Def: A language A mapping-reduces to a language B if there is a computable function $f: \Sigma^* \rightarrow \Sigma^*$ such that for all $x$:

$$\text{dom}(f) = \Sigma^* \quad \text{and} \quad x \in A \iff f(x) \in B.$$ 

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

1. If $B$ is decidable, then $A$ is decidable.
2. If $B$ is Turing-enumerable, then $A$ is Turing-enumerable.
3. If $B$ is co-Turing-enumerable, then $A$ is Turing-enumerable.

Proof: (1) We can take a total TM $M_B$ s.t. $L(M_B) = B$ and a total TM $T$ that computes $f(x)$. Construct $M_A$:

We will build a TM $M_A$ s.t. $L(M_A) = A$ and $A$ is total. $M_A$ is a composition of two total routines so it is total. $M_A$ accepts $x$ if $M_B$ accepts $y$ (which $= f(x)$) accepts $y \in B \iff x \in A$ by "$A \leq_m B$ via $f$".

$L(M_A) = A$

For (2), re-do diagram with $M_B$ not a solid box.
Take a TM \( M' \) s.t. \( L(M') = B \) but since we only know \( B \in \text{RE} \) we \( M' \):

\[ \downarrow \text{input} x \]

\[ \text{Do } y = T(x) \]

\[ \text{If } y \text{ and } \text{when all} \]

\[ M_B \]

\[ \vdash \text{accept} \]

\[ \text{by code} \]

\[ \text{by size} \]

For all \( x \), \( M_a \) accepts \( x \) if \( M_B \) accepts \( y \)

\[ \vdash x \in A \]

\[ y = f(x) = T(x) \]

\[ \text{by size} \]

\[ \text{by reduction} \]

\[ \vdash L(M_a) = A \]

\[ \vdash \text{C} \]

Note:

\[ \forall x : x \in A \Leftrightarrow f(x) \in B \]

\[ \Rightarrow \forall x : x \in A \Leftrightarrow f(x) \in B \]

\[ \Rightarrow A \leq_m B \]

So \( \text{B} \in \text{CORE} \land A \leq_m B \Rightarrow \text{CORE} \land A \leq_m B \Rightarrow \text{CORE} \Rightarrow A \in \text{CORE} \).

Contraposible: Suppose \( A \leq_m B \) Then:

- (a') If \( A \) is undecidable, then \( B \) is undecidable.
- (b') If \( A \) is not Turing recognizable, then neither is \( B \).
- (c') If \( A \notin \text{CORE} \) then \( B \notin \text{CORE} \).

Example reduction:

\( A'' = K \)

\( B'' = \text{Atm} \)

\( K \leq_m \text{Atm} \) via \( f(x) = \langle x, x \rangle \)

\( x \in K \Rightarrow x \) is a TM that \( \text{accepts} \langle x, x \rangle \in \text{Atm} \)

The function \( f(x) = \langle x, x \rangle \) is easily computable.

Define \( \text{NE} = \text{INST} = \text{A TM} \)

Ques: Is \( L(M) \neq \emptyset \)?

Ques: Is \( L(M) = \emptyset \)?
Theorem: \( A_{TM} \leq_m N_E_{TM} \), so \( N_E_{TM} \) (and \( E_{TM} \)) are undecidable.

**Domain:** \( \langle M, w \rangle \)
- **Range:** TMs. Hence our goal is to build \( M' = f(M, w) \) by a computable \( f \), such that:
  \[ \langle M, w \rangle \in A_{TM} \iff \langle M' \rangle \in N_E_{TM} \]
- \( A = A_{TM} \) \( B = N_E_{TM} \)
- \( w \in \Sigma^* \)
- \( A' = A_{TM} \) \( B' = N_E_{TM} \)

Then for all instances \( \langle M, w \rangle \) of the \( A_{TM} \) problem,
- \( A_{TM} \) accepts \( w \) \( \iff \) for all \( z \), \( M' \) accepts \( z \) \( \iff \) \( L(M') = \Sigma^* \)
- \( M \) does not accept \( w \) \( \iff \) for all \( z \), \( M' \) does not accept \( z \) \( \iff \) \( L(M') \neq \emptyset \)
- \( \langle M, w \rangle \in A_{TM} \) \( \iff \) \( \langle M' \rangle \in N_E_{TM} \)

And the code \( \langle M' \rangle = f(M, w) \) is computable "by hand".

So \( A_{TM} \leq_m N_E_{TM} \) via \( f \) because \( \langle M, w \rangle \in A_{TM} \iff \langle M' \rangle \in N_E_{TM} \)

**Added:**
This reduction embodies an important idea: The "All-Or-Nothing Switch".
- \( A_{TM} \) accepts \( w \) \( \iff \) \( L(M') = \Sigma^* \). So you have \( \langle M, w \rangle \in A_{TM} \iff \langle M' \rangle \in \text{ALL}_{TM} \)
- \( A_{TM} \) does not accept \( w \) \( \iff \) \( L(M') = \emptyset \). And also \( \langle M, w \rangle \in A_{TM} \iff \langle M' \rangle \in N_E_{TM} \).

This has the same reduction function \( f \) reduces \( A_{TM} \) to \( \text{ALL}_{TM} \) and to \( N_E_{TM} \). It follows that \( \text{ALL}_{TM} \) is likewise undecidable, indeed, not co-re. Whereas \( N_E_{TM} \) is c-e.

\( B_{TM} \) is not c-e. either. The way to show that is to show \( A_{TM} \leq_m \text{ALL}_{TM} \) too. This uses a different idea I call the "Delay Switch." — to come in Tuesday's lecture.