A Closer Look at $A_{TM}$

Recall that $A_{TM} = \{ \langle M, w \rangle \mid M \text{ is a TM that accepts input string } w \}$.

Consider any TM $T$ that recognizes $A_{TM}$.

- This means $T$ takes input $\langle M, w \rangle$, where $M$ is a TM and $w$ is a string, and halts with accept iff $M$ accepts $w$.
- The TM simulator $Sim_{TM}$ we described earlier is an example of such a TM.

Define a corresponding TM $S$, using $T$ as a subprocedure, as follows:

$$
S = \text{“On input } \langle M \rangle, \text{ where } M \text{ is a TM:}
\begin{align*}
1. & \quad \text{Run } T \text{ on input } \langle M, \langle M \rangle \rangle. \\
2. & \quad \text{If } T \text{ accepts, reject; if } T \text{ rejects, accept.”}
\end{align*}
$$

Clearly:

- $L(S) = \{ \langle M \rangle \mid M \text{ is a TM that rejects } \langle M \rangle \}$
- I.e., $S$ recognizes the language of TM encodings for TMs that reject their own encodings.

What happens when $S$ is run with input $\langle S \rangle$?

- If $S$ accepts $\langle S \rangle$, then:
  - $T$ must reject $\langle S, \langle S \rangle \rangle$, so
  - $\langle S, \langle S \rangle \rangle$ does not belong to $A_{TM}$, so
  - $S$ does not accept $\langle S \rangle$ – Contradiction
- If $S$ rejects $\langle S \rangle$, then:
  - $T$ must accept $\langle S, \langle S \rangle \rangle$, so
  - $\langle S, \langle S \rangle \rangle$ belongs to $A_{TM}$, so
  - $S$ accepts $\langle S \rangle$ – Contradition
- Thus $S$ neither accepts nor rejects $\langle S \rangle$.
- Therefore $S$ must loop on $\langle S \rangle$.  

A Closer Look at $A_{TM}$ (Continued)

So far:

- We assumed that $T$ is an arbitrary recognizer for
  \[ A_{TM} = \{ \langle M, w \rangle \mid M \text{ is a TM that accepts input string } w \} . \]

- We defined a corresponding TM $S$ as follows:
  \[
  S = \text{“On input } \langle M \rangle, \text{ where } M \text{ is a TM:} \\
  1. \text{ Run } T \text{ on input } \langle M, \langle M \rangle \rangle. \\
  2. \text{ If } T \text{ accepts, reject; if } T \text{ rejects, accept.”}
  \]

- We showed that $S$ loops on $\langle S \rangle$.

Could $T$ be a decider?

- If it is then $S$ is a decider.

- But $S$ loops on some input, namely $\langle S \rangle$.

- Thus $S$ is not a decider.

- Therefore $T$ cannot be a decider.

Since $T$ was assumed to be an arbitrary recognizer for $A_{TM}$, we conclude that:

- No recognizer for $A_{TM}$ can be a decider.

- Therefore $A_{TM}$ is an undecidable language.
Notice the Similarity?

Undecidability of $A_{TM}$:

- $S$ is a TM that recognizes the language of TM encodings for TMs that reject their own encodings.
  - Does $S$ accept its own encoding?

Russell's Paradox:

- Let $R$ be the set of all sets that do not contain themselves as members. E.g.:
  - The set of all motorcycles is in $R$.
  - The set of all non-motorcycles is not in $R$.
  - Does $R$ contain itself as a member?

The barber paradox:

- In a certain village there is a man who is a barber. He shaves all and only those men in the village who do not shave themselves.
  - Does this barber shave himself?
A Non-Turing-Recognizable Language

Definition. A language is \(\text{co-Turing-recognizable}\) if its complement is Turing-recognizable.

Theorem. A language is decidable if and only if it is Turing-recognizable and co-Turing-recognizable.

Proof.

- "Only if" direction:
  - If \(L\) is decidable, its complement \(\overline{L}\) is decidable. (This was a homework problem.)
  - Since any decidable language is Turing-recognizable, it follows that both \(L\) and \(\overline{L}\) are Turing-recognizable.

