part 2 places the domino $\begin{bmatrix} q_0 & 0 \\ 2q_7 \end{bmatrix}$

- Part 4 places the dominos $\begin{bmatrix} 0 \\ 0 \\ 1 \\ \frac{2}{2} \end{bmatrix}$ and $\begin{bmatrix} \sqcup \\ \sqcup \end{bmatrix}$ into $P'$ so we can extend the match.

- Part 5 puts in the domino $\begin{bmatrix} \# \\ \# \end{bmatrix}$

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.
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.

\[
\begin{array}{cccccc}
# & q_0 & 0 & 1 & 0 & 0 & # \\
# & q_0 & 0 & 1 & 0 & 0 & # & 2 & q_7 & 1 & 0 & 0 & #
\end{array}
\]

Part 2 places the domino $\begin{bmatrix} q_0 \ 0 \\ 2 \ q_7 \end{bmatrix}$.

Part 4 places the dominos $\begin{bmatrix} 0 \\ 1 \\ 2 \end{bmatrix}$ and $\begin{bmatrix} \sqcup \\ \sqcup \\ \sqcup \end{bmatrix}$ into $P'$ so we can extend the match.

Part 5 puts in the domino $\begin{bmatrix} # \\ # \end{bmatrix}$.

What exactly is going on?
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:

```
  # q0 0 1 0 0 0 #
# q0 0 1 0 0 0 # 2 q7 1 0 0 #
```

Part 2 places the domino:

```
\begin{bmatrix}
  q_0 \\
  2q_7 \\
\end{bmatrix}
```

Part 4 places the dominos:

```
\begin{bmatrix}
  0 \\
  1 \\
  2 \\
\end{bmatrix}
```

and:

```
\begin{bmatrix}
  \sqcup \\
  \sqcup \\
\end{bmatrix}
```

Into $P'$ so we can extend the match.

Part 5 puts in the domino:

```
\begin{bmatrix}
  # \\
  # \\
\end{bmatrix}
```

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 $w$!
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.

\[
\begin{array}{c}
# & q_0 & 0 & 1 & 0 & 0 & # \\
# & q_0 & 0 & 1 & 0 & 0 & # & 2 & q_7 & 1 & 0 & 0 & # \\
\end{array}
\]

Part 2 places the domino $\begin{bmatrix} q_0 & 0 \\ 2 & q_7 \end{bmatrix}$.

Part 4 places the dominos $\begin{bmatrix} 0 \\ 0 \\ 1 \\ 1 \\ 2 \\ 2 \end{bmatrix}$ and $\begin{bmatrix} \sqcup \\ \sqcup \end{bmatrix}$ into $P'$ so we can extend the match.

Part 5 puts in the domino $\begin{bmatrix} # \\ # \end{bmatrix}$.

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.
For every $a \in \Gamma$, put $\left[ \frac{aq\text{accept}}{q\text{accept}} \right]$ and $\left[ \frac{q\text{accept}a}{q\text{accept}} \right]$ into $P'$.

These dominos “clean-up” by adding any symbols to the top string while adding just the state symbol to the lower string.
For every $a \in \Gamma$, 

$$\begin{align*}
\text{put } \begin{bmatrix} aq_{\text{accept}} \\ q_{\text{accept}} \end{bmatrix} \text{ and } \begin{bmatrix} q_{\text{accept}}a \\ q_{\text{accept}} \end{bmatrix} \text{ into } P' 
\end{align*}$$

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

$$\begin{align*}
\cdots \# \\
\cdots \# 2 1 q_{\text{accept}} 0 2 \# 
\end{align*}$$
For every $a \in \Gamma$, put $\begin{bmatrix} aq_{accept} \\ q_{accept} \end{bmatrix}$ and $\begin{bmatrix} q_{accept}a \\ q_{accept} \end{bmatrix}$ 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

\[ \cdots \# \]
\[ \cdots \# 2 1 q_{accept} 0 2 \# \]

After using these dominos, we end up with

\[ \cdots \# \]
\[ \cdots \# q_{accept} \# \]
For every $a \in \Gamma$, put $\begin{bmatrix} aq_{\text{accept}} \\ q_{\text{accept}} \end{bmatrix}$ and $\begin{bmatrix} q_{\text{accept}} a \\ q_{\text{accept}} \end{bmatrix}$ 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

$$\ldots \#$$

$$\ldots \# 2 1 q_{\text{accept}} 0 2 \#$$

After using these dominos, we end up with

$$\ldots \#$$

$$\ldots \# q_{\text{accept}} \#$$

Finally we add the domino

$$\begin{bmatrix} q_{\text{accept}}## \\ # \end{bmatrix}$$

to complete the match.
This concludes the construction of $P'$. 
This concludes the construction of $P'$.

