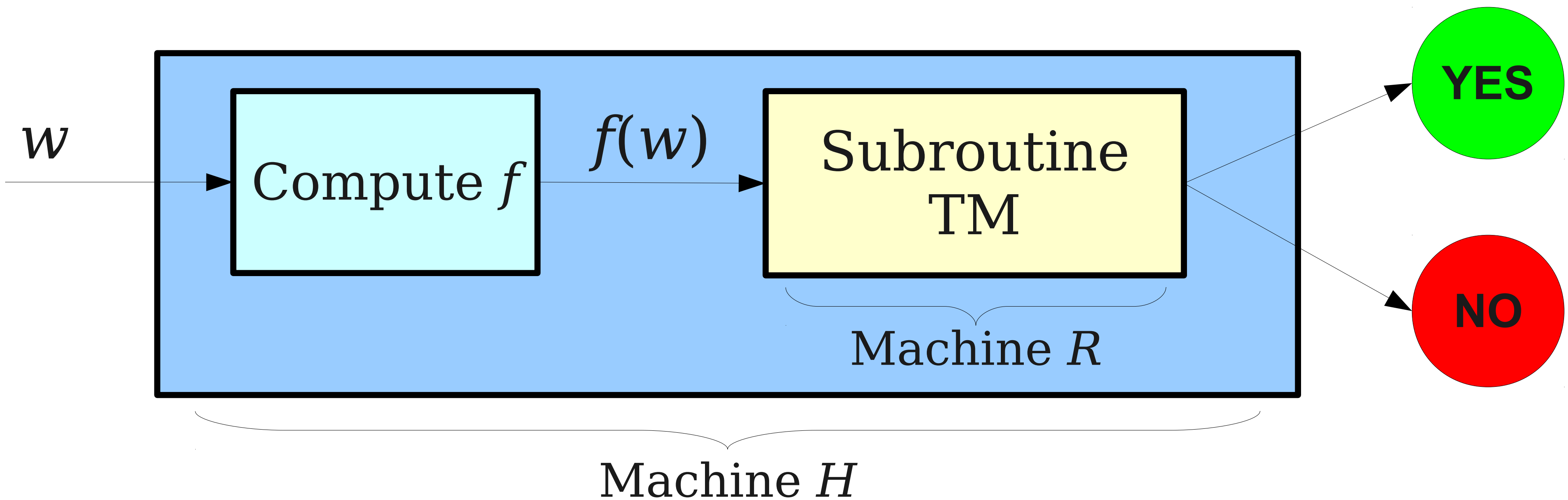
Reducibility

Part II

Problem Set 7
due in the box
up front.

