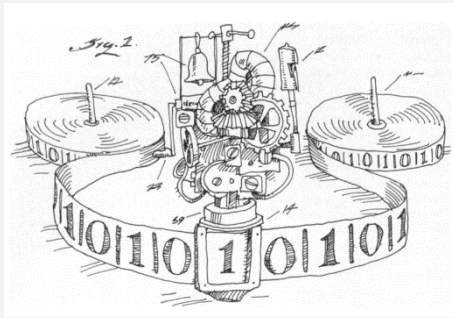
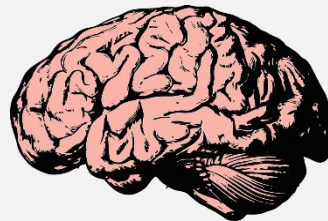
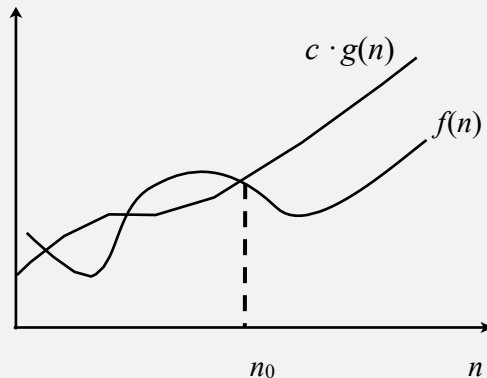


COMP 3602

Intro to Theory of Computation

Lecture 11



Logistics

- First oral exam: as soon as we finish the next topic
 - Which days of the week and times are OK? Mon Mar 2nd?
- Presentation proposals? To say which topic you picked. Mar 2nd?
 - About 30 (50?) minutes intro to a topic in complexity (or computability)
 - In the reading group on in class?
 - Like Jordan's intro to Kolmogorov complexity, but shorter
 - Quantum computing, proof complexity, fine-grained complexity, circuit complexity, communication complexity, pseudorandomness, etc...

Questions/comments?



Computable functions

- *Definition:* A function $f: \Sigma^* \rightarrow \Sigma^*$ is **computable** if there exists a Turing machine M which halts on every input, and when M halts on a w , its tape contains only $f(w)$ and its head points to the first symbol of $f(w)$.
 - That is, it halts in a configuration of the form $q_a f(w)$, ignoring blanks.
 - The definition does not specify whether it has to be in q_a , but WLOG it is.
- Examples:
 - in a practice problem you showed that the function $f: \{0,1\}^* \rightarrow \{0,1\}^*$ defined as $f(x) = x - 2$ is computable
 - All arithmetic operations are computable.
 - A function $f: \{0,1\}^* \rightarrow \{0,1\}^*$ which takes $\langle M \rangle$ and outputs $\langle M' \rangle$ with accept/reject states switched is computable
 - For $x \neq \langle M \rangle$, define $f(x) = x$.
 - A function $f: \{0,1\}^* \rightarrow \{0,1\}^*$ which takes $\langle M, w \rangle$ and outputs 1 if M accepts w , 0 if M rejects w and blank if M loops on w is **not** computable.



