Topics Covered

1. The Language $EQ_{TM}$
2. Mapping Reducibility
3. The Post Correspondence Problem

1 The Language $EQ_{TM}$

The language $EQ_{TM} = \{\{M_1, M_2\} \mid M_1$ and $M_2$ are TMs such that $L(M_1) = L(M_2)\}$ is undecidable, which we can prove by a reduction from $A_{TM}$.

**Proof that $EQ_{TM}$ is Undecidable** Suppose that $EQ_{TM}$ were decidable and let $D$ be its decider. Consider the following decider $D'$ for $A_{TM}$:

\[
D', \text{ on input } M, w:
\]

1. Find descriptions of TMs $M_1$ and $M_2$ such that $L(M_1) = L(M_2)$ if and only if $M$ accepts $w$.
2. Run $D$ on input $\langle M_1, M_2 \rangle$. Accept if $D$ rejects. Reject if $D$ rejects.

How do we accomplish step 1? Let $M_1$ be the TM that always accepts its input $x$. Let $M_2$ be a machine with the description, “On input $x$, run $M$ on $w$. If $M$ accepts, accept. Else, reject.”

To show that $D'$ is a decider for $A_{TM}$, first suppose $\langle M, w \rangle$ is in $A_{TM}$. Then $\langle M_1, M_2 \rangle$ is in $EQ_{TM}$, so $D$ accepts and $D'$ accepts. If $\langle M, w \rangle$ is not in $A_{TM}$, then $L(M_1) = \Sigma^*$ and $L(M_2) = \emptyset$. It follows that $\langle M_1, M_2 \rangle$ is not in $EQ_{TM}$, so $D$ rejects and $D'$ rejects. Because we know that $D'$ cannot exist, $EQ_{TM}$ must be undecidable.
2 Mapping Reducibility

Consider step 1 in the construction of $D'$ above: we find a pair $(M_1, M_2)$ such that $(M_1, M_2) \in EQ_{TM}$ if and only if $(M, w) \in A_{TM}$. If we let the function $f$ represent this step, we say that $f$ is a mapping reduction. We can illustrate $f$ with the following mapping:

In general, let $f : \Sigma^* \rightarrow \Sigma^*$ be a function. Let $A$ and $B$ be languages.

- $f$ is computable if there exists a TM $M$ such that for all $x \in \Sigma^*$, $M$ halts on input $x$ such that once it halts, the content of its tape is $f(x)$.

- $A \leq_m B$ if there exists a computable function $f$ such that $x \in A$ if and only if $f(x) \in B$. We say that $A$ is mapping-reducible to $B$.

For example, let $f$ be step 1 of $D'$, as illustrated above. Then $A_{TM} \leq_m EQ_{TM}$.

**Theorem** Suppose $A \leq_m B$ and $B$ is decidable. Then $A$ is decidable.

**Proof** Let $D$ be a decider for $B$. Then consider $D'$, a decider for $A$:

$$D', \text{ on input } w:$$

1. Computer $f(w) = y$.
2. Run $D$ on input $y$. Accept if $D$ accepts. Otherwise, reject.

$D'$ decides $A$. If $w \in A$, then $y = f(w) \in B$, so $D$ accepts $y$ and $D'$ accepts. If $w \notin A$, then $y = f(w) \notin B$, so $D$ rejects $y$ and $D'$ rejects. Thus, $D'$ is a decider for $A$. ■
Corollary If \( A \leq_m B \) and \( A \) is undecidable, then \( B \) is undecidable. This follows from the previous theorem; if \( B \) were decidable, \( A \) would be decidable—a contradiction.

Proof that \( EQ_{TM} \) is Undecidable Now consider proving that \( EQ_{TM} \) is undecidable via mapping-reducibility. To show that \( A_{TM} \leq_m EQ_{TM} \), we construct a TM for \( f \). On input \( \langle M, w \rangle \), the TM computes \( \langle M_1, M_2 \rangle \) as before. In particular, \( M_2 \) always accepts and \( M_2 \) has description, “On input \( x \), run \( M \) on \( w \). If \( M \) accepts, accept. Else, reject.”