The General Pattern



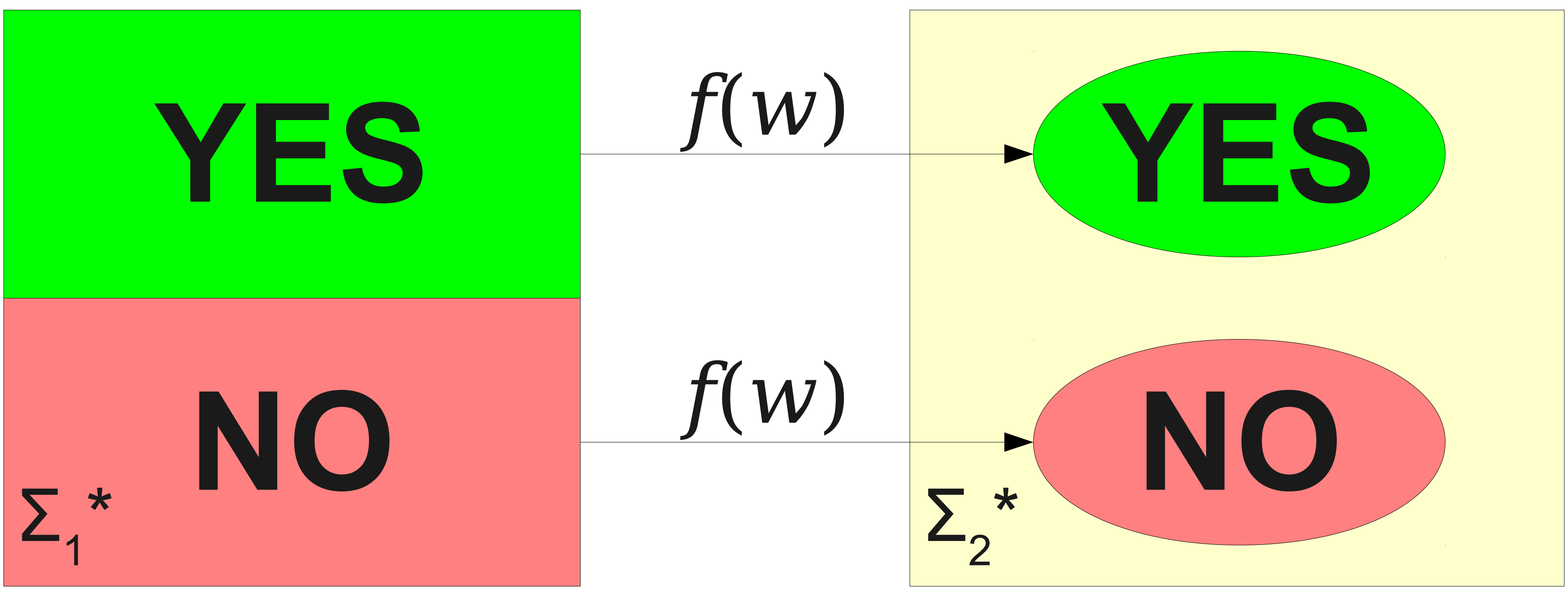
H = "On input w:

- Transform the input w into $f(w)$.
- Run machine R on $f(w)$.
- If R accepts $f(w)$, then H accepts w.
- If R rejects $f(w)$, then H rejects w."

Defining Reductions

- A **reduction** from A to B is a function $f : \Sigma_1^* \to \Sigma_2^*$ such that