Mapping reduction

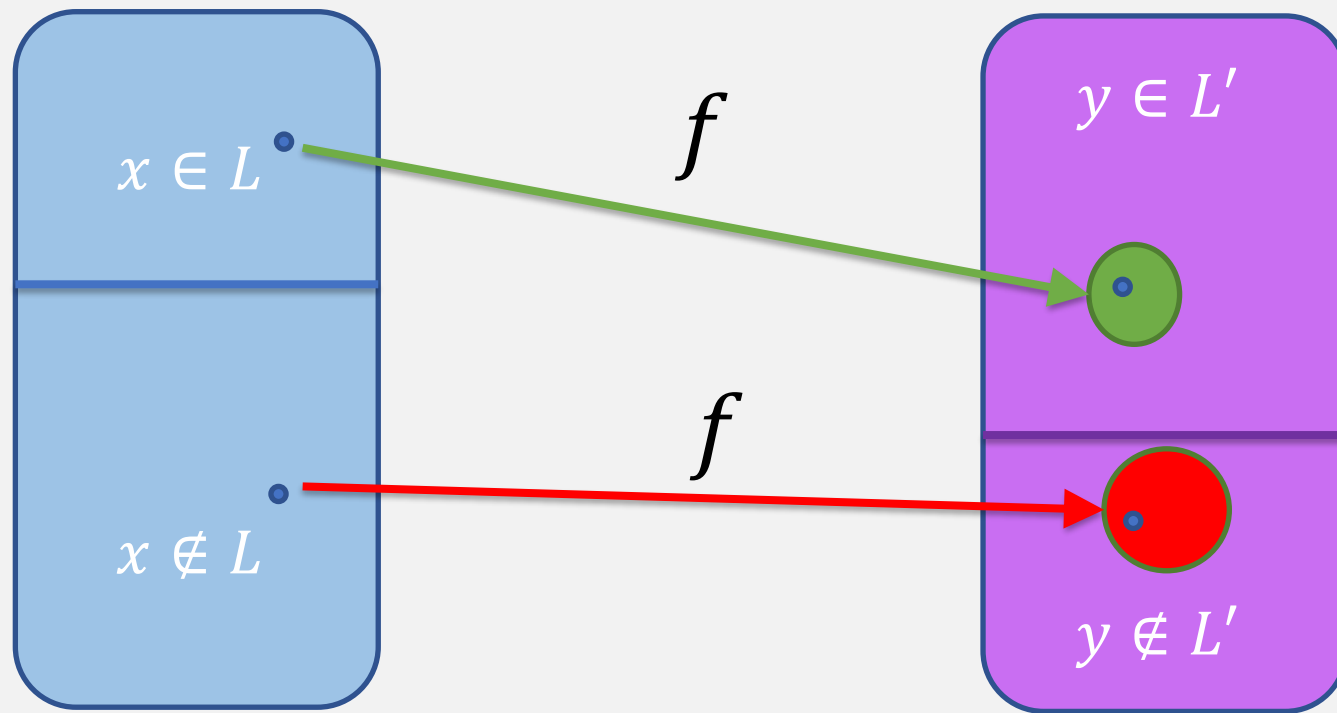
- *Definition:* Let L and L' be languages over Σ .
 - Then L is **mapping-reducible** to L' , written $L \leq_m L'$ iff there exists a computable function $f: \Sigma^* \rightarrow \Sigma^*$ such that $\forall x \in \Sigma^*$
$$x \in L \iff f(x) \in L'$$
The function f is called the **reduction** from L to L' .
- Such f “disguises” instances of L as instances of L'
- Our proof that $HALT_{TM}$ is undecidable reduced A_{TM} to $HALT_{TM}$



Mapping reduction

- Examples of reductions
 - Reduce checking if a graph is not bipartite to existence of an odd cycle
- Note: here we are just considering **mapping** reductions
 - **Not** where you run an algorithm for problem B, potentially multiple times and/or with extra computation on its output, to solve problem A
 - We might look at the latter kind (called Turing reductions) later in the course.

Mapping reduction: $L \leq_m L'$



- If f is a reduction from L to L' , then it also a reduction $\bar{L} \leq_m \bar{L}'$
 - So if $L \leq_m L'$, then $\bar{L} \leq_m \bar{L}'$
- Also, if $L_1 \leq_m L_2$ and $L_2 \leq_m L_3$, then $L_1 \leq_m L_3$