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

Thus if $M$ accepts $w$, the set of MPDP dominos constructed have a solution to the MPDP problem.

But not yet to the PCP problem.
Suppose we have the MPCP instance

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

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

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!
Suppose we have the MPCCP instance

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

We let \( P \) be the collection

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

The only domino that could possibly start a match is the first one!
PCP Proof – Part 2

- Suppose we have the MPCP instance

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

- We let \( P \) be the collection

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

- 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!
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

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

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

   \[ M_A = \text{"On input } \langle I_A \rangle \text{,}\]
   
   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.
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 $A$:

$$M_A = \text{"On input } \langle I_A \rangle$$
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 = \text{"On input } \langle I_A \rangle \text{"

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 $A$:

$$M_A = \text{“On input } \langle I_A \rangle \text{”}$$

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 $A$:

   $M_A = \text{"On input } \langle I_A \rangle$ \n
   1. Algorithmically construct an input $\langle I_B \rangle$ for $M_B$, such that
      a) Either
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 $A$:

$$M_A = \text{"On input } \langle I_A \rangle \text{"}$$

1. Algorithmically construct an input $\langle I_B \rangle$ for $M_B$, such that
   a) Either
      
      $$\text{If } \langle I_A \rangle \in A \text{ then } \langle I_B \rangle \in B$$
      $$\text{If } \langle I_A \rangle \notin A \text{ then } \langle I_B \rangle \notin B$$

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

$$M_A = \text{"On input } \langle I_A \rangle\text{"}$$

1. Algorithmically construct an input $\langle I_B \rangle$ for $M_B$, such that
   a) Either
   b) or
      
      If $\langle I_A \rangle \in A$ then $\langle I_B \rangle \in B$
      If $\langle I_A \rangle \notin A$ then $\langle I_B \rangle \notin B$
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 $\langle I_B \rangle \in B$
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 $A$:

$$M_A = \text{“On input } \langle I_A \rangle$$

1. **Algorithmically** construct an input $\langle I_B \rangle$ for $M_B$, such that
   a) Either
      
      $$\begin{align*}
      &\text{If } \langle I_A \rangle \in A \text{ then } \langle I_B \rangle \in B \\
      &\text{If } \langle I_A \rangle \notin A \text{ then } \langle I_B \rangle \notin B
      \end{align*}$$

   b) or
      
      $$\begin{align*}
      &\text{If } \langle I_A \rangle \in A \text{ then } \langle I_B \rangle \notin B \\
      &\text{If } \langle I_A \rangle \notin A \text{ then } \langle I_B \rangle \in B
      \end{align*}$$

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 \( A \):

\[ M_A = \text{“On input } \langle I_A \rangle \text{”} \]

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.
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 $A$:

$$M_A = \text{“On input } \langle I_A \rangle \text{”}$$

1. Algorithmically construct an input $\langle I_B \rangle$ for $M_B$, such that
   a) Either
   
   \begin{align*}
   \text{If } \langle I_A \rangle &\in A \text{ then } \langle I_B \rangle \in B \\
   \text{If } \langle I_A \rangle &\notin A \text{ then } \langle I_B \rangle \notin B
   \end{align*}
   
   b) or

   \begin{align*}
   \text{If } \langle I_A \rangle &\in A \text{ then } \langle I_B \rangle \notin B \\
   \text{If } \langle I_A \rangle &\notin A \text{ then } \langle I_B \rangle \in B
   \end{align*}

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

Examples:

Let $f(w) = ww$ be a function. Then $f$ is computable.

Let $f(\langle n_1, n_2 \rangle) = \langle n \rangle$ where $n_1$ and $n_2$ are integers and $n = n_1 \times n_2$. Then $f$ is computable.
Computable Functions

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

Computable Function
A function $f : \Sigma^* \rightarrow \Sigma^*$ 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 $f : \Sigma^* \rightarrow \Sigma^*$.

