Undecidable Problems and Reducibility

CSCI 2670

University of Georgia

Fall 2014
Reducibility

- We show a problem decidable/undecidable by reducing it to another problem. One type of reduction: mapping reduction.

Definition

- Let $A, B$ be languages over $\Sigma$. $A$ is mapping reducible to $B$, written $A \leq_m B$, if there is a computable function $f : \Sigma^* \rightarrow \Sigma^*$ such that
  
  $$w \in A \text{ if and only if } f(w) \in B.$$  

- Function $f$ is called the reduction of $A$ to $B$.

Definition

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

- A TM computes a function by starting with the input to the function on the tape and halting with the output of the function on the tape.
Reducibility

Definition

- Let $A$, $B$ be languages over $\Sigma$. $A$ is **mapping reducible** to $B$, written $A \leq_m B$, if there is a **computable function** $f : \Sigma^* \rightarrow \Sigma^*$ such that $w \in A$ if and only if $f(w) \in B$.

- Function $f$ is a **reduction** from $A$ to $B$.

- The idea here is that if $B$ is decidable, then $A$ must be decidable, too.

  - (The proof is shown in the next slide).

- By contraposition, if $A$ is **not** decidable, then $B$ is not decidable.

Note that $A$ could be decidable and $B$ undecidable (consider what happens when $f$ is not surjective).
Theorem

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

Proof.

- Suppose $A \leq_m B$ and $B$ is decidable. Then there exists a TM $M$ to decide $B$, and there is a computable function $f$ such that $w \in A$ if and only if $f(w) \in B$.

- We construct a decider $N$ for $A$ that acts as follows:
  - On input $w$, compute $f(w)$.
  - Run $M$ on $f(w)$.
  - If $M$ accepts $f(w)$, then accept. Otherwise, reject.
Note that $A$ could be decidable and $B$ undecidable.
Consider what happens when $f$ is not surjective.
Recall that the following problem is undecidable.

\[ A_{TM} = \{ \langle M, w \rangle | M \text{ is a TM and } M \text{ accepts } w \} \].

- Before, we called this the “halting problem”.
- Really, we should call it the “acceptance problem” for TMs.
- And we should call the language \( HALT_{TM} \) below the halting problem.

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

- We use the acceptance problem to prove \( HALT_{TM} \) undecidable.

**Theorem**

\( HALT_{TM} \) is undecidable.
Theorem

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

- Suppose we want to decide \( A_{TM} \)
- On input \( \langle M, w \rangle \), if \( M \) halts on \( w \), then it’s “safe” to run \( M \) on \( w \).
- If \( M \) accepts \( w \), then we accept \( \langle M, w \rangle \).
- If \( M \) rejects \( w \), then we reject \( \langle M, w \rangle \).
- So...

if we could decide whether a TM halts on its input, we could decide \( A_{TM} \).

- That’s the idea in the proof. We reduce \( A_{TM} \) to \( HALT_{TM} \).
  - We assume \( HALT_{TM} \) decidable.
  - We then show that a decider for \( HALT_{TM} \) can be used to decide \( A_{TM} \).
  - Since \( A_{TM} \) is undecidable, \( HALT_{TM} \) must be undecidable.
Theorem

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

Proof.

- Suppose for a proof by contradiction that \( \text{HALT}_{TM} \) is decidable. Then it has a TM \( R \) that decides it.
- We construct a TM \( S \) to decide \( A_{TM} \):
  - On input \( \langle M, w \rangle \):
    - Run \( R \) on input \( \langle M, w \rangle \).
    - If \( R \) rejects, then reject.
    - If \( R \) accepts, then run \( M \) on input \( w \).
    - Note that \( M \) must halt on \( w \).
    - If \( M \) accepts \( w \), then accept. Otherwise reject.
- \( S \) clearly decides \( A_{TM} \). But \( A_{TM} \) is undecidable....
- A contradiction, and so \( \text{HALT}_{TM} \) is not decidable.
In the previous problem, we reduced $A_{TM}$ to $HALT_{TM}$.

Since $A_{TM}$ is undecidable, $HALT_{TM}$ must be undecidable, too.

This sort of reduction is a standard technique.

We use it again below.

---

Theorem

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

---

The idea is to assume $E_{TM}$ is decidable and then use that to decide $A_{TM}$.

Given a decider $R$ for $E_{TM}$, we use it in a decider $S$ for $A_{TM}$.

Note that if the input to $S$ is $\langle M, w \rangle$, we can create a new TM $M'$ that only accepts $w$ or else nothing.

