The Emptiness Problem Revisited

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

We showed an informal proof of this during last discussion:

Informally, we assume \( R \) is a decider for \( E_{TM} \). Then we build \( S \) to decide \( A_{TM} \) by building the Turing machine \( M_1 \) and feeding it to \( R \). Finally, \( S \) outputs the opposite result of \( R \).

In fact, what we have done here is reduce the problem of \( A_{TM} \) to the complement of \( E_{TM} \). More formally, we are showing that if \( A_{TM} \leq_m E_{TM} \) and \( A_{TM} \) is undecidable, then \( E_{TM} \) is undecidable (corollary 5.23 on page 208).

Let's do this reduction more formally now, and give a computable function that shows \( A_{TM} \leq_m E_{TM} \).

First, we need to figure out what the input and output of our function needs to be. Since elements of \( A_{TM} \) are in the form \( \langle M, w \rangle \), this will be the input of our function. Since the elements of \( E_{TM} \) are in the form \( \langle M \rangle \), this will be the output of our function. This gives:

\[
F = \text{“On input } \langle M, w \rangle \text{;} \\
1. \ldots \\
2. \text{Output } \langle M' \rangle."
\]

Second, we need to figure out what we want to actually show. Remember, for mapping reducibility we need the relationship where \( \langle M, w \rangle \in A_{TM} \leftrightarrow \langle M' \rangle \in E_{TM} \), or equivalently, \( \langle M, w \rangle \in A_{TM} \leftrightarrow \)
\(M'\) \(\notin E_{TM}\) (definition 5.20 on page 207). This means, we want to construct a Turing machine \(M'\) such that when \(M\) accepts \(w\), \(M'\) is not empty. This gives:

\[
F = \text{"On input } \langle M, w \rangle \text{:}
\]

1. Construct \(M'\) as follows:
   \(M' = \text{"On input } w:\)
   - If \(M\) accepts \(w\) . . . (accept something).
   - If \(M\) does not accept \(w\) . . . (accept nothing)."
2. Output \(\langle M' \rangle\)."

We are getting closer. However, we still have some gaps to fill in. First, let's think about \(M'\) some more. Our aim is to build a Turing machine \(M'\) such that \(L(M') \neq \emptyset\) if \(M\) accepts \(w\) and \(L(M') = \emptyset\) if \(M\) rejects \(w\). We only care about the language of this Turing machine, not the simulation of it. Also, this Turing machine is created for a specific \(M\) and \(w\) pair. However, it may accept input like any other Turing machine. Therefore we have:

\[
F = \text{"On input } \langle M, w \rangle \text{:}
\]

1. Construct \(M'\) as follows:
   \(M' = \text{"On input } x:\)
   - If \(x \neq w\), reject.
   - If \(x = w\), simulate \(M\) on \(w\).
2. Output \(\langle M' \rangle\)."

Now we must decide what to do with the input of \(M'\). Remember, we want \(L(M')\) to be empty when \(M\) rejects \(w\). So let's start by rejecting all input not equal to \(w\):

\[
F = \text{"On input } \langle M, w \rangle \text{:}
\]

1. Construct \(M'\) as follows:
   \(M' = \text{"On input } x:\)
   (a) If \(x \neq w\), reject.
2. Output \(\langle M' \rangle\)."

Finally, if \(x = w\) we want to accept only if \(M\) accepts \(w\). We determine this by simulating \(M\) on \(w\). If \(M\) accepts \(w\), we must accept \(x\):

\[
F = \text{"On input } \langle M, w \rangle \text{:}
\]

1. Construct \(M'\) as follows:
   \(M' = \text{"On input } x:\)
   (a) If \(x \neq w\), reject.
   (b) If \(x = w\), simulate \(M\) on \(w\).
   (c) If \(M\) accepts \(w\), accept.
2. Output \(\langle M' \rangle\)."
This gives us our Turing-computable function $F$. However, we are not quite done. We need to show that $\langle M, w \rangle \in A_{TM} \iff \langle M' \rangle \notin E_{TM}$ holds.

Notice that if $\langle M, w \rangle \in A_{TM}$, then $M'$ will accept a single string $x = w$. Therefore, $L(M') \neq \emptyset$. This gives $\langle M, w \rangle \in A_{TM} \Rightarrow \langle M' \rangle \notin E_{TM}$.

