Reducibility

... Does that mean that the problems are getting smaller?

No.
Reducibility

The primary method for proving that problems are computationally unsolvable

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

If A reduces to B and B is decidable, then A is decidable.

(E.g., multiplication reduces to addition)

If A reduces to B and A is undecidable, then B is undecidable.

(E.g., traveling at the speed of light reduces to having enough propulsion)
Reducibility Proofs

General Form

... Generally used to prove that some problem is undecidable.

Given: A is undecidable
Goal: Prove that B is undecidable

1. Assume that B is decidable.
2. Reduce A to B. i.e., Construct some machine that uses B to solve A.
3. If A is undecidable, then this is a contradiction, thus the assumption is wrong and B must be undecidable.
Overview of Undecidable Languages

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

\[
\text{E}_{TM} = \{\langle M \rangle \mid M \text{ is a TM and } L(M) = \emptyset\}
\]

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

\[
\text{EQ}_{TM} = \{\langle M_1, M_2 \rangle \mid M_1 \text{ and } M_2 \text{ are TMs and } L(M_1) = L(M_2)\}
\]
Theorem - \( \text{HALT}_{TM} \) is undecidable

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

Idea: Proof by contradiction.

- **Assume** that \( \text{HALT}_{TM} \) is decidable.
- Logically **show** that \( A_{TM} \) is decidable, which is a **contradiction**.
- **Therefore**, our assumption was wrong, and \( \text{HALT}_{TM} \) is not decidable.

(We often use letters \( R \) and \( S \). \( R \) is the assumption. \( S \) is the machine that we build which is a contradiction.)

Let’s assume that TM \( R \) decides \( \text{HALT}_{TM} \). Construct TM \( S \) to decide \( A_{TM} \) as follows.

\( S = \) “On input \( \langle M, w \rangle \), an encoding of a TM \( M \) and a string \( w \):

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

Clearly, if \( R \) decides \( \text{HALT}_{TM} \), then \( S \) decides \( A_{TM} \). This is the contradiction. Because \( A_{TM} \) is undecidable, \( \text{HALT}_{TM} \) also **must** be undecidable.
Theorem - $E_{TM}$ is undecidable

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

Proof by contradiction; Reduction from $A_{TM}$.

Let $R$ be a TM that decides $E_{TM}$. We use $R$ to construct TM $S$ that decides $A_{TM}$.

Instead of running $R$ on $\langle M \rangle$, we run $R$ on a modification of $\langle M \rangle$. We modify $\langle M \rangle$ to guarantee that $M$ rejects all strings except $w$, but on input $w$ it works as usual. This modified machine is $M_1$.

$M_1$ = “On input $x$:
1. If $x \neq w$, reject.
2. If $x = w$, run $M$ on input $w$ and accept if $M$ does.”

Assume that TM $R$ decides $E_{TM}$ and construct TM $S$ that decides $A_{TM}$ as follows.

$S$ = “On input $\langle M, w \rangle$, an encoding of a TM $M$ and a string $w$:
1. Use the description of $M$ and $w$ to construct the TM $M_1$, just described.
2. Run $R$ on input $\langle M_1 \rangle$.
3. If $R$ accepts, reject; if $R$ rejects, accept.”
Theorem - $E_{TM}$ is undecidable (cont.)

$M_1 = “On \text{ input } x:\n1. \text{ If } x \neq w, \text{ reject.}\n2. \text{ If } x = w, \text{ run } M \text{ on input } w \text{ and accept if } M\text{ does.”}\n
Assume that TM $R$ decides $E_{TM}$ and construct TM $S$ that decides $A_{TM}$ as follows.

$S = “On \text{ input } \langle M, w \rangle, \text{ an encoding of a TM } M \text{ and a string } w:\n1. \text{ Use the description of } M \text{ and } w \text{ to construct the TM } M_1 \text{ just described.}\n2. \text{ Run } R \text{ on input } \langle M_1 \rangle.\n3. \text{ If } R \text{ accepts, reject; if } R \text{ rejects, accept.”}\n
Note: $S$ must actually be able to compute a description of $M_1$ from a description of $M$ and $w$. It is able to do so because it only needs to 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 $A_{TM}$.