For any $w \in \Sigma_1^*$, $w \in A$ iff $f(w) \in B$



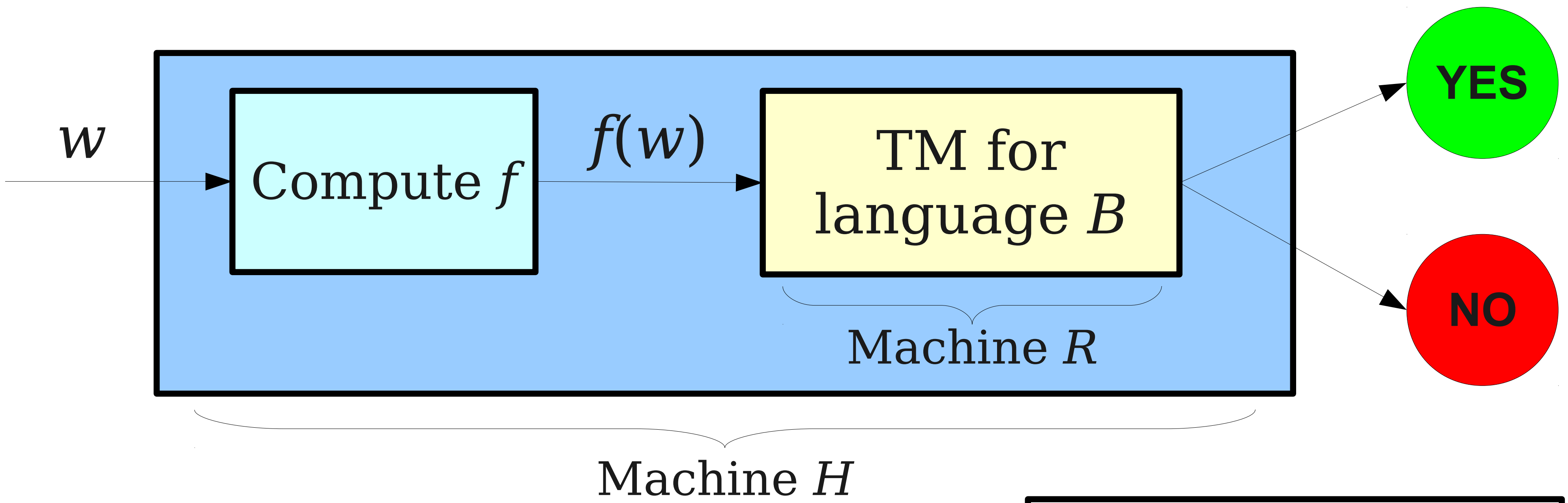
Defining Reductions

- A **reduction** from A to B is a function $f : \Sigma_1^* \to \Sigma_2^*$ such that

$$\textbf{For any } \boldsymbol{w \in \Sigma_1^*, w \in A} \textbf{ iff } \boldsymbol{f(w) \in B}$$

- Every $w \in A$ maps to some $f(w) \in B$.
- Every $w \notin A$ maps to some $f(w) \notin B$.
- f does not have to be injective or surjective.

$$w \in A \quad \text{iff} \quad f(w) \in B$$



H = "On input w:

- Transform the input w into $f(w)$.
- Run machine R on $f(w)$.
- If R accepts $f(w)$, then H accepts w.
- If R rejects $f(w)$, then H rejects w."

H* accepts *w

iff

***R* accepts *f*(*w*)**

iff

$f(w) \in B$

iff

$w \in A$

Mapping Reductions

- A function $f : \Sigma_1^* \to \Sigma_2^*$ is called a **mapping reduction** from A to B iff
  - For any $w \in \Sigma_1^*$, $w \in A$ iff $f(w) \in B$.
  - f is a computable function.
- Intuitively, a mapping reduction from A to B says that a computer can transform any instance of A into an instance of B such that the answer to B is the answer to A.

Mapping Reducibility

- If there is a mapping reduction from language *A* to language *B*, we say that language *A* is **mapping reducible** to language *B*.
- Notation: $\boldsymbol{A \leq_M B}$ iff language *A* is mapping reducible to language *B*.
- Note that we reduce ***languages***, not ***machines***.

Why Mapping Reducibility Matters

- **Theorem**: If $B \in \mathbf{R}$ and $A \leq_{\mathrm{M}} B$, then $A \in \mathbf{R}$.
- **Theorem**: If $B \in \mathbf{RE}$ and $A \leq_{\mathrm{M}} B$, then $A \in \mathbf{RE}$.
- **Theorem**: If $B \in \text{co-}\mathbf{RE}$ and $A \leq_{\mathrm{M}} B$, then $A \in \text{co-}\mathbf{RE}$.
- *Intuitively:* $A \leq_{\mathrm{M}} B$ means "A is not harder than B."

Why Mapping Reducibility Matters

- **Theorem**: If $A \notin \mathbf{R}$ and $A \leq_{\mathrm{M}} B$, then $B \notin \mathbf{R}$.
- **Theorem**: If $A \notin \mathbf{RE}$ and $A \leq_{\mathrm{M}} B$, then $B \notin \mathbf{RE}$.
- **Theorem**: If $A \notin \text{co-}\mathbf{RE}$ and $A \leq_{\mathrm{M}} B$, then $B \notin \text{co-}\mathbf{RE}$.
- *Intuitively:* $A \leq_{\mathrm{M}} B$ means "B is at at least as hard as A."

Why Mapping Reducibility Matters

If this one is "easy" (R, RE, co-RE)...

$$A \leq_{\mathrm{M}} B$$

... then this one is "easy" (R, RE, co-RE) too.

Why Mapping Reducibility Matters

If this one is "hard" (not **R**, not **RE**, or not co-**RE**)…

$$A \leq_{\mathrm{M}} B$$

… then this one is "hard" (not **R**, not **RE**, or not co-**RE**) too.