- "If" direction:
  - Suppose both \(L\) and \(\overline{L}\) are Turing-recognizable.
  - Let \(M_L\) be a recognizer for \(L\) and let \(M_{\overline{L}}\) be a recognizer for \(\overline{L}\).
  - Consider the following TM:
    \[
    M = \text{"On input \(\langle w \rangle\):}
    \]
    \[
    1. \text{Simulate running } M_L \text{ and } M_{\overline{L}} \text{ in parallel on } w
    \]
    \[
    \text{(by using a 2-tape TM and alternately running one step of each at a time)}
    \]
    \[
    2. \text{If } M_L \text{ accepts, accept; if } M_{\overline{L}} \text{ accepts, reject."
    }
    \]
  - Every string \(w\) is either in \(L\) or \(\overline{L}\).
  - If \(w \in L\), then \(M_L\) must halt and accept it.
  - If \(w \in \overline{L}\), then \(M_{\overline{L}}\) must halt and accept it.
  - Thus this TM halts on any input \(w\).
  - Therefore this TM is a decider.
  - Since it accepts a string \(w\) iff \(w \in L\), it’s a decider for \(L\).
  - Therefore \(L\) is decidable.

Corollary. The complement of any undecidable Turing-recognizable language is non-Turing-recognizable.

Proof. Let \(L\) be undecidable and Turing-recognizable. If \(\overline{L}\) were Turing-recognizable, \(L\) would be Turing-recognizable and co-Turing-recognizable, so it would be decidable, contradicting the assumption that it is undecidable. Therefore, \(\overline{L}\) cannot be Turing-recognizable.

Corollary. \(A_{TM}\) is a non-Turing-recognizable language.

Proof. \(A_{TM}\) is Turing-recognizable since \(Sim_{TM}\) recognizes it, but, as we have just seen, it is not decidable.
The Halting Problem

The decision problem: *Given a TM $M$ and a string $w$, does $M$ halt when given input $w$?*

The corresponding language:

$$HALT_{TM} = \{\langle M, w \rangle | M \text{ is a TM and } M \text{ halts on input } w\}$$

**Theorem.** $HALT_{TM}$ is an undecidable language.

**Proof:**

- Assume for the sake of contradiction that $HALT_{TM}$ is decidable, and let $H$ be a decider for it.

- Given any TM $M$, we could then combine it with $H$ to create a decider $M'$ for the language $L(M)$ as follows:

  $$M' = \text{ "On input } w: $$
  
  1. Run $H$ on input $\langle M, w \rangle$. If it rejects, reject.
  2. Run $M$ on $w$. If it accepts, accept; otherwise reject."

- Clearly:
  
  - Since $H$ is assumed to be a decider, stage 1 terminates.
  - Since stage 2 is only run after $H$ has determined that $M$ would not loop on $w$, stage 2 also terminates.
  - Therefore $M'$ halts on all inputs.
  - Therefore $M'$ is a decider

- Also:
  
  - $M'$ accepts $w$ iff $M$ accepts $w$.
  - Therefore $L(M') = L(M)$.

- Thus the assumption that $HALT_{TM}$ is decidable allows a recognizer for any Turing-recognizable language to be converted into a decider for that language.

- Thus the assumption that $HALT_{TM}$ is decidable implies that every Turing-recognizable language is decidable.

- Since $A_{TM}$ is Turing-recognizable but not decidable, the assumption that $HALT_{TM}$ is decidable must be false.

- Therefore $HALT_{TM}$ is undecidable.
General Notion of Reducibility

Useful strategy in any problem-solving context:

- **Reduce** a problem to one or more simpler subproblems.
- Then solve the original problem by first solving these simpler subproblems.

Examples in the specific context of algorithm design:

1. Sorting a list can be reduced to the problem of finding the smallest element in any list:¹
   - Find the smallest element in the original list.
   - Remove this element to obtain a shorter list.
   - Find the smallest element in this list.
   - Etc.

2. The divide-and-conquer strategy amounts to reducing a problem involving a large object (e.g., a list) to subproblems involving objects (e.g., sublists) of about half its size. Examples:
   - quicksort
   - merge sort

3. Our proof of the undecidability of the Halting Problem was based on:
   - assuming there was a decider for it; and
   - showing how we could use such a decider, if it exists, as a subprocedure in the design of a decider for any Turing-recognizable language.

   Thus we showed that the problem of deciding any Turing-recognizable language reduces to the problem of deciding the Halting Problem.

What these all have in common:

If we can reduce a given problem \( A \) to solvable problems \( B_1, B_2, \ldots, B_k \), we can then design a procedure for solving the original problem by using solvers for \( B_1, B_2, \ldots, B_k \) as subprocedures.

Two ways to take advantage of such reductions:

1. Use solvers for the “reduced-to” (i.e., simpler) problem(s) to actually design a solver for the “reduced-from” problem.

2. Assume for the sake of contradiction that the “reduced-to” problem(s) can be solved when we know the “reduced-from” problem can’t be. This then proves that the assumption that the “reduced-to” problem(s) can be solved must be false, so the “reduced-to” problem(s) can’t be solved either. Our third example above used a reduction for this purpose.