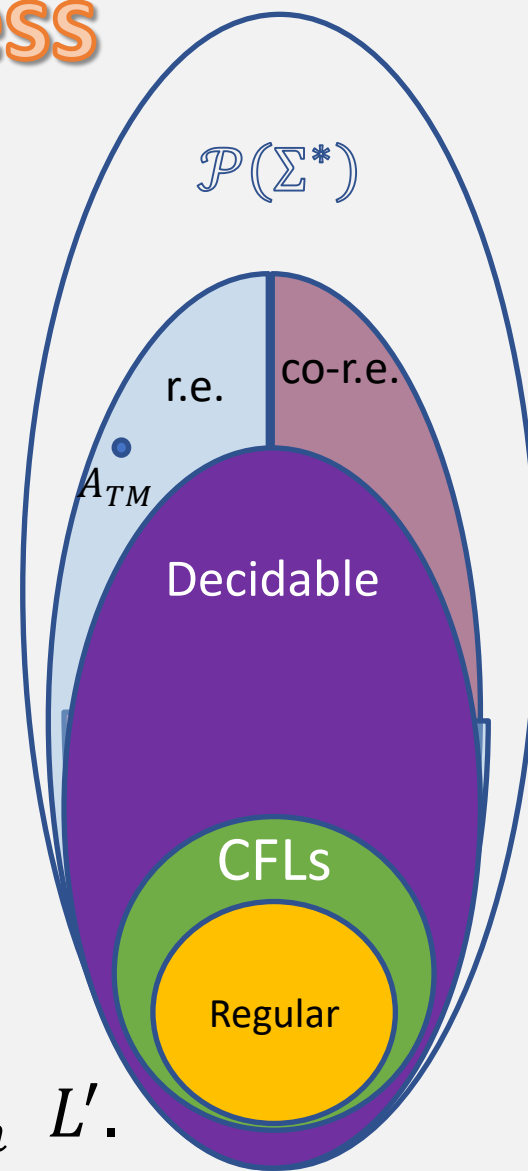


Reducibility for proving hardness

- The “ $\leq$ ” symbol indicates comparative hardness:
 - When $L \leq_m L'$, that means that L' is at least as hard as L
- Suppose that $L \leq_m L'$.
 - If L is **undecidable**, then L' is
 - If L is **not Turing-recognizable**, then L' is

Reduction is the main technique for proving hardness
For easiness, just give an algorithm for the original problem.

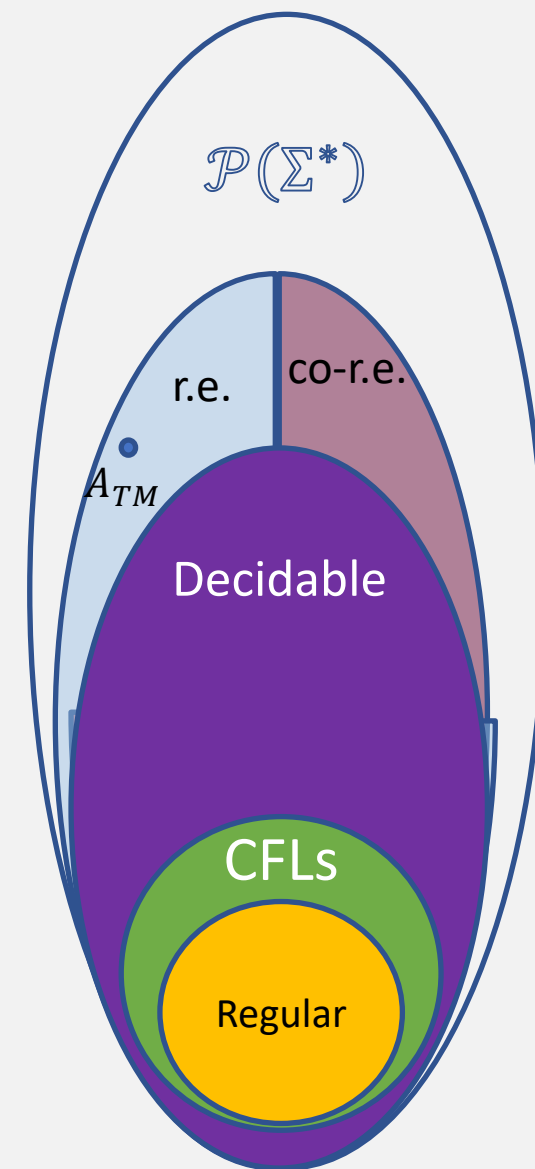
- Can prove that L is easy by proving that L' is easy and that $L \leq_m L'$.
 - This is done a lot in algorithm design