In our analysis of \( f \), we first need to show that \( f \) is correct. If \( \langle M, w \rangle \in A_{TM} \), then \( \langle M_1, M_2 \rangle \in EQ_{TM} \), and if \( \langle M, w \rangle \notin A_{TM} \), then \( \langle M_1, M_2 \rangle \notin EQ_{TM} \). The reasoning is the same as in the previous proof. Next, we argue that \( f \) is computable. All that \( f \) does is construct \( M_1 \) and \( M_2 \). We have described an algorithm to do this (i.e. there is a Turing machine that can compute it), so \( f \) is computable. Therefore \( A_{TM} \leq_m EQ_{TM} \), and \( A_{TM} \) is undecidable. By the corollary, \( EQ_{TM} \) is undecidable.

Theorem Suppose \( A \leq_m B \) and \( B \) is Turing-recognizable. Then \( A \) is Turing-recognizable.

Proof Let \( R \) be a recognizer for \( B \). Then consider \( R' \), which recognizes \( A \):

\[ R', \text{ on input } w: \]

1. Compute \( f(w) = y \).
2. Run \( R \) on input \( y \). Accept if it accepts. Otherwise, reject.

\( R' \) recognizes \( A \) because if \( w \in A \), then \( f(w) \in B \), so \( R \) accepts and \( R' \) accepts. If \( w \notin A \), then \( f(w) \notin B \), so \( R \) does not accept (it will reject or loop), so \( R' \) does not accept.

Corollary If \( A \leq_m B \) and \( A \) is not Turing-recognizable, then \( B \) is not Turing-recognizable. Like the previous corollary, this is proved by contradiction. If \( B \) were Turing-recognizable, then \( A \) would be as well, a contradiction.

Theorem If \( A \leq_m B \), then \( A^C \leq_m B^C \).

Proof As seen in the earlier illustration of a mapping-reduction, the same function \( f \) works for the complements of languages. Let \( f \) be the mapping-reduction from \( A \) to \( B \). Then \( w \in A \) if and only if \( f(w) \in B \). Thus \( w \in A^C \) if and only if \( f(w) \in B^C \). Using the same \( f \), we have \( A^C \leq_m B^C \).
We’ve seen that $A_{TM} \leq_m EQ_{TM}$. This theorem implies that $A_{TM}^{C} \leq_m EQ_{TM}^{C}$. We can also show that $A_{TM} \leq_m EQ_{TM}^{C}$, which implies that $A_{TM}^{C} \leq_m EQ_{TM}$. Since $A_{TM}^{C}$ is not Turing-recognizable, it follows that $EQ_{TM}$ and $EQ_{TM}^{C}$ are also not Turing-recognizable.

**Proof that** $A_{TM} \leq_m EQ_{TM}^{C}$  
We want a function $f$ such that $f((M, w)) = (M_1, M_2)$ such that $L(M_1) \neq L(M_2)$ if and only if $M$ accepts $w$. Let $M_1$ have description, “On input $x$, reject.” Let $M_2$ have description, “On input $x$, if $x \neq w$, reject. Else, run $M$ on $w$ and do as $M$ does.”

First, consider the correctness of $f$. If $(M, w) \in A_{TM}$, then $L(M_1) \neq L(M_2)$ because $M_1$ rejects $w$ but $M_2$ accepts $w$. That is, $L(M_1) = \emptyset$ and $L(M_2) = \{w\}$, so $(M_1, M_2) \in EQ_{TM}$. Alternatively, if $(M, w) \notin A_{TM}$, then $L(M_1) = \emptyset$ and $L(M_2) = \emptyset$. Then $(M_1, M_2) \notin EQ_{TM}$.

The function $f$ is also computable. We can describe an algorithm for constructing $M_1$ and $M_2$ (although we don’t go into the details). 

As a result of this proof, we conclude that $EQ_{TM}$ and $EQ_{TM}^{C}$ are not Turing-recognizable.