Using Mapping Reductions

Revisiting our Proofs

- Consider the language

$$L = \{ \langle M \rangle \mid M \text{ is a TM and } M \text{ accepts } \varepsilon \}$$

- We have already proven that this language is in **RE** by building a TM for it.
- Let's repeat this proof using mapping reductions.
- Specifically, we will prove

$$L \leq_M A_{TM}$$

$L = \{ \langle M \rangle \mid M \text{ is a TM and } M \text{ accepts } \varepsilon \}$

- To prove $L \leq_M A_{TM}$, we will need to find a computable function f such that

$$\boldsymbol{\langle M \rangle \in L \quad \textbf{iff} \quad f(\langle M \rangle) \in A_{TM}}$$

- Since A_{TM} is a language of TM/string pairs, let's assume $f(\langle M \rangle) = \langle N, w \rangle$ for some TM N and string w (which we'll pick later):

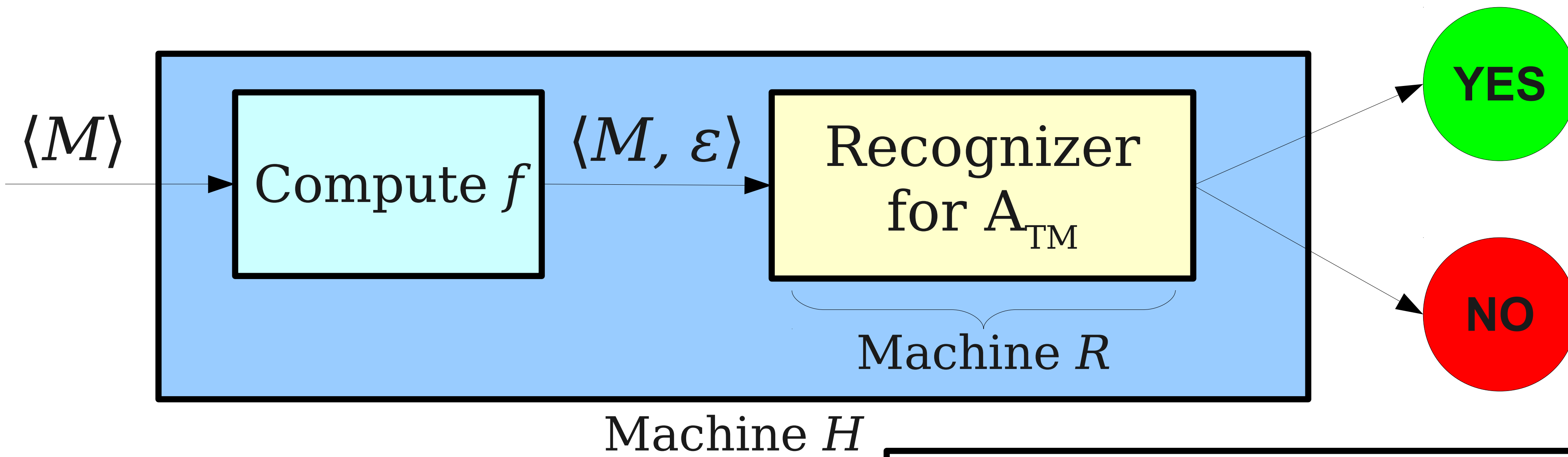
$$\boldsymbol{\langle M \rangle \in L \quad \textbf{iff} \quad \langle N, w \rangle \in A_{TM}}$$

- Substituting definitions:

$$\boldsymbol{M \textbf{ accepts } \varepsilon \quad \textbf{iff} \quad N \textbf{ accepts } w}$$

- Choose $\boldsymbol{N = M}$, $\boldsymbol{w = \varepsilon}$. So $\boldsymbol{f(\langle M \rangle) = \langle M, \varepsilon \rangle}$.