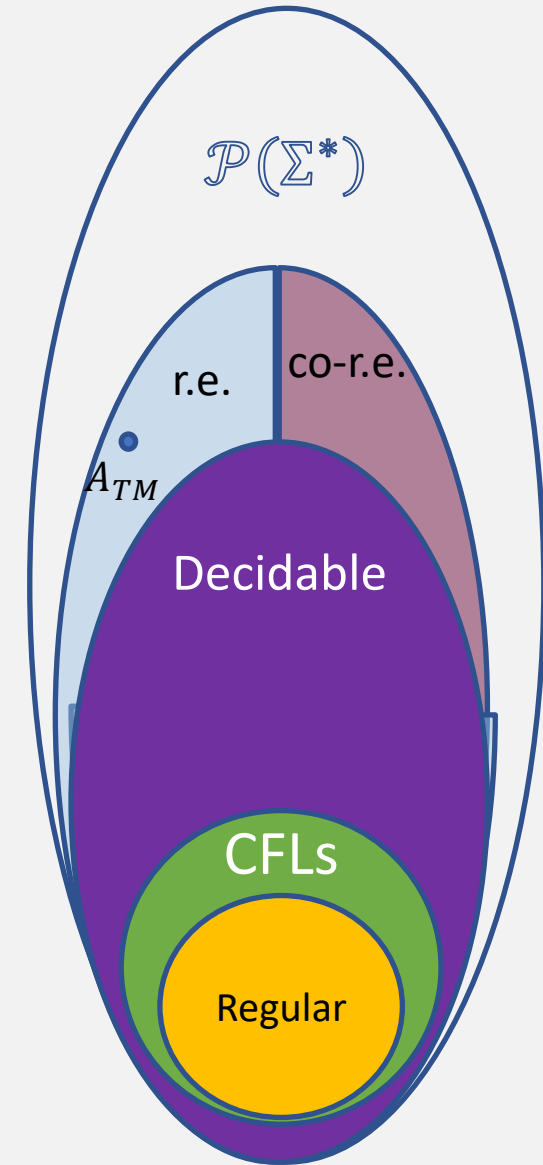
Template

- To show that a language L is undecidable (and not co-r.e.), prove $A_{TM} \leq_m L$
 - To show that L is r.e., construct M_L
- To show that a language L is undecidable and not r.e., prove $\overline{A_{TM}} \leq_m L$
 - To show that L is co-r.e., construct $M_{\bar{L}}$
- To show that L is neither r.e. nor co-r.e., prove both $A_{TM} \leq_m L$ and $\overline{A_{TM}} \leq_m L$
 - Or reduce another language outside of both r.e. and co-r.e. to L



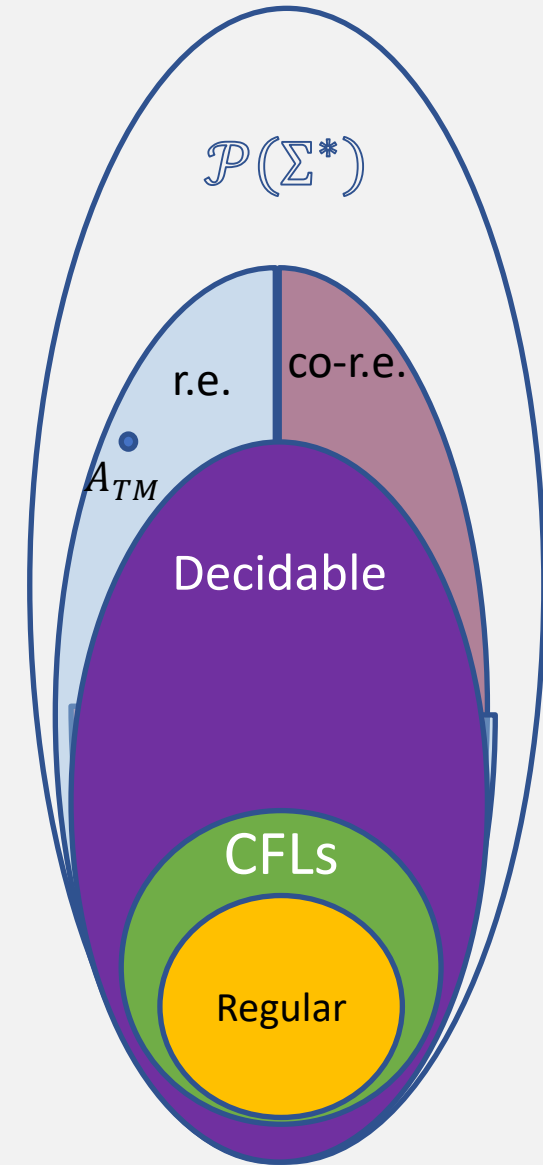
R.e. vs co-r.e.