This is the secret to constructing $S$. 
Theorem

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

Proof.

- Suppose \( E_{TM} \) is decidable and let \( R \) be a decider for it.
- From \( R \), we construct a decider \( S \) for \( A_{TM} \), which works as follows.
- On input \( \langle M, w \rangle \):
  1. Construct TM \( M' \): On any input \( v \), if \( v \neq w \), then reject. Otherwise, run \( M \) on \( v \). If \( M \) accepts, then accept.
     **Note:** \( v \in L(M') \) if and only if \( v = w \) and \( w \in L(M) \).
  2. Run \( R \) on \( \langle M' \rangle \).
  3. If \( R \) accepts \( \langle M' \rangle \), then \( L(M') = \emptyset \) (meaning \( w \notin L(M) \)), and so reject.
  4. If \( R \) rejects \( \langle M' \rangle \), then \( L(M') = \{w\} \) (meaning \( w \in L(M) \)), and so accept.
- \( S \) decides \( A_{TM} \). A contradiction!
- And so \( E_{TM} \) must be undecidable.
Rice’s Theorem

Rice’s theorem asserts that all “nontrivial” properties of Turing machines are undecidable. (To determine whether a given Turing machine’s language has property $P$ is undecidable.)

**Theorem**

- Let $P$ be a language consisting of TM descriptions such that
  1. $P$ is nontrivial—it contains some, but not all, TM descriptions.
  2. $P$ is a property of the TM’s language (Here, $M_1$ and $M_2$ are any TMs.)

  Whenever $L(M_1) = L(M_2)$, we have $\langle M_1 \rangle \in P$ iff $\langle M_2 \rangle \in P$.

- Then $P$ is undecidable.
Undecidable Problems from Language Theory: $\text{REGULAR}_{TM}$

- For instance, determining whether the language of a TM is regular is undecidable.

**Theorem**

$\text{REGULAR}_{TM} = \{ \langle M \rangle | M$ is a TM and $L(M)$ is regular $\}$ is undecidable.

- We can prove this by reduction from $A_{TM}$.
- We assume $\text{REGULAR}_{TM}$ has decider $R$ and then use it to decide $A_{TM}$.
- On input $\langle M, w \rangle$ We construct a new machine $M_2$ such that $L(M_2)$ is regular iff $w \in L(M)$.
- We then run $R$ on $\langle M_2 \rangle$.
- Note that we never actually run $M_2$. We instead use $R$ to decide a property of $M_2$.
- The difficult part is knowing how to construct $M_2$ from $M$ and $w$. 
**Theorem**

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

- Given TM \( M \) and string \( w \), construct \( M_2 \) which operates as follows:
  - On input \( x \):
    - If \( x \) has form \( 1^n0^n \), then accept.
    - If not, then run \( M \) on \( w \) (not \( x \)) and accept \( x \) if \( M \) accepts \( w \).
  - \( L(M_2) = \Sigma^* \) if \( w \in L(M) \),
  - Observe that \( 1^n0^n \) is nonregular and \( \Sigma^* \) is regular.
  - \( M_2 \) recognizes a regular language if and only if \( M \) accepts \( w \).
Theorem

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

Proof.

Let \( R \) be a TM that decides \( \text{REGULAR}_{TM} \) and construct TM \( S \) to decide \( A_{TM} \). 
\( S = \) “On input \( \langle M, w \rangle \), where \( M \) is a TM and \( w \) is a string:

1. Construct the following TM \( M_2 \).
2. \( M_2 = \) “On input \( x \):
   
   ▶ If \( x \) has the form \( 1^n0^n \), accept.
   
   ▶ If \( x \) does not have this form, run \( M \) on input \( w \) and accept if \( M \) accepts \( w \).”
3. Run \( R \) on input \( \langle M_2 \rangle \).
4. If \( R \) accepts, accept; if \( R \) rejects, reject.”
Similarly, determining the following properties of TMs is undecidable.

- $CF_{TM} = \{ \langle M \rangle | M \text{ is a TM and } L(M) \text{ is context free} \}$
- $FINITE_{TM} = \{ \langle M \rangle | M \text{ is a TM and } L(M) \text{ is finite} \}$
- $DECIDABLE_{TM} = \{ \langle M \rangle | M \text{ is a TM and } L(M) \text{ is decidable} \}$
The idea is simple: if \( EQ_{TM} \) were decidable, \( E_{TM} \) also would be decidable, by giving a reduction from \( E_{TM} \) to \( EQ_{TM} \).

**Theorem**

The following language is undecidable.

