We have an algorithm that converts instances of problem $P_1$ to instances of problem $P_2$ where the answer to $P_2$ can be used to come up with an answer to $P_1$.

- We have reduced $P_1$ to $P_2$
- $P_2$ is at least as hard as $P_1$
  - Because if we have an algorithm for $P_2$, we have an algorithm for $P_1$.

Start with a known hard problem $P_1$ for which no machine exists:
- Like “Does $M$ on $w$ accept?”

Assume there’s a TM, $M_2$, that answers some other question $P_2$:
- Like “Does $M$ accept the empty language?”

Show a way to create a Turing Machine, $M_1$ that decides $P_1$:
- Takes the inputs for $P_1$.
- Converts them into inputs for $M_2$.
- Run $M_2$ on these new inputs
- Use the answer $M_2$ provides to come up with an answer for $M_1$.
- But, since $M_1$ can’t exist, $M_2$ can’t exist either.
Reduction redux

We reduce one language to another language
- Not one machine to another machine

When we reduce A to B:
- We know we can’t decide A
- We’re showing we can’t decide B

When we reduce A to B
- Assume we have a decider for B, M
- Create a machine M’ that takes input for A
- M’ gets to call the decider for B
- M’ must decide A
- Since A is undecidable, no decider for B can exist

Reducing A_TM to halting problem

Halting problem:
- Given M on w, does M halt (accept or reject)?
- \( \text{HALT}_\text{TM} = \{(M, w)| M \text{ is a TM and } M \text{ halts on input } w\} \)

Assume there exists machine M_Halt that decides the halting problem
Create M’ = “On input <M, w>, an encoding of TM M and string w:
- Call M_Halt on <M,w>
  - If rejects, reject
  - If accepts, simulate M on w and when it halts, accept or reject appropriately”

M’ decides A_TM={<M, w>| M is a TM and M accepts input w}
- But this is undecidable!
- So, M_Halt doesn’t exist
- Therefore, Halting problem is undecidable
\[ \text{EMPTY}_\text{TM} = \{ <M> \mid M \text{ is a TM and } M \text{ accepts the empty language} \} \]

Is \( \text{EMPTY}_\text{TM} \) decidable?

\[ \text{REGULAR}_\text{TM} = \{ <M> \mid M \text{ is a TM and } M \text{ accepts a regular language} \} \]

Is \( \text{REGULAR}_\text{TM} \) decidable?
\( \text{CF}_{\text{TM}}=\{<M> \mid M \text{ is a TM and } M \text{ accepts a context-free language}\} \)

Is \( \text{CF}_{\text{TM}} \) decidable?

\( \text{EQ}_{\text{TM}}=\{<M_1, M_2> \mid M_1, M_2 \text{ are TM and } M_1, M_2 \text{ accept the same language}\} \)

Is \( \text{EQ}_{\text{TM}} \) decidable?
Linear Bounded Automaton

Turing machines that can’t read/write outside the input

- $A_{\text{LBA}} = \{<M, w>| M \text{ is an LBA such that } M(w) \text{ accepts}\}$
- $A_{\text{LBA}}$ is decidable

Linear-Bounded Automaton

$E_{\text{LBA}}$ is undecidable

- If decidable, here’s algorithm for $A_{\text{TM}}$
Does a CFG $G$ generate all strings?

$\text{ALLCFG} = \{ G \mid G \text{ is a CFG and } L(G) = \Sigma^* \}$

- Given $M$ and $w$, construct $G$ such that
  - $G$ accepts all strings except accepting computation history for $M$ on $w$