One Interpretation of the Reduction



H = "On input $\langle M \rangle$:

- Run machine R on $\langle M, \varepsilon \rangle$.
- If R accepts $\langle M, \varepsilon \rangle$, then H accepts w.
- If R rejects $\langle M, \varepsilon \rangle$, then H rejects w."

H accepts $\langle M \rangle$

iff

R accepts $\langle M, \varepsilon \rangle$

iff

M accepts ε

iff

$\langle M \rangle \in L$

$$L = \{ \langle M \rangle \mid M \text{ is a TM that accepts } \varepsilon \}$$

Theorem: $L \in$ **RE**.

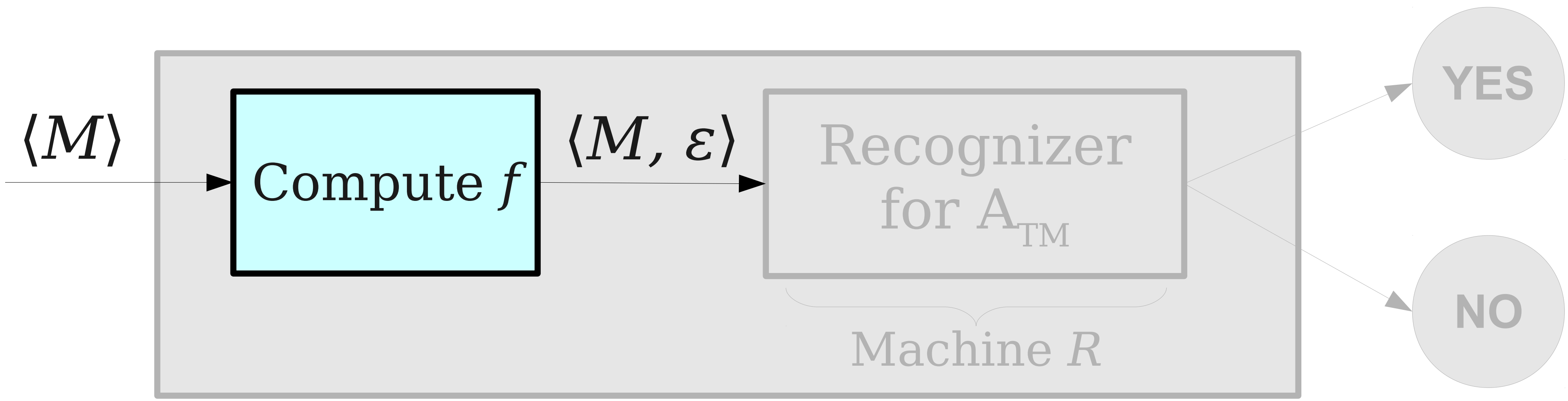
Proof: We will prove that $L \le_M A_{TM}$. Since $A_{TM} \in$ **RE**, this proves $L \in$ **RE** as well.

To prove this, we will give a mapping reduction from L to A_{TM}. For any TM M, let $f(\langle M \rangle) = \langle M, \varepsilon \rangle$. This function can be computed by a Turing machine.

Now, we will prove that f is a mapping reduction by proving for all TMs M that $\langle M \rangle \in L$ iff $\langle M, \varepsilon \rangle \in A_{TM}$. To do this, consider any TM M. Note that by the definition of L, we see $\langle M \rangle \in L$ iff M accepts ε. By the definition of A_{TM}, we know that M accepts ε iff $\langle M, \varepsilon \rangle \in A_{TM}$. Combining these statements together, we have that $\langle M \rangle \in L$ iff $\langle M, \varepsilon \rangle \in A_{TM}$.

This means that f is a mapping reduction from L to A_{TM}, so $L \le_M A_{TM}$, as required. ■

What Did We Prove?



Machine H

H = "On input ⟨M⟩:

- Run machine R on ⟨M, ε⟩.
- If R accepts ⟨M, ε⟩, then H accepts w.
- If R rejects ⟨M, ε⟩, then H rejects w."