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

- The \( E_{TM} \) problem is a special case of the \( EQ_{TM} \) problem wherein one of the machines is fixed to recognize the empty language.
- The idea here is to construct a TM \( M_2 \) such that \( L(M_2) = \emptyset \).
- Then use this to determine whether \( L(M_1) = \emptyset \).
Recall that $EQ_{DFA}$ is decidable.

$EQ_{TM}$ isn't, as can be proved via reduction from $E_{TM}$.

**Theorem**

The following language is undecidable.

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

**Proof.**

- Suppose that $EQ_{TM}$ is decidable and let $R$ be a decider for it.
- We construct a TM $S$ to decide $E_{TM}$ as follows.
- $S =$ “On Input $\langle M \rangle$, where $M$ is a TM:
  1. Run $R$ on input $\langle M, M_1 \rangle$, where $M_1$ is a TM that rejects all inputs.
  2. If $R$ accepts $\langle M, M_1 \rangle$, then accept ($L(M) = L(M_1) = \emptyset$);
  3. if $R$ rejects $\langle M, M_1 \rangle$, then reject ($L(M) \neq \emptyset$).”
Recall that the state, tape contents, and tape head position of a TM constitute a \textbf{configuration}.

Using configurations, we can define \textbf{computation histories}.

**Definition**

An \textbf{accepting computation history} of TM $M$ on string $w$ is a finite sequence of configurations $C_1, \ldots, C_n$, where

- $C_1$ is the start configuration of $M$ on $w$,
- $C_n$ is an accepting configuration, and
- for each $1 \leq i < n$, $C_{i+1}$ follows from $C_i$ via $M$'s transition function.

A \textbf{rejecting computation history} is defined similarly, save that $C_n$ is a rejecting configuration.

Deterministic TMs have $\leq 1$ computation history for an input $w$.

Computation histories are useful in proving properties of a restricted type of TM called a \textbf{linear bounded automaton}.
Definition

A **Linear Bounded Automaton** is a Turing machine that may only use the portion of tape originally occupied by the input string $w$. Its tapehead cannot move beyond the left- or rightmost tape cells. Hence we say that for an input of length $n$, the amount of memory available is linear in $n$.

- LBAs are restricted TMs, but they are powerful.
- The languages accepted by LBAs are called **context sensitive**.
- These are called Type-1 languages in the Chomsky Hierarchy.

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
<th>Accepting Machine</th>
<th>Grammar</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Turing Recognizable</td>
<td>Turing Machine</td>
<td>Unrestricted</td>
</tr>
<tr>
<td>1</td>
<td>Context Sensitive</td>
<td>LBA</td>
<td>Context Sensitive</td>
</tr>
<tr>
<td>2</td>
<td>Context Free</td>
<td>PDA</td>
<td>Context Free</td>
</tr>
<tr>
<td>3</td>
<td>Regular</td>
<td>DFAs</td>
<td>Regular</td>
</tr>
</tbody>
</table>

- Each class $n$ is a proper subset of class $n - 1$ (there are some caveats).
Recall that the following were decidable: $A_{DFA}$, $A_{CFG}$. (In fact, they are decidable by LBAs).

- The language $A_{TM}$ was not decidable.
- The language $A_{LBA}$ is, however.

**Theorem**

$A_{LBA} = \{\langle M, w \rangle | M \text{ is an LBA and } w \in L(M) \}$ is decidable.

To prove this we need the following lemma, which asserts that there is a finite number of configurations for an LBA with input length $n$.

**Lemma**

- Let $M$ be an LBA with $|Q| = q$, $|\Gamma| = g$, and let $w \in \Sigma^*$.
- For a tape of length $n$, there are $qng^n$ possible configurations of $M$.

So, if $M$ is a LBA and $w \in L(M)$ $w$ will be accepted within $qng^n$ steps.
Linear Bounded Automata: $A_{LBA}$ is decidable

**Theorem**

$A_{LBA} = \{ \langle M, w \rangle | M \text{ is an LBA and } w \in L(M) \}$ is decidable.

**Proof.**

We construct a TM $L$ to decide $A_{LBA}$. On input $\langle M, w \rangle$:

- Run $M$ on $w$, counting how many steps have been taken.
- If $M$ accepts (rejects) $w$ before $qng^n$ steps, then accept (reject).
- If $M$ has not halted within $qng^n$ steps, then reject.

- If the $M$ has not halted within $qng^n$ steps, it never will.
- Instead, it will begin to repeat steps it’s previously entered.
Linear Bounded Automata: $E_{LBA}$ is undecidable

- Not all problems involving LBAs are decidable.
- Although $E_{DFA}$ and $E_{CFG}$ are decidable, $E_{LBA}$ is not.