COMPUTABLE FUNCTION
A function $f : \Sigma^* \rightarrow \Sigma^*$ 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 $f : \Sigma^* \rightarrow \Sigma^*$.

**Computable Function**
A function $f : \Sigma^* \rightarrow \Sigma^*$ 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 $f(w) \text{ def } = ww$ be a function. Then $f$ is computable.
- Let $f(\langle n_1, n_2 \rangle) \text{ def } = \langle n \rangle$ where $n_1$ and $n_2$ are integers and $n = n_1 \times n_2$. Then $f$ is computable.
Turing Machines can also compute function $f : \Sigma^* \rightarrow \Sigma^*$.

A function $f : \Sigma^* \rightarrow \Sigma^*$ 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 $f(w) \overset{\text{def}}{=} ww$ be a function. Then $f$ is computable.
Computable Functions

**Idea**

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

**Computable Function**

A function $f : \Sigma^* \rightarrow \Sigma^*$ 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 $f(w) \overset{\text{def}}{=} ww$ be a function. Then $f$ is computable.
- Let $f(\langle n_1, n_2 \rangle) \overset{\text{def}}{=} \langle n \rangle$ where $n_1$ and $n_2$ are integers and $n = n_1 \ast n_2$. Then $f$ is computable.
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 $f : \Sigma^* \rightarrow \Sigma^*$ 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.
**Definition**

Let \( A, B \subseteq \Sigma^* \). We say that language \( A \) is **mapping reducible** to language \( B \), written \( A \prec_m B \), if and only if

1. There is a computable function \( f : \Sigma^* \rightarrow \Sigma^* \) such that

   - 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 \( A \prec_m B \) 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 \( A \):

\[
N = \text{"On input } w \text{ compute } f(w) \text{ run } M \text{ on input } f(w) \text{ and output whatever } M \text{ outputs."}
\]

If \( A \prec_m B \) and \( A \) is undecidable, then \( B \) is undecidable.
**Definition**

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

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

**Theorem 5.22**

If $A \prec_m B$ 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 $A$.

$N = \text{"On input } w \text{, compute } f(w) \text{, run } M \text{ on input } f(w) \text{ and output whatever } M \text{ outputs."}$

If $A \prec_m B$ and $A$ is undecidable, then $B$ is undecidable.
**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 $f : \Sigma^* \rightarrow \Sigma^*$ 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 $A \prec_m B$, if and only if

1. There is a computable function $f : \Sigma^* \rightarrow \Sigma^*$ 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 $A \prec_m B$ 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 $A \prec_m B$, if and only if

1. There is a computable function $f : \Sigma^* \rightarrow \Sigma^*$ 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 $A \prec_m B$ 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 $f(w)$
**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 $f : \Sigma^* \rightarrow \Sigma^*$ 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 $f : \Sigma^* \rightarrow \Sigma^*$ 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.”
**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 \( f : \Sigma^* \rightarrow \Sigma^* \) 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.
THEOREM

\[ A_{TM} \lesssim_m HALT_{TM} \]
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 \( \langle M, w \rangle \)
Mapping Reducibility

**Theorem**

\[ A_{TM} \prec_m HALT_{TM} \]

**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 \( x \)
**Theorem**

\[ A_{TM} \prec_m HALT_{TM} \]

**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 \( x \).
**THEOREM**

\[ A_{TM} \lessdot_m HALT_{TM} \]

**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 \(\text{accept}\)
Mapping Reducibility

**Theorem**

\[ A_{TM} \lessdot_m HALT_{TM} \]

**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

\[ A_{TM} <_m \text{HALT}_{TM} \]

Proof.

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

\[ \langle M, w \rangle \in A_{TM} \text{ if and only if } \langle M', w' \rangle \in \text{HALT}_{TM} \]

\( M_f = \) “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 \( \langle M', w \rangle \).”
More Examples of Mapping Reducibility

- Earlier we showed
More examples of Mapping Reducibility

Earlier we showed

- $A_{TM} \lessim_m MP$
More examples of Mapping Reducibility

- Earlier we showed
  - $A_{TM} <_{m} MPCCP$
  - $MPCCP <_{m} PCP$
More examples of Mapping Reducibility

- Earlier we showed
  - $A_{TM} \leq_{m} MPCP$
  - $MPCP \leq_{m} PCP$

- In Theorem 5.4 we showed $E_{TM} \leq_{m} EQ_{TM}$. 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
  - $A_{TM} \lesssim_{m} MPCP$
  - $MPCP \lesssim_{m} PCP$

- In Theorem 5.4 we showed $E_{TM} \lesssim_{m} EQ_{TM}$. 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 $A \lesssim_{m} B$ and $B$ is Turing-recognizable, then $A$ is Turing-recognizable.
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.