If $\langle M' \rangle \notin E_{TM}$, then we know $L(M') \neq \emptyset$. The only string $M'$ will ever accept is $x = w$, and this happens only when $M$ accepts $w$. Therefore, we have $\langle M' \rangle \notin E_{TM} \Rightarrow (M, w) \in A_{TM}$.

Showing that $\langle M, w \rangle \in A_{TM} \iff \langle M' \rangle \notin E_{TM}$ holds may not take a lot of work, but is necessary in showing that $A_{TM} \leq_m E_{TM}$.

So now, we have proven that $E_{TM}$ is undecidable. What about $E_{TM}$? (Think about Theorem 4.22 on page 181.)

The Equivalence Problem

The equivalence problem, $EQ_{TM} = \{ \langle M_1, M_2 \rangle \mid M_1$ and $M_2$ are TMs and $L(M_1) = L(M_2) \}$, is undecidable. We will show this by showing that $E_{TM} \leq_m EQ_{TM}$ and using Corollary 5.23.

First, we need to figure out what the input and output of our function needs to be. Since elements of $E_{TM}$ are in the form $\langle M \rangle$, this will be the input of our function. Since the elements of $EQ_{TM}$ are in the form $\langle M_1, M_2 \rangle$, this will be the output of our function. This gives:

$$F = \text{"On input } \langle M \rangle:\n$$
1. . .
2. Output $\langle M, M' \rangle$.

Second, we need to figure out what we want to actually show. We want the situation where if $L(M)$ is empty, then $L(M) = L(M')$. Since $L(M)$ is empty, we have $L(M) = L(M')$ only when $L(M')$ is also empty. Therefore, we get:

$$F = \text{"On input } \langle M \rangle:\n$$
1. Construct $M'$ as follows:
   $M' = \text{"On input } x:\text{ reject."
2. Output $\langle M, M' \rangle$.

Now, we must show that $\langle M \rangle \in E_{TM} \iff \langle M, M' \rangle \in EQ_{TM}$ holds.

If $L(M)$ is empty, then $L(M) = L(M')$ since $L(M')$ is empty. This gives $\langle M \rangle \in E_{TM} \Rightarrow \langle M, M' \rangle \in EQ_{TM}$. If $L(M) = L(M')$, then $L(M)$ is empty since $L(M')$ is empty. This gives $\langle M, M' \rangle \in EQ_{TM} \Rightarrow \langle M \rangle \in E_{TM}$.

Again, these statements seem apparent, but are necessary in completing our proof.
Guide To Classifying Languages

Claim: \( L \) is decidable.

There are three methods you may use to prove this is true. The easiest is to use definition 3.6 (page 142). This states that a language is decidable if some Turing machine decides it. Therefore, you may provide a decider Turing machine \( M \) such that \( L(M) = L \) to prove \( L \) is decidable.

Alternatively, you may use theorem 4.22 (page 181). This states that a language is decidable iff it is Turing-recognizable and co-Turing recognizable. If you show that \( L \) is both recognizable and co-recognizable, you prove that \( L \) is decidable. How to prove a language is Turing-recognizable or co-Turing-recognizable is covered in the following sections.

Finally, you may use theorem 5.22 (page 208). This states that if \( A \leq_m B \) and \( B \) is decidable, then \( A \) is decidable. If you show that \( L \leq_m D \) where \( D \) is already proven to be decidable, then you prove that \( L \) is also decidable.

Claim: \( L \) is Turing-recognizable (or acceptable).

The easiest method is to use definition 3.5 (page 142). This states that a language is Turing-recognizable if some Turing machine recognizes it. Therefore, you may provide a Turing machine \( M \) such that \( L(M) = L \) to prove \( L \) is recognizable.

You may also use theorem 3.21 (page 153). This states that a language is Turing-recognizable if and only if some enumerator enumerates it. Therefore, if you provide an enumerator \( M \) such that \( L(M) = L \), then you prove \( L \) is Turing-recognizable.

We also know that every decidable language is Turing-recognizable (page 142). Therefore, if you already know \( L \) is decidable, then you know \( L \) is also Turing-recognizable.

Finally, you may use theorem 5.28 (page 209). This states that if \( A \leq_m B \) and \( B \) is Turing-recognizable, then \( A \) is Turing-recognizable. If you show that \( L \leq_m R \) where \( R \) is recognizable, you prove that \( L \) is also Turing-recognizable.