**Theorem**

$E_{LBA} = \{ \langle M \rangle | M$ is an LBA and $L(M) = \emptyset \}$ is undecidable.

- The proof is via reduction from $A_{TM}$ to $E_{LBA}$.
- From input $\langle M, w \rangle$, an LBA $B$ is constructed to accept all and only accepting computation histories of $M$ on $w$.

1. On input $x$, $B$ checks that $x$ is $C_1 \# C_2 \# \ldots \# C_n$, where $C_1$ is the start configuration of $M$ on $w$ and $C_n$ is an accepting configuration. If not, it rejects.
2. $B$ then checks to see that each $C_{i+1}$ follows from $C_i$. If so, it accepts. Otherwise, it rejects.
3. Observe this can all be done by marking symbols on $B$’s tape, and without exceeding the tape boundaries.
Linear Bounded Automata: $E_{LBA}$ is undecidable

Theorem

$E_{LBA} = \{\langle M \rangle | M$ is an LBA and $L(M) = \emptyset \}$ is undecidable.

- Note that $B$ is constructed from TM $M$ and string $w$.
- $B$ accepts all and only accepting computation histories of $M$ on $w$.
- So, if we had a decider $R$ for $E_{LBA}$ we can run it on $\langle B \rangle$.
- This could be used to decide $A_{TM}$. 
Linear Bounded Automata: $E_{LBA}$ is undecidable

**Theorem**

$E_{LBA} = \{\langle M \rangle | M \text{ is an LBA and } L(M) = \emptyset \}$ is undecidable.

**Proof.**

- Suppose $E_{LBA}$ is decidable, and let $R$ be a decider for it.
- We construct a decider $S$ for $A_{TM}$ as follows.
- On input $\langle M, w \rangle$,
- Construct $B$ as in the previous slide.
- Run $R$ on $\langle B \rangle$.
  1. If $R$ accepts, then reject (there are no accepting computation histories of $M$ on $w$).
  2. If $R$ rejects, then accept (there is an accepting computation history of $M$ on $w$).
- Observe that $B$ is never actually executed. It’s just input into $R$. 
Undecidable Languages: $ALL_{CFG}$

- Computation histories can be used to show properties of other languages.
- For instance, $ALL_{CFG}$ is undecidable.

**Theorem**

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

- To prove this, we show that for a TM $M$ and string $w$, we can construct a special PDA $D$.
- $D$ accepts all and only strings that are **not valid accepting computation histories** $C_1, C_2, \ldots, C_n$ of $M$ on $w$.
- For the PDA (for technical reasons), we encode a history with every other configuration reversed:

  $$C_1 \# C_2^R \# C_3 \# C_4^R \ldots \# C_n.$$
Undecidable Languages: $\text{ALL}_{\text{CFG}}$

**Theorem**

$\text{ALL}_{\text{CFG}} = \{ \langle G \rangle | G \text{ is a CFG and } L(G) = \Sigma^* \}$ is undecidable.

- From a TM $M$ and an input $w$, we construct a PDA $D$ that generates all strings if and only if $M$ does not accept $w$.
- So, if $M$ does accept $w$, $D$ does not generate some particular string.
- This particular string is the accepting computation history for $M$ on $w$.
- That is, $D$ is designed to generate all strings that are not accepting computation histories for $M$ on $w$. 
A computation history fails to be an accepting one if:

1. $C_1$ is not the start configuration;
2. $C_n$ is not an accepting configuration;
3. some $C_{i+1}$ doesn’t follow from $C_i$.

PDA $D$ nondeterministically chooses a failure to check.

To check $C_1$, $D$ reads through $C_1$, accepting if the first symbol is not $q_{start}$ or if the string between $q_{start}$ and the first # is not $w$.

To check $C_n$, $D$ reads through $C_n$, accepting if a state other than $q_{accept}$ appears (or more than one state appears).
To check for a failure in $\delta$, $D$ selects a $C_i$ to compare to $C_{i+1}$.

- It pushes $C_i$ onto the stack.
- It then reads through $C_{i+1}$, comparing symbols to $C_i$.
- $C_i$ is reversed relative to $C_{i+1}$. The symbols popped from the stack should match those read from $C_{i+1}$ (except for changes due to $\delta$).