H accepts ⟨M⟩

iff

R accepts ⟨M, ε⟩

iff

M accepts ε

iff

⟨M⟩ ∈ L

Interpreting Mapping Reductions

- If $A \leq_M B$, there is a known construction to turn a TM for B into a TM for A.
- When doing proofs with mapping reductions, you do *not* need to show the overall construction.
- You just need to prove that
  - f is a computable function, and
  - $w \in A$ iff $f(w) \in B$.

Another Mapping Reduction

L_D and $\overline{A}_{TM}$

- Earlier, we proved $\overline{A}_{TM} \notin \mathbf{RE}$ by proving that

$$\text{If } \overline{A}_{TM} \in \mathbf{RE}\text{, then } L_D \in \mathbf{RE}.$$

- The proof constructed this TM, assuming R was a recognizer for $\overline{A}_{TM}$.

H = "On input $\langle M \rangle$:

- Construct the string $\langle M, \langle M \rangle \rangle$.
- Run R on $\langle M, \langle M \rangle \rangle$.
- If R accepts $\langle M, \langle M \rangle \rangle$, then H accepts $\langle M \rangle$.
- If R rejects $\langle M, \langle M \rangle \rangle$, then H rejects $\langle M \rangle$."

- Let's do another proof using mapping reductions.

$L_D \leq_M \overline{A}_{TM}$

- To prove that $\overline{A}_{TM} \notin \mathbf{RE}$, we will prove

$$\boldsymbol{L_D \leq_M \overline{A}_{TM}}$$

- By our earlier theorem, since $L_D \notin \mathbf{RE}$, we have that $\overline{A}_{TM} \notin \mathbf{RE}$.
- *Intuitively*: $\overline{A}_{TM}$ is "at least as hard" as L_D, and since $L_D \notin \mathbf{RE}$, this means $\overline{A}_{TM} \notin \mathbf{RE}$.

$L_D \leq_M \overline{A}_{TM}$

- Goal: Find a computable function f such that

$$\boldsymbol{\langle M \rangle \in L_D \quad \text{iff} \quad f(\langle M \rangle) \in \overline{A}_{TM}}$$

- Simplifying this using the definition of L_D

$$\boldsymbol{M \text{ does not accept } \langle M \rangle \quad \text{iff} \quad f(\langle M \rangle) \in \overline{A}_{TM}}$$

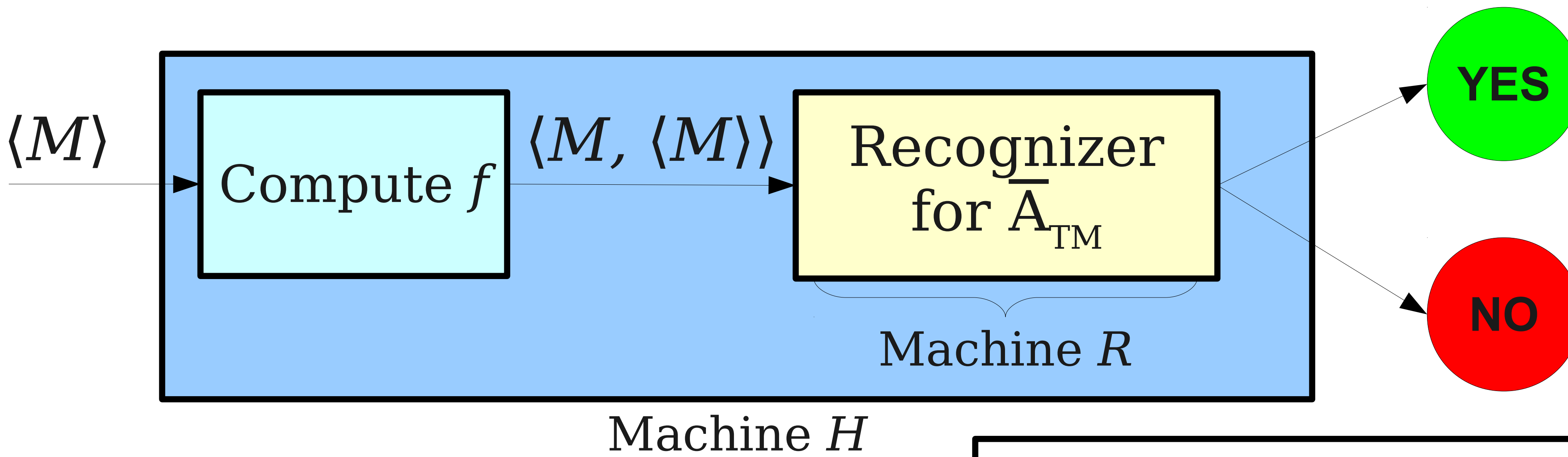
- Let's assume that $f(\langle M \rangle)$ has the form $\langle N, w \rangle$ for some TM N and string w. This means that