A decider for $A_{TM}$ cannot exist, so we know that $E_{TM}$ must be undecidable.

To better understand, think about this:

- What does it mean if $R$ accepts, and what does that mean for $S$?
- What if $R$ rejects?
Theorem - \( \text{REGULAR}_{\text{TM}} \) is undecidable

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

PROOF: Let \( R \) be a TM that decides \( \text{REGULAR}_{\text{TM}} \) and construct TM \( S \) to decide \( A_{\text{TM}} \).

\( S = \) “On input \( \langle M, w \rangle \), where \( M \) is a TM and \( w \) is a string:

1. Construct the following TM \( M_2 \).
   \( M_2 = \) “On input \( x \):
   a. If \( x \) has the form \( 0^n1^n \), accept.
   b. If \( x \) does not have this form, run \( M \) on input \( w \) and accept if \( M \) accepts \( w \).”
2. Run \( R \) on input \( \langle M_2 \rangle \).
3. If \( R \) accepts, accept; if \( R \) rejects, reject.”

This proof is complicated. Don’t get thrown off by \( x \) or \( M_2 \).

We are not trying to feed \( w \) into \( x \)!

Q: If \( w \) is not in \( M \), what is \( L(M_2) \)?
A: \( 0^n1^n \), which is not Regular!

Q: If \( w \) is in \( M \), what is \( L(M_2) \)?
A: \( \Sigma^* \), which is Regular!

If \( R \) is decidable, then it can be used to determine whether or not \( w \) is in \( M \), making \( A_{\text{TM}} \) decidable.

This is a contradiction.
Theorem - $EQ_{TM}$ is undecidable

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

PROOF: Let TM $R$ decide $EQ_{TM}$ and construct TM $S$ to decide $E_{TM}$ as follows.

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

1. Run $R$ on input $\langle M, M_1 \rangle$, where $M_1$ is a TM that rejects all inputs.
2. If $R$ accepts, accept; if $R$ rejects, reject.”

If $R$ decides $EQ_{TM}$, then $S$ decides $E_{TM}$.

But $E_{TM}$ is undecidable by Theorem 5.2, so $EQ_{TM}$ also must be undecidable.

Observations:

Shows a reduction from $E_{TM}$ to $EQ_{TM}$.

Important implication for developing intuition about the nature of Reducibility.

If $EQ_{TM}$ were decidable:

- $E_{TM}$ would be decidable.
- $A_{TM}$ would be decidable!

Conversely:

- $A_{TM}$ reduced to $E_{TM}$.
- $E_{TM}$ reduced to $EQ_{TM}$.
Is anything decidable?!?

Well actually...

YES!
A linear bounded automaton is a restricted type of Turing machine wherein the tape head isn’t permitted to move off the portion of the tape containing the input.

Have limited power, yet are still quite powerful. LBAs are deciders for $A_{DFA}$, $A_{CFG}$, $E_{DFA}$, and $E_{CFG}$.

Also, every CFL is decidable by a LBA. (see ch. 9)

Lemma: Let $M$ be an LBA with $q$ states and $g$ symbols in the tape alphabet. There are exactly $qng^n$ distinct configurations of $M$ for a tape of length $n$.

Proof: Recall that a configuration of $M$ is like a snapshot in the middle of its computation. A configuration consists of the state of the control, position of the head, and contents of the tape. Here, $M$ has $q$ states. The length of its tape is $n$, so the head can be in one of $n$ positions, and $g^n$ possible strings of tape symbols appear on the tape. The product of these three quantities is the total number of different configurations of $M$ with a tape of length $n$. 
Computation Histories

Definition:

Let $M$ be a TM and $w$ an input string. An accepting computation history for $M$ on $w$ is a sequence of configurations, $C_1, C_2, ..., C_r$ where $C_1$ is the start configuration of $M$ on $w$, $C_i$ is an accepting configuration of $M$, and each $C_i$ legally follows from $C_{i-1}$ according to the rules of $M$. A rejecting computation history for $M$ on $w$ is defined similarly, except that $C_i$ is a rejecting configuration.