- Is there a reduction $A_{TM} \leq_m \overline{A_{TM}}$?
 - Or, equivalently, $\overline{A_{TM}} \leq_m A_{TM}$?
 - No!
 - If $\overline{A_{TM}} \leq_m A_{TM}$, then since A_{TM} is r.e., then $\overline{A_{TM}}$ would also be r.e.
 - But if both were r.e., that would make A_{TM} decidable!
 - Contradiction.
- There can **never** be a reduction between two undecidable languages one of which is r.e. and another co-r.e.
 - In either direction.
 - So to prove undecidability of some languages, namely co-r.e. ones, you cannot reduce A_{TM} to them, and have to use $\overline{A_{TM}}$ instead



Reduction examples: HaltB

- $HaltB = \{\langle M \rangle \mid M \text{ halts (stops) on input } \lambda\}$
 - That is, the language of machines that stop when run on blank input.
- $HaltB$ is Turing-recognizable
 - Exercise! Hint: build M_{HaltB}
- $HaltB$ is undecidable.
 - Let's show that $A_{TM} \leq_m HaltB$
 - That is, show how to disguise instances of A_{TM} as instances of $HaltB$



- $HaltB = \{\langle M \rangle \mid M \text{ halts (stops) on input } \lambda\}$
- $HaltB$ is undecidable.

- Let's show that $A_{TM} \leq_m HaltB$
- Construct a computable function f

$$f(x) = \begin{cases} \langle M' \rangle & \text{if } x = \langle M, w \rangle \\ 0 & \text{otherwise} \end{cases}$$

- assuming “0” is not an encoding of any TM

- Where M' is:



$M' =$ “ On input x

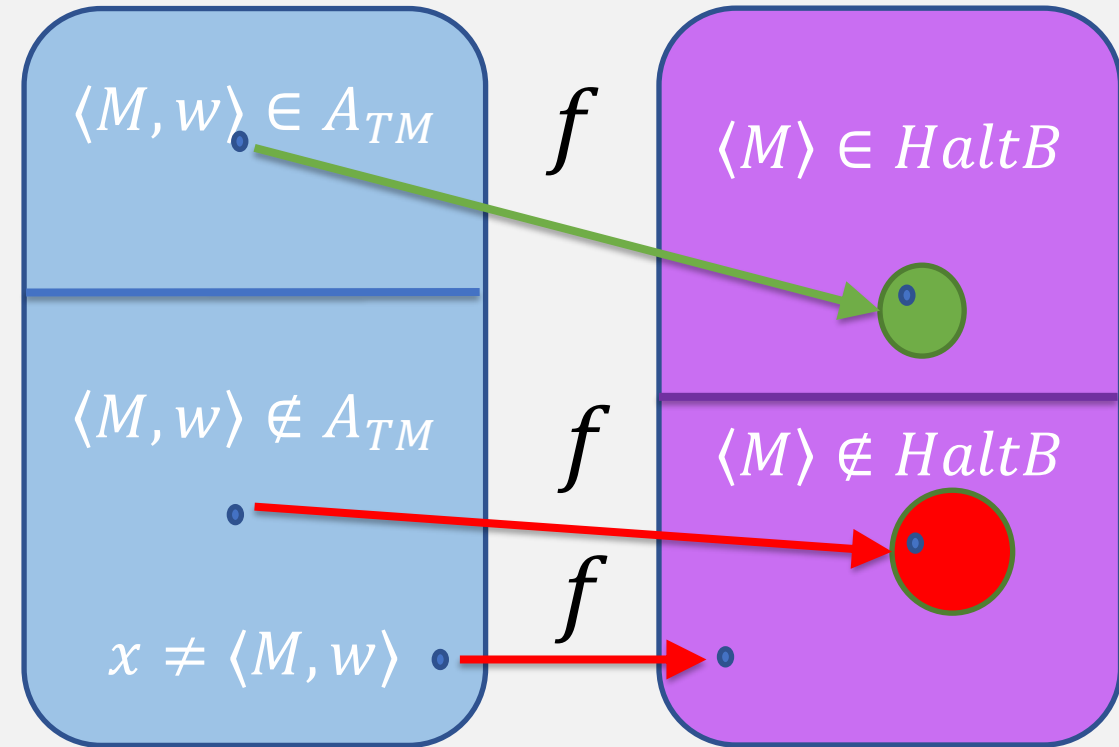
If $x \neq \lambda$, **reject**.