_Important:_ It is this latter use of reductions that makes them such an valuable tool in theoretical computer science – to generate proofs by contradiction showing that certain algorithms **cannot** exist. This is the main use we make of them here.

¹This particular approach to sorting is called selection sort.
Undecidability of $E_{TM}$

The decision problem: *Given a TM $M$, is the language $M$ recognizes empty?*

The language: $E_{TM} = \{\langle M \rangle \mid M$ is a TM and $L(M) = \Phi\}$

**Theorem.** $E_{TM}$ is undecidable.

**Proof Idea:**

- We assume for the sake of contradiction that this language is decidable and show that this implies that $A_{TM}$ is decidable, which we know is false. From this contradiction we conclude that $E_{TM}$ must be undecidable.
- The argument involves showing that the problem of deciding $A_{TM}$ instances reduces to the problem of deciding $E_{TM}$ instances.
- I.e., we show that the answer to the question of decidability of $E_{TM}$ provides an answer to the question of decidability of $A_{TM}$. In particular, we show that decidability of $E_{TM}$ implies decidability of $A_{TM}$.
- The way we do this is to show how a TM can transform any $A_{TM}$ problem instance into a $E_{TM}$ problem instance in such a way that an accept/reject decision by an assumed $E_{TM}$ decider on the transformed instance gives rise to a corresponding decision on the original $A_{TM}$ instance.

Here is a high-level description of the approach:

1. Transform any $A_{TM}$ problem instance into some $E_{TM}$ problem instance.
2. Apply the assumed $E_{TM}$ decider to the transformed problem instance.
3. Use the answer provided by this decider to give an answer for the original $A_{TM}$ problem instance.

This represents a particular way to design an $A_{TM}$ decider using an $E_{TM}$ decider as a subprocedure.

**Key Challenge:** determining how the transformation in step 1 of this description should be done so that the final answers provided in step 3 are valid. Some basic observations on this transformation:

- $A_{TM}$ problem instances have the form $\langle M, w \rangle$, where $M$ is a TM and $w$ is a string.
- $E_{TM}$ problem instances have the form $\langle M \rangle$, where $M$ is a TM.
- We will use $\langle M' \rangle$ to denote the transformed version of $\langle M, w \rangle$. 

7
Undecidability of $E_{TM}$ (Continued)

What we need our transformation to do:

- Each $\langle M, w \rangle$ must be transformed to its corresponding $\langle M' \rangle$ in such a way that accept/reject decisions made by the $E_{TM}$ decider correspond (one way or the other) to the correct accept/reject decisions for $A_{TM}$.

- This means that the language of the TM $M'$ whose encoding is the transformed problem instance $\langle M' \rangle$ must be
  - empty whenever $\langle M, w \rangle \in A_{TM}$ (i.e., whenever $M$ accepts $w$)
  - non-empty whenever $\langle M, w \rangle \not\in A_{TM}$ (i.e., whenever $M$ does not accept $w$)
  - or vice-versa

Consider this description of a TM $M'$:

$M' =$ “On input $x$:
1. Run $M$ on input $w$.
2. If $M$ accepts, accept; if $M$ rejects, reject.”

Remarks:

- This TM will not actually be run or simulated. Instead, its encoding is all that will be used by the actual TM about to be described.

- $M'$ has $M$ and $w$ built into it and ignores its input $x$.

- All that matters is what language $M'$ accepts, which we examine below. Another way to design a TM that accepts exactly the same language would be to change line 2 so that if $M$ rejects $w$, this TM goes into an infinite loop.

What is $L(M')$?

- If $M$ does not accept $w$, this TM accepts no strings, so $L(M') = \Phi$ in this case.

- If $M$ accepts $w$, this TM accepts all strings, so $L(M') = \Sigma^*$ in this case.

- That is,

$$L(M') = \begin{cases} \Sigma^* & \text{if } \langle M, w \rangle \in A_{TM} \\ \Phi & \text{if } \langle M, w \rangle \not\in A_{TM}. \end{cases}$$

- Therefore $L(M')$ is non-empty exactly when $M$ accepts $w$, i.e., exactly when $\langle M, w \rangle \in A_{TM}$.  

8
Undecidability of $E_{TM}$ (Continued)

Now that we’ve identified a way to transform $A_{TM}$ problem instances into $E_{TM}$ problem instances in a way that respects membership/non-membership distinctions, we restate the theorem and give the full proof.