### 3 The Post Correspondence Problem

In the Post correspondence problem, we consider a set of “dominoes” over an alphabet $\Sigma$. Acceptance in the language of the Post correspondence problem (PCP) corresponds to the existence of a sequence of dominoes such that the top and bottom strings are equal. For a concrete example, let $\Sigma = \{a, n\}$, and suppose we have the following dominoes:

$$
\begin{align*}
    d_1 &= \begin{bmatrix} a \\ nm \end{bmatrix}, \\
    d_2 &= \begin{bmatrix} aa \\ ann \end{bmatrix}, \\
    d_3 &= \begin{bmatrix} ann \\ a \end{bmatrix}
\end{align*}
$$

Then consider the ordering of dominoes $d_3, d_1, d_3, d_2$:

$$
\begin{bmatrix} ann \\ a \end{bmatrix} \begin{bmatrix} a \\ nm \end{bmatrix} \begin{bmatrix} ann \\ a \end{bmatrix} \begin{bmatrix} aa \\ anna \end{bmatrix}
$$

Note that the top and bottom strings are both $annaanaa$. Then $d_3, d_1, d_3, d_2$ is called a “match” and proves that the collection of dominoes $\langle d_1, d_2, d_3 \rangle$ is in PCP. Formally, let $PCP = \{\langle d_1, \ldots, d_k \rangle | \text{each } d_i \text{ is a domino over } \Sigma^* \text{ and there exists a match } i_1, \ldots, i_n \text{ such that } d_{i_1}, \ldots, d_{i_n} \text{ has matching top and bottom strings} \}$. Rather than prove that $PCP$ is undecidable, we consider a slightly modified language $MPCP = \{\langle d_1, \ldots, d_k \rangle | \text{there exists a match starting with } d_1 \}$. 

**Proof that** $MPCP$ is Undecidable  
Our proof is a reduction from $A_{TM}$. Given $\langle M, w \rangle$, we want to choose $d_1, \ldots, d_k$ such that $M$ accepts $w$ if and only if there exists a match.
using \(d_1, \ldots, d_k\) that starts with \(d_1\). Note that \(M\) accepts \(w\) if and only if \(M\)'s computation history on \(w\) is accepting. To map an instance \((M, w)\) to a set of dominoes, we will build a collection of dominoes such that the only possible matching string is an accepting computation history of \(M\) on \(w\). There are several steps in this transformation (which are also described in Sipser section 5.2).

1. First, add the domino \(d_1\), which contains the starting configuration of \(M\)'s computation history on \(w\):

\[d_1 = \begin{bmatrix} \# \\ \#q_0 w_1 w_2 \ldots w_n \# \end{bmatrix}\]

2. Next, suppose \(M\) has a transition from state \(q\) to state \(r\), where the head overwrites \(a\) with \(b\) and moves right. For each of these transitions, we add a corresponding domino:

\[\begin{bmatrix} qa \\ br \end{bmatrix}\]

3. For every transition from state \(q\) to \(r\) where the head overwrites \(a\) with \(b\) and moves left, and for every \(c \in \Gamma\), add the domino:

\[\begin{bmatrix} cqa \\ rcb \end{bmatrix}\]

4. For every \(a \in \Gamma\), add the domino:

\[\begin{bmatrix} a \\ -a \end{bmatrix}\]

5. Add the following two dominoes to denote separations between configurations, possibly with infinitely many blanks to the right side of the tape:

\[\begin{bmatrix} \# \\ \# \end{bmatrix}, \begin{bmatrix} \# \\ -\# \end{bmatrix}\]

6. Add two dominoes to ensure that if \(M\) accepts, then we include the rest of the tape in the string. We can think of the tape head as “eating” the extra symbols to the left and right of \(q_{\text{accept}}\):

\[\begin{bmatrix} aq_{\text{accept}} \\ q_{\text{accept}} \end{bmatrix}, \begin{bmatrix} q_{\text{accept}}a \\ q_{\text{accept}} \end{bmatrix}\]

7. Finally, we add a domino to complete the matching strings:

\[\begin{bmatrix} q_{\text{accept}}\#\# \\ \# \end{bmatrix}\]

The idea behind this construction is that if \(M\) accepts \(w\), then the dominoes have a match corresponding the the accepting computation history of \(M\) on \(w\). Alternatively, if there exists a match, then there is an accepting computation history.