- $D$ accepts if the transition is invalid.
- Observe that $D$ only rejects accepting computation histories of $M$ on $w$.
- And so if $D$ accepts all strings, then there are no such histories (and so $M$ does not accept $w$).
- As such, we could use a decider $R$ for $\text{ALL}_{CFG}$ to decide $A_{TM}$. 
Undecidable Languages: \( \text{ALL}_{\text{CFG}} \)

**Theorem**

\[ \text{ALL}_{\text{CFG}} = \{ \langle G \rangle | \text{G is a CFG and } L(G) = \Sigma^* \} \text{ is undecidable.} \]

**Proof.**

Suppose \( \text{ALL}_{\text{CFG}} \) is decidable and let \( R \) be a decider for it. We use it to construct a TM \( S \) deciding \( A_{TM} \).

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

- Construct a PDA \( D \) from \( M \) and \( w \) as described in the previous slide.
- Convert \( D \) into a CFG \( G \).
- Run \( R \) on \( \langle G \rangle \).
  - If \( R \) accepts \( \langle G \rangle \), then reject (there are no accepting histories).
  - If \( R \) rejects \( \langle G \rangle \), then accept (an accepting history exists).
The previous problems all dealt with automata. Below is an undecidable problem involving string manipulation.

(The Post Correspondence Problem (PCP))

Given a set of dominoes \( P = \{[t_1, b_1], [t_2, b_2], \ldots, [t_n, b_n]\} \), where each \( t_i \) and \( b_i \) is a string, a match for \( P \) is a sequence \( i_1, i_2, \ldots, i_m \) such that

\[
t_{i_1} t_{i_2} \ldots t_{i_m} = b_{i_1} b_{i_2} \ldots b_{i_m}.
\]

\( PCP = \{\langle P \rangle \mid P \text{ is a collection of dominoes with a match}\} \)

Example

If \( P = \{[\frac{b}{ca}], [\frac{a}{ab}], [\frac{ca}{a}], [\frac{abc}{c}]\} \), then the following is a match.

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

Note that duplicates are possible.
Find a match in the following instance of the Post Correspondence Problem.

$$\left\{ \left[ \frac{ab}{abab} \right], \left[ \frac{b}{a} \right], \left[ \frac{aba}{b} \right], \left[ \frac{aa}{a} \right] \right\}$$
The Post Correspondence Problem: PCP

Theorem

\[ PCP = \{ \langle P \rangle | P \text{ is a collection of dominoes with a match} \} \text{ is undecidable.} \]

- The idea behind showing it undecidable is to reduce \( A_{TM} \) to it.
- TM configurations are converted into domino sequences.
- For a given \( M \) and \( w, w \in L(M) \) iff a match exists for the dominoes.
- Since \( A_{TM} \) is undecidable, \( PCP \) must be, too.

(Some restrictions)

For the sake of the problem, we will assume:

- The TM \( M \) never attempts to move beyond the left of the tape.
- If \( w = \varepsilon \), string ⊘ is used for \( w \).
- The match always begins with \( \frac{t_1}{b_1} \).

These restrictions can all be done away with.
The Post Correspondence Problem: PCP

**Theorem**

\[ PCP = \{ \langle P \rangle | P \text{ is a collection of dominoes with a match} \} \text{ is undecidable.} \]

The construction of the problem proceeds in stages.

**Part 1:** Domino \( \frac{t_1}{b_1} = \frac{\#q_0 w_1 \ldots w_n \#}{w} \), where \( w = w_1 \ldots w_n \).

- \( t_1 \) must be extended using more dominoes to match \( b_1 \).
- We add dominoes to simulate moves (parts 2 and 3).
- We also add dominoes for strings unaffected by moves (part 4).

**Part 2:** For every \( \delta(q, a) = (r, b, R), q \neq q_{reject} \), construct \[ \frac{qa}{br} \].

**Part 3:** For every \( c \in \Gamma \) and \( \delta(q, a) = (r, b, L), q \neq q_{reject} \), construct \[ \frac{cqa}{rcb} \].

**Part 4:** For every \( a \in \Gamma \), construct \[ \frac{a}{a} \].
Theorem

$PCP = \{ \langle P \rangle | P \text{ is a collection of dominoes with a match} \}$ is undecidable.

- Extending $t_1$ forces beginning of a new configuration on the bottom.
- That is, we are forced to simulate the execution of $M$ on $w$.
- Dominoes are added to fill out the part of the configuration not affected by a transition.
- To mark configuration ends, we need more dominoes:

**Part 5:** Add dominoes $[#]$ and $[#\#]$. The latter allows us to represent the empty space at the right of the tape.
Suppose a TM $M$ with

- $Q = \{q_0, q_1, \ldots, q_7, q_{\text{acc}}\}$
- $\Gamma = \{0, 1, 2, 3, \sqcup\}$
- $\delta$ includes the following: $\delta(q_0, 0) = (q_7, 2, R)$,

Using steps 1-5, we can construct the following, representing a partial computation history of $M$ on $w = 0100$.

