Reducibility

• Now we examine several additional unsolvable problems.
• In doing so we introduce the primary method for proving that problems are computationally unsolvable.
• It is called *reducibility*.

A reduction is a way of converting one problem into another problem in such a way that a solution to the second problem can be used to solve the first problem.

When A is reducible to B, solving A cannot be harder that solving B because a solution to B gives a solution to A.

In terms of computability theory, if A is reducible to B and B is decidable then A also is decidable.

Equivalently, if A is undecidable and reducible to B, B is undecidable.

This is the key to proving that various problems are undecidable.

Our method for proving that a problem is undecidable will be: show that some other problem already known to be undecidable reduces to it.

We will consider the following problems (~ as membership in languages):

\[
\begin{align*}
\text{HALT}_{TM} &= \{ <M,w> : M \text{ is a TM that halts on input string } w \}, \\
E_{TM} &= \{ <M> : M \text{ is a TM such that } L(M) = \emptyset \}, \\
EQ_{TM} &= \{ <M_1,M_2> : M_1, M_2 \text{ are TMs with } L(M_1) = L(M_2) \}.
\end{align*}
\]
The Halting Problem for TMs.

• We have seen that the acceptance problem for TMs is undecidable
  \[ A_{TM} = \{ <M, w> : M \text{ is a TM that accepts input string } w \} \].

**Theorem:** \( A_{TM} \) is undecidable.

• Consider the problem determining whether a Turing machine halts (by accepting or rejecting) on a given input.
  \[ \text{HALT}_{TM} = \{ <M, w> : M \text{ is a TM that halts on input string } w \} \].

**Theorem 1:** \( \text{HALT}_{TM} \) is undecidable.

• We use undecidability of \( A_{TM} \) to prove the undecidability of \( \text{HALT}_{TM} \) by reducing \( A_{TM} \) to \( \text{HALT}_{TM} \).

• Let assume that TM \( R \) decides \( \text{HALT}_{TM} \). We construct a TM \( S \) to decide \( A_{TM} \).

\( S = \) “on input \( <M, w> \), where \( M \) is a TM and \( w \) is a string:

1. Run TM \( R \) on input \( <M, w> \).
2. If \( R \) rejects, reject.
3. If \( R \) accepts, simulate \( M \) on \( w \) until it halts.
4. If \( M \) has accepted, accept; if \( M \) rejected, reject.”

Clearly, if \( R \) decides \( \text{HALT}_{TM} \), then \( S \) decides \( A_{TM} \). Because \( A_{TM} \) is undecidable, \( \text{HALT}_{TM} \) is undecidable too.
The Emptiness Problem for the Language of a TM.

\[ E_{TM} = \{ <M> : M \text{ is a TM such that } L(M) = \emptyset \} . \]

**Theorem 2:** \( E_{TM} \) is undecidable.

- Let assume that TM \( R \) decides \( E_{TM} \). We construct a TM \( S \) to decide \( A_{TM} \).

- Idea is for \( S \) to run \( R \) on input \( <M> \) and see whether it accepts. If it does then \( L(M) \) is empty and hence \( M \) does not accept \( w \). But if \( M \) rejects \( w \) we still do not know whether \( M \) accepts \( w \).

- Instead of running \( R \) on \( <M> \) we run \( R \) on a modification of \( <M> \) (\( <M_1> \)). The only string \( M_1 \) accepts is \( w \), so its language is nonempty if and only if it accepts \( w \).

\( S = \) “on input \( <M,w> \), an encoding of a TM \( M \) and a string \( w \):

1. Use the description of \( M \) and \( w \) to construct the following TM \( M_1 \).
   \[ M_1 = \text{“on input } x \text{:} \]
   1. If \( x \neq w \), reject.
   2. If \( x = w \), run \( M \) on input \( w \) and accept if \( M \) does.”

2. Run \( R \) on input \( <M_1> \).

3. If \( R \) accepts, reject; if \( R \) rejects, accept.”

- The test whether \( x = w \) is obvious; scan the input and compare it character by character with \( w \) to determine whether they are the same.

- Note that \( S \) must be able to compute a description of \( M_1 \) from a description of \( M \) and \( w \). It is able because it needs only add extra states to \( M \) that perform the \( x = w \) test.

- If \( R \) were a decider for \( E_{TM} \), \( S \) would be a decider for \( A_{TM} \) which is impossible.
The Equivalence Problem for TMs.

\[ EQ_{TM} = \{ <M_1, M_2> : M_1, M_2 \text{ are TMs with } L(M_1) = L(M_2) \}. \]

**Theorem 3:** \( EQ_{TM} \) is undecidable.

- We could prove it by a reduction from \( A_{TM} \), but we use this opportunity to give an example of an undecidability proof by reduction from \( E_{TM} \).
- Let TM \( R \) decides \( EQ_{TM} \) and construct TM \( S \) to decide \( E_{TM} \) as follows.

\[ S = \text{“on input } <M>, \text{ an encoding of a TM } M:\]

1. Run \( R \) on input \( <M,M1> \), where \( M1 \) is a TM that rejects all inputs.
   2. If \( R \) accepts, accept; if \( R \) rejects, reject.”

- 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.
- This idea makes giving the reduction easy.
- So, If \( R \) were a decider for \( EQ_{TM} \), \( S \) would be a decider for \( E_{TM} \), which is impossible.
- One can also show that \( EQ_{TM} \) is neither Turing-recognizable nor co-Turing-recognizable.

*In the textbook, a simple problem called Post Correspondence Problem is shown to be unsolvable by algorithms.*