**Theorem.** $E_{TM}$ is undecidable.

**Proof.**

- Assume for the sake of contradiction that $E_{TM}$ is decidable and let $D_{E_{TM}}$ be a decider for it.

- Consider the following TM:

  $$D_{A_{TM}} = \text{ “On input } \langle M, w \rangle, \text{ where } M \text{ is a TM and } w \text{ is a string:}$$
  1. Construct $\langle M' \rangle$, the encoding of the following TM:
     
     “$M'$ = “On input $x$:
     1. Run $M$ on $w$.
     2. If $M$ accepts, accept; if $M$ rejects, reject.”
     
     2. Run the emptiness decider $D_{E_{TM}}$ on input $\langle M' \rangle$.
     3. If $D_{E_{TM}}$ accepts, reject; if $D_{E_{TM}}$ rejects, accept.”

- Since the construction of $\langle M' \rangle$ from $\langle M, w \rangle$ can be carried out by a TM in a finite number of steps, stage 1 terminates.

- Since $D_{E_{TM}}$ is assumed to be a decider, stage 2 terminates as well.

- Therefore this TM is a decider.

- As discussed on the previous page, $L(M')$ is empty iff $\langle M, w \rangle \notin A_{TM}$.

- Therefore this TM is a decider for $A_{TM}$

- Since $A_{TM}$ is undecidable, the original assumption that $E_{TM}$ is undecidable must be false.

Here is a diagram illustrating the design of the above TM:
Mapping Reductions

Key ingredient in the proof just given that \( E_{TM} \) is undecidable:

- showing that the problem of deciding membership in \( A_{TM} \) reduces to the problem of deciding membership in \( E_{TM} \);
- more precisely, designing the “Construct \( \langle M' \rangle \)” box in the diagram in such a way that accept/reject decisions for the transformed \( E_{TM} \) problem instance \( \langle M' \rangle \) yield correct accept/reject decisions for the original \( A_{TM} \) problem instance \( \langle M, w \rangle \).

We now isolate and formalize this notion.

Suppose that:

1. \( A \) and \( B \) are languages over an alphabet \( \Sigma \).
2. There is a function \( f : \Sigma^* \longrightarrow \Sigma^* \) such that
   - \( f \) can be computed by a TM; and
   - \( w \in A \) iff \( f(w) \in B \).

Note that this function \( f \) assigns to every member of \( A \) some member of \( B \) and it assigns to every member of \( \overline{A} \) some member of \( \overline{B} \). Thus, to test whether a given \( w \in A \), it is equivalent to test whether \( f(w) \in B \). The answer to both questions is the same.

**Definition.** A function \( f \) is **computable** if there is a transducer TM that, when given any input \( w \), halts with only \( f(w) \) on its tape.

**Definition.** If \( A, B \), and \( f \) satisfy conditions 1 and 2 above, then we say that \( f \) is a **mapping reduction** from \( A \) to \( B \) and that \( A \) is **mapping reducible** to \( B \), denoted\(^2\) \( A \leq_m B \).

This is clearly a special case of the broader notion of reducibility described earlier. When language \( A \) is mapping reducible to language \( B \), i.e., \( A \leq_m B \), then the problem of testing membership in \( A \) reduces, in the broader sense, to the problem of testing membership in \( B \).

\(^2\)A helpful intuition is to think of the inequality as representing the idea that \( A \) problem instances are “no harder than” \( B \) problem instances to solve or, equivalently, that \( B \) problem instances are “at least as hard as” \( A \) problem instances to solve. Here, by “solving a problem instance” we mean determining language membership.
Implications of Mapping Reducibility

Suppose that there is a mapping reduction $f$ from language $A$ to language $B$. The following diagram depicts how a recognizer for $A$ can be constructed by combining a TM that computes $f$ with a recognizer for $B$:

![Diagram of recognizer for A and TM for f leading to accept/reject decision]

Here is a description of the above TM, where $F$ denotes the TM that computes $f$ and $M_B$ denotes the recognizer for $B$:

$M_A =$ “On input $w$:
1. Run $F$ on $w$ to compute $f(w)$.
2. Run $M_B$ on $f(w)$. If it accepts, accept; if it rejects, reject.”

**Theorem.** Suppose $A \leq_m B$. Then:

1. If $B$ is Turing-recognizable, then $A$ is Turing-recognizable.
2. If $B$ is decidable, then $A$ is decidable.
3. If $A$ is non-Turing-recognizable, then $B$ is non-Turing-recognizable.
4. If $A$ is undecidable, then $B$ is undecidable.