The first domino is from Part 1, the second from Part 2, and the rest from part 4 and 5.
Suppose $\delta(q_7, 1) = (q_5, 0, R)$. Then we can form...
Suppose $\delta(q_5, 0) = (q_9, 2, L)$. Then we can form...
The Post Correspondence Problem: PCP

Theorem

PCP = \{ \langle P \rangle | P \text{ is a collection of dominoes with a match} \} \text{ is undecidable.}

- Special dominoes are needed to ensure that the end sequence of dominoes match.

**Part 6:** For each $a \in \Gamma$, add dominoes $[\frac{aq_{\text{accept}}}{q_{\text{accept}}}]$ and $[\frac{q_{\text{accept}}a}{q_{\text{accept}}}]$.

(This is a technical step, intended to remove symbols around $q_{\text{accept}}$ until it is adjacent only to #).

**Part 7:** add dominoes $[\frac{q_{\text{accept}}##}{#}]$. 
Suppose we have arrived at the following:

Observe that the 0 to the right of $q_{\text{accept}}$ has been "eaten".
The Post Correspondence Problem: PCP

\[
\left[ \begin{array}{c}
q_{\text{accept}} \\
# 
\end{array} \right]
\] is needed at the very end.
In-class Questions???

Find a match in the following instance of the Post Correspondence Problem.

\[
\{ \left[ \frac{ab}{abab} \right], \left[ \frac{b}{a} \right], \left[ \frac{aba}{b} \right], \left[ \frac{aa}{a} \right] \}
\]
Mapping Reducibility

Definition

- Let \( A \) and \( B \) be languages over \( \Sigma \).
- \( A \) is **mapping reducible** to \( B \), written \( A \leq_m B \), if there is a *computable function* \( f : \Sigma^* \rightarrow \Sigma^* \) such that
  \[
  w \in A \text{ if and only if } f(w) \in B.
  \]
- Function \( f \) is a **reduction** from \( A \) to \( B \).

- Reducibility here hinges on there being a computable function.

Definition

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

- From the definition, one sees that a function is computable if and only if there is some algorithm that computes it.
Show that $\leq_m$ is a transitive relation.

**Proof.**

Suppose that $A \leq_m B$ and $B \leq_m C$. Then there are computable functions $f$ and $g$ such that $x \in A \iff f(x) \in B$ and $y \in B \iff g(y) \in C$.

Consider that composition function $h(x) = g(f(x))$. We can build a TM that computes $h$ as follows:

1. Simulate a TM for $f$ (such a TM exists because we assumed that $f$ is computable) on input $x$ and call the output $y$.
2. Simulate a TM for $g$ on $y$. The output is $h(x) = g(f(x))$.

Therefore $h$ is a computable function. Moreover, $x \in A \iff h(x) \in C$. Hence $A \leq_m C$ via the reduction function $h$. 
In previous examples, we reduced one problem $A$ to another $B$ and then leveraged a property of one to conclude something about the other.

We can formalize what we’ve been doing in a theorem.

**Theorem**

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

**Proof.**

- Suppose $A \leq_m B$ and $B$ is decidable. Then there exists a TM $M$ to decide $B$, and there is a computable function $f$ such that $w \in A$ if and only if $f(w) \in B$.
- We construct a decider $N$ for $A$ that acts as follows:
  - On input $w$, compute $f(w)$.
  - Run $M$ on $f(w)$.
  - If $M$ accepts $f(w)$, then accept. Otherwise, reject.
Mapping Reducibility

- A corollary to the previous theorem exists.
- Similarly, The theorem can be reformed for recognizability.

**Corollary**

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

**Theorem**

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

- Suppose we have a recognizer $R$ for $B$. A recognizer $S$ for $A$ would work as follows: On input $w$, compute $f(w)$ and run $R$ on $f(w)$. Accept if $R$ accepts; Reject if $R$ rejects.

**Corollary**

If $A \leq_m B$ and $A$ is not Turing-recognizable, then $B$ is not Turing-recognizable.
If $A \leq_m B$, then there’s a computable function $f : \Sigma^* \to \Sigma^*$ with

\[ w \in A \text{ if and only if } f(w) \in B. \]

Observe that this implies

\[ w \notin A \text{ if and only if } f(w) \notin B. \]

\[ w \in \overline{A} \text{ if and only if } f(w) \in \overline{B}. \]

Thus, if $A \leq_m B$, then $\overline{A} \leq_m \overline{B}$.