**Proof**

Essentially the same as the previous proof.
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.

Useful observation: Suppose you can show $A_{TM} <_m B$. This means $A_{TM} <_m B$. Since $A_{TM}$ is Turing-unrecognizable then $B$ is Turing-unrecognizable.
Summary of Mapping Reducibility Results

Summary of Theorems

Assume that $A \leq_m B$. Then

1. If $B$ is decidable then $A$ is decidable.
Assume that \( A \lessdot_m B \). 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 $A \lessdot_m B$. 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 $A \lessdot_m B$. 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.

Useful observation:

Suppose you can show $A_{TM} \lessdot_m B$

This means $A_{TM} \lessdot_m B$

Since $A_{TM}$ is Turing-unrecognizable then $B$ is Turing-unrecognizable.
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. $A <_m B$
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 MAPPING REDUCIBILITY RESULTS

SUMMARY OF THEOREMS

Assume that $A \leq_m B$. 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. $\overline{A} \leq_m \overline{B}$

Useful observation:

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

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

Proof

We show \( A_{\text{TM} < m} \leftarrow A_{E_{TM}} \) and \( A_{\text{TM} < m} \leftarrow A_{\overline{E}_{TM}} \). These then imply the theorem.
**Example of Use**

**Theorem 5.30**

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

**Proof Idea**

We show...
**Theorem 5.30**

\[ \text{EQ}_{TM} = \{ \langle M_1, M_2 \rangle \mid M_1 \text{ and } M_2 \text{ are TMs and } L(M_1) = L(M_2) \} \] is neither Turing recognizable nor co-Turing-recognizable.

**Proof Idea**

We show

- \( A_{TM} <_m \text{EQ}_{TM} \)
\textbf{Theorem 5.30}

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

\textbf{Proof Idea}

We show

- \[ A_{TM} \not\leq_m EQ_{TM} \]
- \[ \overline{A_{TM}} \not\leq_m \overline{EQ_{TM}} \]
**Example of Use**

**Theorem 5.30**

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

**Proof Idea**

We show

- \( A_{TM} <_m EQ_{TM} \)
- \( \overline{A_{TM}} <_m \overline{EQ_{TM}} \)

These then imply the theorem.
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 $\overline{A_{TM}} <_m EQ_{TM}$)
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 $M_2$
Example of Use

Proof for $\overline{\mathcal{A}_{TM}} <_m \overline{\mathcal{EQ}_{TM}}$

We show $\mathcal{A}_{TM} <_m \mathcal{EQ}_{TM}$ (and hence $\overline{\mathcal{A}_{TM}} <_m \overline{\mathcal{EQ}_{TM}}$) 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”
Example of Use

Proof for $\overline{A_{TM}} \prec_m EQ_{TM}$

We show $A_{TM} \prec_m \overline{E Q_{TM}}$ (and hence $\overline{A_{TM}} \prec_m E Q_{TM}$) with the following $f$:

$$F = “\text{On input } \langle M, w \rangle \text{ where } M \text{ is a TM and } w \text{ is a string:}”$$

1. Construct the following two machines $M_1$ and $M_2$
   $M_1 = “\text{On any input:}”$
   1. Reject”
   $M_2 = “\text{On any input:}”$
   1. Run $M$ on $w$. If it accepts, accept.”
Example of Use

Proof for $\overline{A_{TM}} \leq_m EQ_{TM}$

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

$F = \text{“On input } \langle M, w \rangle \text{ where } M \text{ is a TM and } w \text{ is a string:} \$

1. Construct the following two machines $M_1$ and $M_2$
   $M_1 = \text{“On any input:} \$
   1. Reject”
   $M_2 = \text{“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 = \text{“On input } \langle M, w \rangle \text{ where } M \text{ is a TM and } w \text{ is a string:} $

1. Construct the following two machines $M_1$ and $M_2$
   $M_1 = \text{“On any input:}$
   1. Reject$
   $M_2 = \text{“On any input:}$
   1. Run $M$ on $w$. If it accepts, accept.$
   2. Output $\langle M_1, M_2 \rangle$.$