**Proof.** Let $f$ denote the reduction. Recall that this means that it has the property that $f(w) \in B$ iff $w \in A$. For 1 and 2, just consider the diagram and/or description of $M_A$ given above. If $w \in A$, then $f(w) \in B$, so $M_B$ accepts $w$, so $M_A$ accepts $w$. If $w \not\in A$, then $f(w) \not\in B$, so $M_B$ does not accept $w$, so $M_A$ does not accept $w$. Therefore $M_A$ is a recognizer for $A$. Furthermore, step 1 always terminates, so if $M_B$ is a decider then so is $M_A$. Parts 3 and 4 are each just the contrapositives of parts 1 and 2, respectively, so they follow immediately.

We’ll make extensive use of part 4 of this theorem to prove undecidability of several languages.

We’ll also use part 3 to prove some languages are not Turing-recognizable.
Observations on Mapping Reducibility

Easily proved facts about $\leq_m$:

- Invariance under complement: $A \leq_m B$ if and only if $\overline{A} \leq_m \overline{B}$.
- Transitivity: If $A \leq_m B$ and $B \leq_m C$, then $A \leq_m C$.

These follow easily from the definition; you may find it a useful exercise to write down their proofs.

*Have we already used mapping reductions and not realized it?*

Yes. Examine the previous proofs of decidability we’ve covered or that are given in Chapter 4 of Sipser. Implicit in some of these proofs are the following mapping reductions:

- $A_{NFA} \leq_m A_{DFA}$ (using a mapping assigning to any NFA encoding the encoding of its corresponding equivalent DFA)
- $A_{REG} \leq_m A_{NFA}$ (using a mapping assigning to any regular expression encoding the encoding of its corresponding equivalent NFA)
- $SUB_{DFA} \leq_m E_{DFA}$ (using a mapping assigning to any $(D_1, D_2)$, where $D_1$ and $D_2$ are DFAs, the encoding of the DFA $C$ constructed so that $L(C) = L(D_1) \setminus L(D_2)$)
- $L \leq_m A_{CFG}$ for any CFL $L$ (using a mapping assigning to any string $w$ the string $(G, w)$, where $G$ is a CFG that generates $L$)
- $EQ_{DFA} \leq_m E_{DFA}$ (using a mapping assigning to any $(D_1, D_2)$, where $D_1$ and $D_2$ are DFAs, the encoding of the DFA $C$ constructed so that its language is the symmetric difference of $L(D_1)$ and $L(D_2)$)$^{3}$

---

$^{3}$The approach used in the lecture handout, which uses two “calls” to a $SUB_{DFA}$ decider, is not based on a mapping reduction; it is, however, an example of a reduction from the problem of testing membership in $EQ_{DFA}$ to the problem of testing membership in $E_{DFA}$ in the broader sense discussed earlier.
Undecidability of $E_{TM}$ Revisited

Theorem. $E_{TM}$ is undecidable.

Proof. We create a mapping reduction by essentially imitating what we did in the earlier proof. But this time we give the description of a transducer TM that transforms any $A_{TM}$ problem instance $\langle M, w \rangle$ to its corresponding $E_{TM}$ problem instance $\langle M' \rangle$:

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

1. Construct $\langle M' \rangle$, where $M'$ is the following TM:
   $$M' = \text{ “On input } x:\n   1. \text{ Run } M \text{ on } w.$$
   2. If $M$ accepts, accept; if $M$ rejects, reject.”
2. Output $\langle M' \rangle$.

However, recall from the earlier proof that

- $L(M')$ is non-empty iff $M$ accepts $w$, so
- $\langle M' \rangle \not\in E_{TM}$ iff $\langle M, w \rangle \in A_{TM}$, so
- this transformation is not a mapping reduction from $A_{TM}$ to $E_{TM}$.

However, it is a mapping reduction from $A_{TM}$ to $\overline{E_{TM}}$ since $\langle M' \rangle \in \overline{E_{TM}}$ iff $\langle M, w \rangle \in A_{TM}$.

Therefore:

- $A_{TM} \leq_m \overline{E_{TM}}$, so
- $\overline{E_{TM}}$ is undecidable since $A_{TM}$ is (by part 4 of the theorem on reducibility implications), so
- $E_{TM}$ is undecidable since the complement of a undecidable language is undecidable (which follows from the fact that the complement of a decidable language is decidable).

If we had not simply cited the theorem on reducibility implications we could have gone through a few additional steps to obtain a self-contained proof by contradiction that $\overline{E_{TM}}$ is undecidable since $A_{TM}$ is undecidable. Here is a diagram that essentially illustrates that full argument:

![Diagram of reduction process](attachment://reduction_diagram.png)
Undecidability of $\text{REGULAR}_{\text{TM}}$

The decision problem: Given TM $M$, is the language recognized by $M$ regular?