Simulate M on input w

If M ever enters accept state, **accept**.

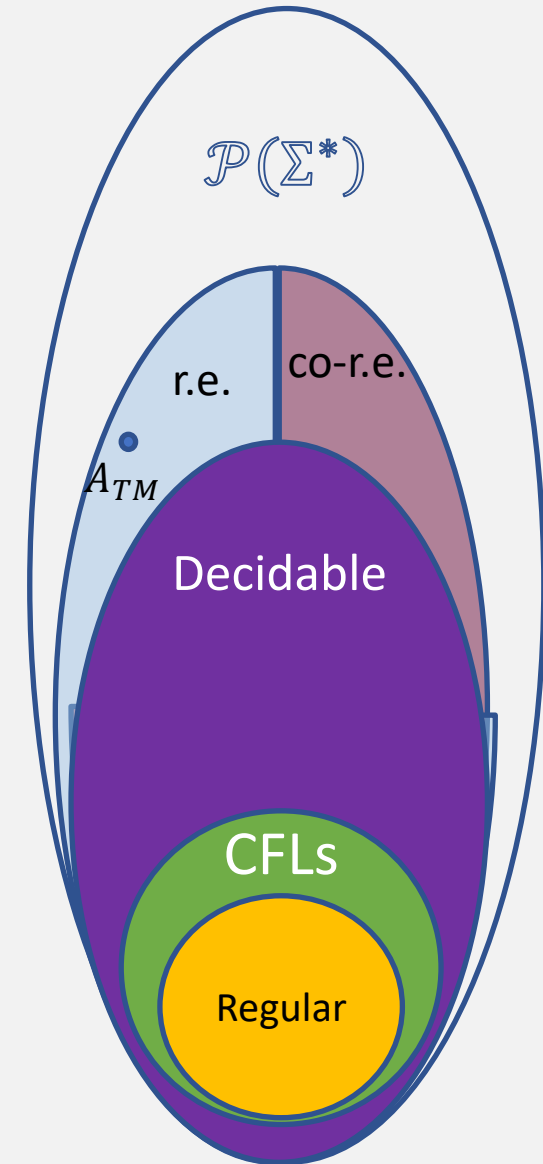
If M ever enters reject state, **loop**. “

- Note: M' depends on both M and w , as w is hardcoded in M' .



Reduction examples: E_{TM}

- $E_{TM} = \{\langle M \rangle \mid \mathcal{L}(M) = \emptyset\}$
 - Set of machines M that do not accept anything
- *Claim:* E_{TM} is co-Turing-recognizable and undecidable.
$$\overline{E_{TM}} = \{x \mid x \neq \langle M \rangle\} \cup \{\langle M \rangle \mid M \text{ accepts some string } w\}$$
- E_{TM} is co-Turing-recognizable
 - Show that $\overline{E_{TM}}$ is Turing-recognizable
- E_{TM} is undecidable.
 - Show that $\overline{A_{TM}} \leq_m E_{TM}$, or, equivalently, $A_{TM} \leq_m \overline{E_{TM}}$
 - Remember that **there cannot be a reduction** $A_{TM} \leq_m E_{TM}$!



- $E_{TM} = \{\langle M \rangle \mid M \text{ does not accept anything, that is, } \mathcal{L}(M) = \emptyset\}$
- $\overline{E_{TM}} = \{x \mid x \neq \langle M \rangle\} \cup \{\langle M \rangle \mid M \text{ accepts some string } w\}$
- E_{TM} is co-Turing-recognizable
 - Equivalently, $\overline{E_{TM}}$ is Turing-recognizable, and this is what we will show

Idea: $M_{\overline{E_{TM}}}$ will work in stages, on each stage trying a few more strings, and doing one more step on the previously considered ones.

$M_{\overline{E_{TM}}}$ = “ On input x
 If $x \neq \langle M \rangle$, **accept**
 For $i = 0$ to ∞
 Simulate M on all strings of length at most i for i steps each
 If M enters accept state on any of these strings, **accept**.