The converse is also true.

And so the following proposition is true.

\[ A \leq_m B \text{ if and only if } \overline{A} \leq_m \overline{B}. \]
Show that if $A$ is Turing-recognizable and $A \leq_m \overline{A}$, then $A$ is decidable.

**Proof.**

- Suppose that $A \leq_m \overline{A}$, then $\overline{A} \leq_m A$ via the same mapping reduction.
- Because $A$ is Turing-recognizable, it implies that $\overline{A}$ is Turing-recognizable.
- Then $A$ is Turing-recognizable and co-Turing-recognizable.
- Hence it implies that $A$ is decidable.
We define a function $F$ showing $A_{TM} \leq_m EQ_{TM}$.

$F$: 
- On input $\langle M, w \rangle$ construct TMs $M_1$ and $M_2$.
  - $M_1$: On any input $x$, reject.
  - $M_2$: On any input $x$, run $M$ on $w$. Accept $x$ if $M$ accepts $w$.
- Output $\langle M_1, M_2 \rangle$.

- $F$ is really a Turing machine, one that computes a function.
- $M_1$ is trivial to create, and given $M$, $M_2$ is also easy to create.
- Observe that if $w \notin L(M)$, then $M_2$ doesn’t accept any strings.
- If $w \in L(M)$, then $L(M_2)$ is the set of all strings.
- Thus $w \in L(M)$ iff $\langle M_1, M_2 \rangle \in \overline{EQ_{TM}}$ (i.e., $L(M_1) \neq L(M_2)$).
- This shows $A_{TM} \leq_m EQ_{TM}$.
- $F$ is a mapping reduction from $A_{TM}$ to $\overline{EQ_{TM}}$. 
We define a function $G$ showing $A_{TM} \leq_m EQ_{TM}$.

$G$:
- On input $\langle M, w \rangle$ construct TMs $M_1$ and $M_2$.
  - $M_1$: On any input $x$, accept.
  - $M_2$: On any input $x$, run $M$ on $w$. Accept $x$ if $M$ accepts $w$.
- Output $\langle M_1, M_2 \rangle$.

- $F$ and $G$ are very similar, save that $M_1$ accepts all inputs.
- If $w \in L(M)$, then $M_2$ accepts all strings, too.
- If $w \notin L(M)$, then $M_2$ accepts no strings.
- Thus, $w \in L(M)$ iff $\langle M_1, M_2 \rangle \in EQ_{TM}$ (i.e., $L(M_1) = L(M_2)$).
- Hence, $G$ is a mapping reduction from $A_{TM}$ to $EQ_{TM}$. 
Mapping Reducibility

Theorem

\( EQ_{TM} \) is neither Turing recognizable nor co-Turing recognizable.

Proof.

- We have \( A_{TM} \leq_m EQ_{TM} \) and \( A_{TM} \leq_m EQ_{TM} \) (earlier slides).
- From this, \( \overline{A_{TM}} \leq_m EQ_{TM} \) and \( \overline{A_{TM}} \leq_m EQ_{TM} \).
- Since \( \overline{A_{TM}} \) is not Turing recognizable, neither \( EQ_{TM} \) nor \( \overline{EQ_{TM}} \) is.
- Since \( \overline{EQ_{TM}} \) is not recognizable, then \( EQ_{TM} \) is not co-recognizable.
  - This follows by definition of co-recognizability.

What does this mean?

- We can’t reliably recognize when pairs of Turing machines have the same language (\( EQ_{TM} \) is not Turing recognizable).
- Nor can we reliably recognize when pairs of Turing machines have different languages (\( EQ_{TM} \) is not co-Turing recognizable).
Show that $EQ_{CFG}$ is co-Turing-recognizable.
Show that $EQ_{CFG}$ is co-Turing-recognizable.

Proof.

We can construct a TM $M$ which recognizes the complement of $EQ_{CFG}$:

$M = \text{“On input } \langle G, H \rangle:\n
1. \text{For each string } x \in \Sigma^* \text{ in lexicographic order:} \\
2. \text{Test whether } x \in L(G) \text{ and whether } x \in L(H), \text{ using the algorithm for } A_{CFG}. \\
3. \text{If one of the tests accepts and the other rejects, accept; otherwise, continue.”}
Show that $EQ_{CFG}$ is undecidable.
Show that $EQ_{CFG}$ is undecidable.