The language: $\text{REGULAR}_{\text{TM}} = \{ \langle M \rangle \mid M$ is a TM and $L(M)$ is regular $\}$

**Theorem.** $\text{REGULAR}_{\text{TM}}$ is undecidable.

**Proof.** We show that $A_{\text{TM}} \leq_m \text{REGULAR}_{\text{TM}}$ and the result follows immediately from part 4 of the theorem on reducibility implications since $A_{\text{TM}}$ is undecidable.

Here is the description of a transducer TM that transforms any $A_{\text{TM}}$ problem instance $\langle M, w \rangle$ to its corresponding $\text{REGULAR}_{\text{TM}}$ problem instance $\langle M' \rangle$.

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

1. Construct $\langle M' \rangle$, where $M'$ is the following TM:
   
   $M' =$ “On input $x$:
   
   1. If $x$ has the form $0^n1^n$ for some $n \geq 0$ accept.
   2. Run $M$ on input $w$.
   3. If $M$ accepts, accept; if $M$ rejects, reject.”

2. Output $\langle M' \rangle$.”

What is $L(M')$?

- In its stage 1, it always accepts any string in $\{0^n1^n \mid n \geq 0\}$.
- In addition, whenever $M$ accepts $w$ it accepts all other strings in its stage 2.
- Thus

$$L(M') = \begin{cases} 
\text{the regular language } \Sigma^* & \text{if } \langle M, w \rangle \in A_{\text{TM}} \\
\text{the non-regular language } \{0^n1^n \mid n \geq 0\} & \text{if } \langle M, w \rangle \notin A_{\text{TM}}.
\end{cases}$$

Therefore $M' \in \text{REGULAR}_{\text{TM}}$ iff $\langle M, w \rangle \in A_{\text{TM}}$, proving that $A_{\text{TM}} \leq_m \text{REGULAR}_{\text{TM}}$. Since $A_{\text{TM}}$ is undecidable, $\text{REGULAR}_{\text{TM}}$ must also be undecidable.

Here is a diagram that summarizes the full argument by contradiction proving that $\text{REGULAR}_{\text{TM}}$ is undecidable since $A_{\text{TM}}$ is undecidable:

![Diagram showing the process of transforming a TM problem instance into a REGULAR_TM problem instance and vice versa.](image-url)
Undecidability of $EQ_{TM}$

The decision problem: Given two TMs $M_1$ and $M_2$, are they equivalent?

The language: $EQ_{TM} = \{\langle M_1, M_2 \rangle \mid M_1$ and $M_2$ are TMs and $L(M_1) = L(M_2)\}$

**Theorem.** $EQ_{TM}$ is undecidable.

**Proof.** We show that $E_{TM} \leq_m EQ_{TM}$ and the result follows immediately from part 4 of the theorem on reducibility implications since $E_{TM}$ is undecidable.

Here is the description of a transducer TM that transforms any $E_{TM}$ problem instance $\langle M \rangle$ to its corresponding $EQ_{TM}$ problem instance $\langle M_1, M_2 \rangle$.

$F = \text{"On input } \langle M \rangle, \text{ where } M \text{ is a TM:}\$

1. Construct $\langle M, M_\phi \rangle$, where $M_\phi$ is the following TM:
   $M_\phi = \text{"On input } x:\$
   
   1. reject."
   
   2. Output $\langle M, M_\phi \rangle.$"

$M_\phi$ is just a trivial TM that rejects all inputs.

Clearly:

- $\langle M \rangle \in E_{TM}$ iff $L(M) = \Phi = L(M_\phi)$.
- Therefore $\langle M \rangle \in E_{TM}$ iff $\langle M, M_\phi \rangle \in EQ_{TM}$.
- Thus $E_{TM} \leq_m EQ_{TM}$.
- Therefore $EQ_{TM}$ is undecidable since $E_{TM}$ is.

Here is a diagram that summarizes the full argument by contradiction proving that $EQ_{TM}$ is undecidable since $E_{TM}$ is undecidable.

\[\text{Diagram: } E_{TM} \text{ Decider} \quad \langle M \rangle \quad \text{Combine with } \langle M_\phi \rangle \quad \langle M, M_\phi \rangle \quad EQ_{TM} \text{ Decider} \quad \text{accept/reject}\]
Every Turing-Recognizable Language Reduces to $\text{HALT}_{\text{TM}}$

Recall that $\text{HALT}_{\text{TM}} = \{ \langle M, w \rangle \mid M$ is a TM and $M$ halts on input $w \}$

The proof we gave earlier for the undecidability of $\text{HALT}_{\text{TM}}$ was not based on a mapping reduction. Now we prove the following theorem, from which it follows immediately that $\text{HALT}_{\text{TM}}$ is undecidable by choosing for $L$ any undecidable Turing-recognizable language.