- Note: could also take strings in, say, string order (so that $w_0 = \lambda, w_1 = 0, w_2 = 1, w_3 = 00, \dots$) and simulate M on inputs $w_0 \dots w_i$ for i steps each at each stage.
 - As long as each stage is guaranteed to finish in finite time,
- If there is w which M accepts, say in t steps, then $M_{\overline{E_{TM}}}$ accepts $\langle M \rangle$ during stage $\max(|w|, t)$
- If $\mathcal{L}(M) = \emptyset$, $M_{\overline{E_{TM}}}$ will run forever and never accept.

- $E_{TM} = \{\langle M \rangle \mid M \text{ does not accept anything, that is, } \mathcal{L}(M) = \emptyset\}$
- $\overline{E_{TM}} = \{x \mid x \neq \langle M \rangle\} \cup \{\langle M \rangle \mid M \text{ accepts some string } w\}$

• E_{TM} is undecidable.

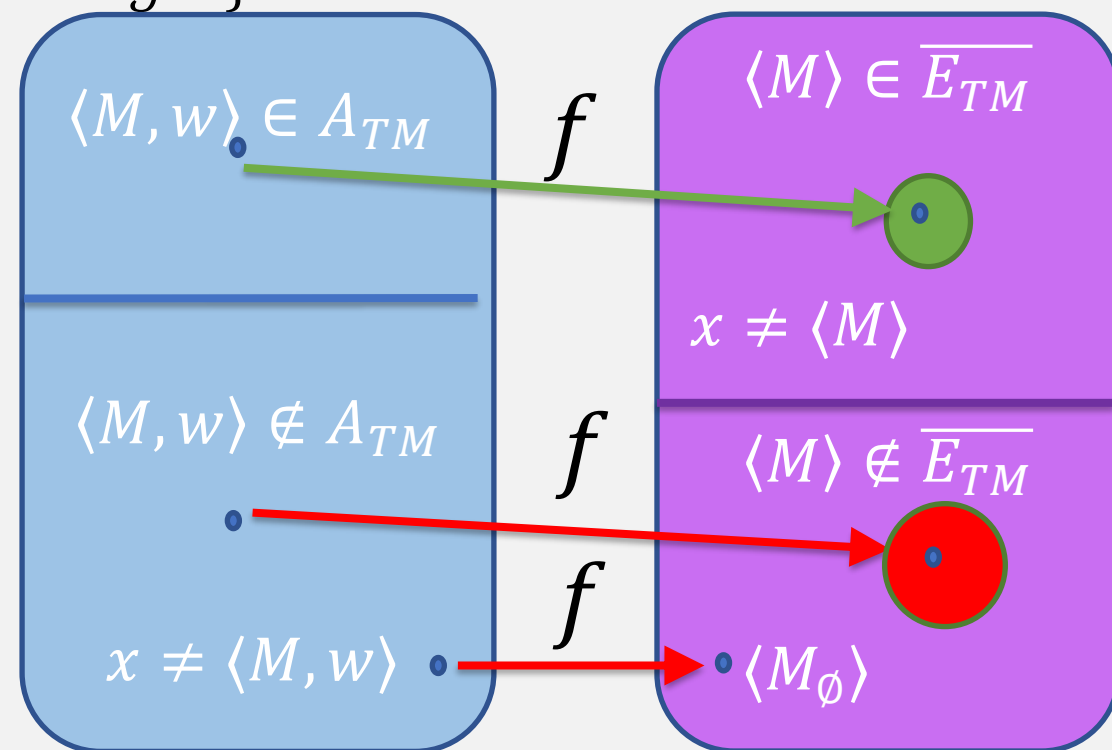
- Let's show that $A_{TM} \leq_m \overline{E_{TM}}$
- Construct a computable function f

$$f(x) = \begin{cases} \langle M' \rangle & \text{if } x = \langle M, w \rangle \\ \langle M_\emptyset \rangle & \text{otherwise} \end{cases}$$

- Where M' is:



M' = “ On input x
 Simulate M on input w
 If M ever enters accept state, *accept*.
 If M ever enters reject state, *reject*. ”



$M_\emptyset$: $\Sigma = \{0,1\}, \Gamma = \{0,1,_ \}$

$Q = \{q_0, q_a, q_r\}$

$\delta: (q_0, s) \rightarrow (q_r, s, R)$ for all $s \in \Gamma$