**Proof.**

Suppose that $EQ_{CFG}$ were decidable. We can construct a decider $M$ for $ALL_{CFG} = \{\langle G \rangle \mid G$ is a CFG and $L(G) = \Sigma^*\}$ as follows:

$M =$ “On input $\langle G \rangle$:

1. Construct a CFG $H$ such that $L(H) = \Sigma^*$.
2. Run the decider for $EQ_{CFG}$ on $\langle G, H \rangle$.
3. If it accepts, accept. If it rejects, reject.”
Let $J = \{w | w = 0x \text{ for some } x \in A_{TM}, \text{ or } w = 1y \text{ for some } y \in \overline{A_{TM}}\}$. Show that neither $J$ nor $\overline{J}$ is Turing recognizable.
1. A state in an automaton is **useless** if it is never entered on any input string. Consider the language $U_{PDA} = \{ \langle P \rangle | P \text{ is a PDA with useless states} \}$. Show that it is **decidable**. Hint: If given a PDA $P$ with state $q$, consider modifying $P$ so that $q$ is the only accept state of $P$.

2. Consider the problem of determining whether a Turing machine $M$ on input $w$ ever attempts to move its tapehead left at any point while processing $w$. Let $L = \{ \langle M, w \rangle | M \text{ attempts moves left at some point when processing } w \}$. Show that $L$ is **decidable**.
Questions for Group Discussion

A state in an automaton is **useless** if it is never entered on any input string. Consider the language $U_{PDA} = \{ \langle P \rangle | P \text{ is a PDA with useless states} \}$. Show that it is **decidable**. Hint: If given a PDA $P$ with state $q$, consider modifying $P$ so that $q$ is the only accept state of $P$.

**Proof.**

$E_{CFG}$ is decidable and so has decider $N$. We create a decider $M$ for $U_{PDA}$.

$M =$ “On input $w$:

- Scan $w$. Reject if it is not a valid representation of a PDA $P$.
- Identify the states $q_1, q_2, \ldots, q_n$ of $P$, and for each $q_i$ do the following:
  - Modify $P$ so that $q_i$ is its only accept state (call the modified PDA $P_{q_i}$).
  - Convert $P_{q_i}$ to an equivalent CFG $G_{q_i}$ using techniques from Chapter 2.
  - Run $N$ on $\langle G_{q_i} \rangle$.
  - If $N$ accepts, then **accept** $w$.
  - If $N$ rejects, then continue.
- If each $q_i$ has been processed without accepting, **reject** $w$.”
A state in an automaton is **useless** if it is never entered on any input string. Consider the language $U_{PDA} = \{ \langle P \rangle | P \text{ is a PDA with useless states} \}$. Show that it is **decidable**. Hint: If given a PDA $P$ with state $q$, consider modifying $P$ so that $q$ is the only accept state of $P$.

**Proof.**

$N$ accepts $\langle G_{q_i} \rangle$ iff $L(G_{q_i})$ is empty. However, since $L(P_{q_i}) = L(G_{q_i})$, it must be that $q_i$ is never entered (because $q_i$ is the only accept state of $P_{q_i}$). So if $N$ accepts on some $P_{q_i}$, then $P$ has a useless state, and so $w = \langle P \rangle$ should be accepted. Above, $M$ rejects only if $N$ rejects on every $\langle P_{q_i} \rangle$. 
Let $L = \{\langle M, w \rangle | M \text{ attempts moves left at some point when processing } w \}$. Show that $L$ is decidable.

Proof.

We construct a decider $R$ for $L$. It works as follows:

$R =$ “On input $x$:

1. Scan $x$, checking whether it is of the form $\langle M, w \rangle$, where $M$ is a TM and $w$ a string. If not, reject. Otherwise continue.

2. Run $M$ on $w$ for $|w| + |Q| + 1$ steps. If $M$ ever moves left within that number of steps, then accept. Otherwise reject.”
Let $L = \{ \langle M, w \rangle | M \text{ attempts moves left at some point when processing } w \}$. Show that $L$ is decidable.

**Proof.**

The idea here is that if $M$ does not move left within $w$ steps, then it must have moved right, moving passed input string $w$. At that point, it will read only blank spaces. It can move from one state to another, reading blanks, but at some point, it must return to a state it has previously been in (that point is $|w| + |Q| + 1$ steps). And so, if it hasn’t moved left within $|w| + |Q| + 1$ steps, the machine will simply repeat states (reading blanks forever). It will never move left.