**Theorem.** Let $L$ be any Turing-recognizable language. Then $L \leq_m \text{HALT}_{\text{TM}}$.

**Proof.** Let $M$ be a recognizer for $L$. Here is the description of a transducer TM that transforms any string $w$ in $L$ to a string in $\text{HALT}_{\text{TM}}$:

$$F = \text{"On input } w:\}
1. \text{ Construct } \langle M', w \rangle, \text{ where } M' \text{ is the following TM:}
   \begin{align*}
   M' &= \text{"On input } x:\}
       \begin{align*}
       &1. \text{ Run } M \text{ on input } x. \\
       &2. \text{ If } M \text{ accepts, accept; if } M \text{ rejects, loop forever."}
   \end{align*}
   \end{align*}
   \text{"Output } \langle M', w \rangle."
\text{"}

Observe that:

- If $w \in L$:
  - $M$ accepts $w$, so
  - $M'$ halts and accepts $w$, so
  - $\langle M', w \rangle \in \text{HALT}_{\text{TM}}$.

- If $w \notin L$:
  - $M$ does not accept $w$ (either by rejecting or looping), so
  - $M'$ loops on $w$, so
  - $\langle M', w \rangle \notin \text{HALT}_{\text{TM}}$.

Therefore $L \leq_m \text{HALT}_{\text{TM}}$. 

16
A Non-Turing-Recognizable, Non-Co-Turing-Recognizable Language

Recall that $EQ_{TM} = \{(M_1, M_2) \mid M_1$ and $M_2$ are TMs and $L(M_1) = L(M_2)\}$.

**Theorem.** $EQ_{TM}$ is neither Turing-recognizable nor co-Turing-recognizable.

**Proof.** We break this into two parts, first proving that $EQ_{TM}$ is not Turing-recognizable, then proving that its complement is not Turing-recognizable.

**Lemma 1.** $EQ_{TM}$ is not Turing-recognizable.

**Proof.** We show that $\overline{A_{TM}} \leq_m EQ_{TM}$. Since $\overline{A_{TM}}$ is not Turing-recognizable, it will then follow from part 3 of the theorem on reducibility implications that $EQ_{TM}$ is not Turing-recognizable.

Consider this transducer TM mapping $\overline{A_{TM}}$ problem instances $\langle M, w \rangle$ to $EQ_{TM}$ problem instances, which have the form $\langle M_1, M_2 \rangle$:

- $F$ = “On input $\langle M, w \rangle$ where $M$ is a TM and $w$ is a string:
  0. If the input is not a valid encoding $\langle M, w \rangle$, output $\langle T, T \rangle$, where $T$ is any convenient TM (e.g., $M_\Phi$, defined below).
  1. Construct $\langle M', M_\Phi \rangle$, where $M'$ and $M_\Phi$ are the following TMs:
     - $M'$ = “On input $x$:
       1. Run $M$ on input $w$.
       2. If $M$ accepts, accept; if $M$ rejects, reject.”
     - $M_\Phi$ = “On input $x$:
       1. reject.”
     2. Output $\langle M', M_\Phi \rangle$.”

Note:

- For completeness we have included a stage 0 just to handle the case when the input string is not a valid encoding of any TM/string combination. Generally, even when such a stage is necessary it is ignored in other TM descriptions, with the tacit understanding that there is a simple way to deal with invalid input strings like this without spelling it out explicitly.
- In this case, the mapping needs to produce a string that belongs to $EQ_{TM}$.
- If the input is in $\overline{A_{TM}}$ because it fails to be a valid encoding of any $\langle M, w \rangle$, then stage 0 guarantees that the corresponding output string $\langle T, T \rangle$ belongs to $EQ_{TM}$, as desired.