Computation histories are finite. (They only exist for halting computations.)

Deterministic machines have at most one history for each input.

What can we say about the bounds of the computation histories for any LBA?
Theorem - $A_{LBA}$ is decidable. Finally!

$A_{LBA} = \{ \langle M, w \rangle \mid M \text{ is an LBA that accepts string } w \}$

**PROOF:** $L = \{ \text{On input } \langle M, w \rangle, \text{ where } M \text{ is an LBA and } w \text{ is a string:} \}$

1. Simulate $M$ on $w$ for $qng^n$ steps or until it halts.
2. If $M$ has halted, accept if it has accepted and reject if it has rejected. If it has not halted, reject.”

If $M$ on $w$ has not halted within $qng^n$ steps, it must be repeating a configuration according to Lemma 5.8 and therefore looping. That is why our algorithm rejects in this instance.

Fundamental difference between general TMs and LBAs:

- $A_{LBA}$ is decidable.
- $A_{TM}$ is undecidable.

Why?
Theorem - $E_{\text{LBA}}$ is undecidable

$E_{\text{LBA}} = \{\langle M \rangle \mid M \text{ is an LBA where } L(M) = \emptyset\}$

Proof by reduction from $A_{\text{TM}}$, using Computation Histories.

For a given TM $M$ and input $w$, we will construct a LBA $B$. We will show that if $E_{\text{LBA}}$ is decidable, then $A_{\text{TM}}$ is decidable.

$B$ is a LBA whose language comprises all accepting computation histories for $M$ on $w$.

The construction of $B$ is as follows.

When $B$ receives an input $x$, $B$ is supposed to accept if $x$ is an accepting computation history for $M$ on $w$. First, $B$ breaks up $x$ according to the delimiter into strings $C_1, C_2, ..., C_l$. Then $B$ determines whether the $C_i$'s satisfy the three conditions of an accepting computation history.

1. $C_1$ is the start configuration for $M$ on $w$.
2. Each $C_{i+1}$ legally follows from $C_i$.
3. $C_l$ is an accepting configuration for $M$.

If all conditions are satisfied, $B$ accepts.

By inverting the decider’s answer, we obtain the answer to whether $M$ accepts $w$. 
Theorem - $E_{LBA}$ is undecidable (cont.)

Proof:

Suppose TM $R$ decides $E_{LBA}$. Construct TM $S$ to decide $A_{TM}$ as follows.

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

1. Construct LBA $B$ from $M$ and $w$ as described in the proof idea.
2. Run $R$ on input $\langle B \rangle$.
3. If $R$ rejects, accept; if $R$ accepts, reject.”

If $R$ accepts $\langle B \rangle$, then $L(B) = \emptyset$. Thus, $M$ has no accepting computation history on $w$ and $M$ doesn’t accept $w$. Consequently, $S$ rejects $\langle M, w \rangle$.

Similarly, if $R$ rejects $\langle B \rangle$, the language of $B$ is nonempty. The only string that $B$ can accept is an accepting computation history for $M$ on $w$. Thus, $M$ must accept $w$. Consequently, $S$ accepts $\langle M, w \rangle$.

Because $S$ will either reject or accept, $A_{TM}$ is shown to be decidable for $\langle M, w \rangle$, which is a contradiction.

Therefore, $E_{LBA}$ must be undecidable.
Recap

Reducibility converts one problem into another.

Reduction is the primary method to prove that a problem is computationally unsolvable.

Common: Proof by contradiction

Undecidable Languages:
- $\text{HALT}_{TM}$
- $E_{TM}$
- $\text{REGULAR}_{TM}$
- $\text{EQ}_{TM}$
- $E_{LBA}$

Decidable Language:
- $A_{LBA}$