However, if you want to prove that \( L \) is just Turing-recognizable and not also decidable, you must prove that \( L \) is undecidable. How to do this is given in the following sections.

Claim: \( L \) is co-Turing-recognizable.

This is done by showing that the complement of \( L \) is Turing-recognizable. Use the methods from above to show this.

Claim: \( L \) is undecidable.

You may use theorem 4.22 (page 181). This states that a language is decidable iff it is Turing-recognizable and co-Turing recognizable. Therefore, if \( L \) is not Turing-recognizable or co-Turing recognizable, then \( L \) is not decidable. How to show this is provided in the following sections.

Finally, you may use corollary 5.23 (page 208). This states that if \( A \leq_m B \) and \( A \) is undecidable, then \( B \) is undecidable. Therefore, you must show that \( U \leq_m L \) for some undecidable language \( U \).
Claim: \( L \) is not Turing-recognizable.

You may again use theorem 4.22 (page 181). This states that a language is decidable iff it is Turing-recognizable and co-Turing recognizable. Therefore, if you know that \( L \) is undecidable and \( \overline{L} \) is recognizable, then \( L \) may not also be recognizable. This method was used on corollary 4.23 (page 182).

Finally, you may use corollary 5.29 (page 210). This states that if \( A \leq_m B \) and \( A \) is not Turing-recognizable, then \( B \) is not Turing-recognizable. Therefore, you must show that \( S \leq_m L \) for some language \( S \) which is not Turing-recognizable.

Claim: \( L \) is not co-Turing-recognizable.

This is done by showing that the complement of \( L \) is not Turing-recognizable. For example, you could use theorem 4.22 and show that \( L \) is undecidable and recognizable, meaning \( \overline{L} \) must not also be recognizable.

Summary:

I’ve tried to summarize all the methods we have covered in the following table. Please let me know if anything is missing!

<table>
<thead>
<tr>
<th>Claim:</th>
<th>Method:</th>
<th>Thm:</th>
<th>Pg:</th>
</tr>
</thead>
<tbody>
<tr>
<td>( L ) is decidable.</td>
<td>Give a decider ( M ) such that ( L(M) = L ).</td>
<td>3.6</td>
<td>142</td>
</tr>
<tr>
<td></td>
<td>Show ( L ) is recognizable and co-recognizable.</td>
<td>4.22</td>
<td>181</td>
</tr>
<tr>
<td></td>
<td>Show ( L \leq_m B ) for a decidable language ( B ).</td>
<td>5.22</td>
<td>208</td>
</tr>
<tr>
<td>( L ) is recognizable.</td>
<td>Give a Turing machine ( M ) such that ( L(M) = L ).</td>
<td>3.5</td>
<td>142</td>
</tr>
<tr>
<td></td>
<td>Give an enumerator ( M ) such that ( L(M) = L ).</td>
<td>3.21</td>
<td>153</td>
</tr>
<tr>
<td></td>
<td>Show ( L \leq_m B ) for a recognizable language ( B ).</td>
<td>5.28</td>
<td>209</td>
</tr>
<tr>
<td>( L ) is co-recognizable.</td>
<td>Show that ( \overline{L} ) is recognizable.</td>
<td>–</td>
<td>181</td>
</tr>
<tr>
<td>( L ) is undecidable.</td>
<td>Show ( L ) is not recognizable.</td>
<td>4.22</td>
<td>181</td>
</tr>
<tr>
<td></td>
<td>Show ( L ) is not co-recognizable.</td>
<td>4.22</td>
<td>181</td>
</tr>
<tr>
<td></td>
<td>Show ( A \leq_m L ) for some ( A ) which is undecidable.</td>
<td>5.23</td>
<td>208</td>
</tr>
<tr>
<td>( L ) is not recognizable.</td>
<td>Show ( L ) is undecidable &amp; co-recognizable.</td>
<td>4.22</td>
<td>181</td>
</tr>
<tr>
<td></td>
<td>Show ( A \leq_m L ) for some ( A ) which isn’t recognizable.</td>
<td>5.29</td>
<td>210</td>
</tr>
<tr>
<td>( L ) is not co-recognizable.</td>
<td>Show ( L ) is undecidable &amp; recognizable.</td>
<td>4.22</td>
<td>181</td>
</tr>
<tr>
<td></td>
<td>Show that ( \overline{L} ) is not recognizable.</td>
<td>5.29</td>
<td>210</td>
</tr>
</tbody>
</table>