Continuing with the proof, we first examine $L(M')$. Clearly,

$$L(M') = \begin{cases} 
\Sigma^* & \text{if } \langle M, w \rangle \in A_{TM} \\
\Phi & \text{if } \langle M, w \rangle \notin A_{TM}.
\end{cases}$$
A Non-Turing-Recognizable, Non-Co-Turing-Recognizable Language
(Continued)

Thus (restricting attention to valid encodings \( \langle M, w \rangle \)) we see that

\[
\langle M, w \rangle \in \overline{A_{TM}} \quad \Rightarrow \quad \langle M, w \rangle \notin A_{TM}
\]

\[
\Rightarrow \quad L(M') = \Phi = L(M_\Phi)
\]

\[
\Rightarrow \quad \langle M', M_\Phi \rangle \in EQ_{TM}
\]

and

\[
\langle M, w \rangle \notin \overline{A_{TM}} \quad \Rightarrow \quad \langle M, w \rangle \in A_{TM}
\]

\[
\Rightarrow \quad L(M') = \Sigma^* \neq \Phi = L(M_\Phi)
\]

\[
\Rightarrow \quad \langle M', M_\Phi \rangle \notin EQ_{TM}.
\]

Therefore, in all cases, the input string to the TM \( F \) belongs to \( \overline{A_{TM}} \) iff the output string from \( F \) belongs to \( EQ_{TM} \), so \( F \) is a mapping reduction \( \overline{A_{TM}} \leq_m EQ_{TM} \). Since \( \overline{A_{TM}} \) is not Turing-recognizable, it follows that \( EQ_{TM} \) is not Turing-recognizable.

Here is a diagram that summarizes the full argument by contradiction proving that \( EQ_{TM} \) cannot be Turing-recognizable since a recognizer for it could be used as a subprocedure to construct a recognizer for \( A_{TM} \). (This diagram ignores the invalid-encoding case handled by stage 0).

![Diagram](http://example.com/diagram.png)
A Non-Turing-Recognizable, Non-Co-Turing-Recognizable Language (Continued)

**Lemma 2.** $\overline{EQ_{\text{TM}}}$ is not Turing-recognizable.

*Proof.* As in the proof of Lemma 1 we could construct from scratch a mapping reduction from some non-Turing-recognizable language (we now know of two: $\overline{A_{\text{TM}}}$ and $\overline{EQ_{\text{TM}}}$) to $\overline{EQ_{\text{TM}}}$. Instead we will take advantage of mapping reductions already derived.

Recall that:

- we proved the undecidability of $E_{\text{TM}}$ by constructing a mapping reduction $A_{\text{TM}} \leq_m E_{\text{TM}}$; and
- we proved the undecidability of $EQ_{\text{TM}}$ by constructing a mapping reduction $E_{\text{TM}} \leq_m EQ_{\text{TM}}$.

Therefore:

- by transitivity, $A_{\text{TM}} \leq_m EQ_{\text{TM}}$,
- which is equivalent to $A_{\text{TM}} \leq_m \overline{EQ_{\text{TM}}}$,
- so it follows that $\overline{EQ_{\text{TM}}}$ is not Turing-recognizable since $\overline{A_{\text{TM}}}$ is not Turing-recognizable.
Summary of Mapping Reductions
Explicitly Described in This Handout

- $A_{TM} \leq_m E_{TM}$
- $A_{TM} \leq_m REGULAR_{TM}$
- $E_{TM} \leq_m EQ_{TM}$
- $L \leq_m HALT_{TM}$ for any Turing-recognizable language $L$
- $\overline{A_{TM}} \leq_m EQ_{TM}$
Designing Mapping Reductions: Two Examples

Consider the language
\[ L = \{ \langle M \rangle \mid M \text{ is a TM and } |L(M)| = 5 \} . \]

Try to design mapping reductions
- \( A_{TM} \leq_m L \) and
- \( \overline{A_{TM}} \leq_m L \).

Need to fill in this template, where \( F \) is the TM implementing the desired mapping reduction:\(^4\)

\( F = \) “On input \( \langle M, w \rangle \) where \( M \) is a TM and \( w \) is a string:
1. Construct \( \langle M' \rangle \), for the following TM:
   \( M' = \) “On input \( x \):
   .
   .
   .
   ”
2. Output \( \langle M' \rangle \).”

To prove \( A_{TM} \leq_m L \):
- Want \( \langle M, w \rangle \in A_{TM} \) iff \( \langle M' \rangle \in L \).
- Equivalently, want \( M' \) to accept exactly 5 strings exactly when \( M \) accepts \( w \).

Can we design such an \( M' \)?

To prove \( \overline{A_{TM}} \leq_m L \):
- Want \( \langle M, w \rangle \in \overline{A_{TM}} \) iff \( \langle M' \rangle \in L \).
- Equivalently, want \( M' \) to accept exactly 5 strings exactly when \( M \) does not accept \( w \).

Can we design such an \( M' \)?

In addition, if either or both of these mapping reductions can be shown to exist, what can we conclude about \( L \)?

\(^4\)For simplicity, we ignore the invalid-input case.