$M_1$ accepts nothing.
We show $A_{TM} <_m EQ_{TM}$ (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 = \text{"On input } \langle M, w \rangle \text{ where } M \text{ is a TM and } w \text{ is a string:}
\begin{enumerate}
1. Construct the following two machines $M_1$ and $M_2$
   \begin{enumerate}
   1. $M_1 = \text{"On any input:}
   \begin{enumerate}
   1. Reject"
   \end{enumerate}
   \end{enumerate}
   \begin{enumerate}
   2. $M_2 = \text{"On any input:}
   \begin{enumerate}
   1. Run $M$ on $w$. If it accepts, accept."
   \end{enumerate}
   \end{enumerate}
   \end{enumerate}
2. Output $\langle M_1, M_2 \rangle$."
\end{enumerate}$

- $M_1$ 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 \( \overline{A_{TM}} \leq_m EQ_{TM} \)**

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

\[ f = \text{“On input } \langle M, w \rangle \text{ where } M \text{ is a TM and } w \text{ is a string:} \]

1. Construct the following two machines \( M_1 \) and \( M_2 \)
   \[ M_1 = \text{“On any input:} \]
   \[ \quad 1. \text{Reject”} \]
   \[ M_2 = \text{“On any input:} \]
   \[ \quad 1. \text{Run } M \text{ on } w. \text{ If it accepts, accept.”} \]

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 \( A_{TM} <_m EQ_{TM} \) (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 $g$:

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

1. Construct the following two machines $M_1$ and $M_2$:
   - $M_1 = \text{“On any input:} \quad$
     - Accept
   - $M_2 = \text{“On any input:} \quad$
     - 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 $\overline{A_{TM}} <_m \overline{EQ_{TM}}$).
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 \( g \):

\[ G = \text{“On input } \langle M, w \rangle \text{ where } M \text{ is a TM and } w \text{ is a string:} \]
1. Construct the following two machines \( M_1 \) and \( M_2 \)

\( M_1 \) 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 \( \overline{A_{TM}} <_m \overline{EQ_{TM}} \)).
We show $A_{TM} <_m EQ_{TM}$ (and hence $\overline{A_{TM}} <_m \overline{EQ_{TM}}$) with the following $g$:

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

1. Construct the following two machines $M_1$ and $M_2$
   $M_1 = \text{“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 = \text{“On input } \langle M, w \rangle \text{ where } M \text{ is a TM and } w \text{ is a string:}$$

1. Construct the following two machines $M_1$ and $M_2$
   $M_1 = \text{“On any input:}$$
   1. Accept”
   $M_2 = \text{“On any input:}$$
   1. Run } M \text{ on } w. \text{ If it accepts, accept.”}$

2. Output $\langle M_1, M_2 \rangle.$"
**Proof for $\overline{A_{TM}} \leq_m \overline{EQ_{TM}}$**

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

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

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

   $M_2 = \text{"On any input:} $
   
   1. Run $M$ on $w$. If it accepts, accept."$

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

   $M_1$ accepts everything.
**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 $g$:

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

1. Construct the following two machines $M_1$ and $M_2$
   - $M_1 = \text{“On any input:}$$
     1. Accept”
   - $M_2 = \text{“On any input:}$$
     1. Run } M \text{ on } w. \text{ 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.
**Example of Use**

**Proof for \( A_{TM} <_m EQ_{TM} \)**

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

\[ G = \text{“On input } \langle M, w \rangle \text{ where } M \text{ is a TM and } w \text{ is a string:} \]

1. Construct the following two machines \( M_1 \) and \( M_2 \)
   \[ M_1 = \text{“On any input:} \]
   1. Accept”
   \[ M_2 = \text{“On any input:} \]
   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 \( A_{TM} <_m \overline{EQ_{TM}} \)**

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

\( G = \text{“On input } \langle M, w \rangle \text{ where } M \text{ is a TM and } w \text{ is a string:} \)

1. Construct the following two machines \( M_1 \) and \( M_2 \)
   \( M_1 = \text{“On any input:} \)
   1. Accept”
   \( M_2 = \text{“On any input:} \)
   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 \( \overline{A_{TM}} <_m \overline{EQ_{TM}} \)