$$\boldsymbol{M \text{ does not accept } \langle M \rangle \quad \text{iff} \quad \langle N, w \rangle \in \overline{A}_{TM}}$$

$$\boldsymbol{M \text{ does not accept } \langle M \rangle \quad \text{iff} \quad N \text{ does not accept } w}$$

- If we can choose w and N such that the above is true, we will have our reduction from L_D to $\overline{A}_{TM}$.
- Choose $\boldsymbol{N = M}$ and $\boldsymbol{w = \langle M \rangle}$.

One Interpretation of the Reduction



H = "On input $\langle M \rangle$:

- Run machine R on $\langle M, \langle M \rangle \rangle$.
- If R accepts $\langle M, \langle M \rangle \rangle$, then H accepts w.
- If R rejects $\langle M, \langle M \rangle \rangle$, then H rejects w."

H accepts $\langle M \rangle$

iff

R accepts $\langle M, \langle M \rangle \rangle$

iff

M does not accept $\langle M \rangle$

iff

$\langle M \rangle \in L_D$

Theorem: $\overline{\mathrm{A}}_{\mathrm{TM}} \notin \mathbf{RE}$.

Proof: We will prove that $L_{\mathrm{D}} \leq_{\mathrm{M}} \overline{\mathrm{A}}_{\mathrm{TM}}$. Since $L_{\mathrm{D}} \notin \mathbf{RE}$, this proves that $\overline{\mathrm{A}}_{\mathrm{TM}} \notin \mathbf{RE}$.

To show that $L_{\mathrm{D}} \leq_{\mathrm{M}} \overline{\mathrm{A}}_{\mathrm{TM}}$, we will give a mapping reduction from L_{D} to $\overline{\mathrm{A}}_{\mathrm{TM}}$. For any TM M, let $f(\langle M \rangle) = \langle M, \langle M \rangle \rangle$. This function f is computable.

To prove that f is a mapping reduction from L_{D} to $\overline{\mathrm{A}}_{\mathrm{TM}}$, we will prove for all TMs M that $\langle M \rangle \in L_{\mathrm{D}}$ iff $\langle M, \langle M \rangle \rangle \in \overline{\mathrm{A}}_{\mathrm{TM}}$. By the definition of L_{D}, we know $\langle M \rangle \in L_{\mathrm{D}}$ iff M does not accept $\langle M \rangle$. Similarly, by definition of $\overline{\mathrm{A}}_{\mathrm{TM}}$, we know that M does not accept $\langle M \rangle$ iff $\langle M, \langle M \rangle \rangle \in \overline{\mathrm{A}}_{\mathrm{TM}}$. Combining these statements together, we see $\langle M \rangle \in L_{\mathrm{D}}$ iff $\langle M, \langle M \rangle \rangle \in \overline{\mathrm{A}}_{\mathrm{TM}}$. Thus f is a mapping reduction from L_{D} to $\overline{\mathrm{A}}_{\mathrm{TM}}$, so $L_{\mathrm{D}} \leq \overline{\mathrm{A}}_{\mathrm{TM}}$, as required. ■

The Amplifier Machine

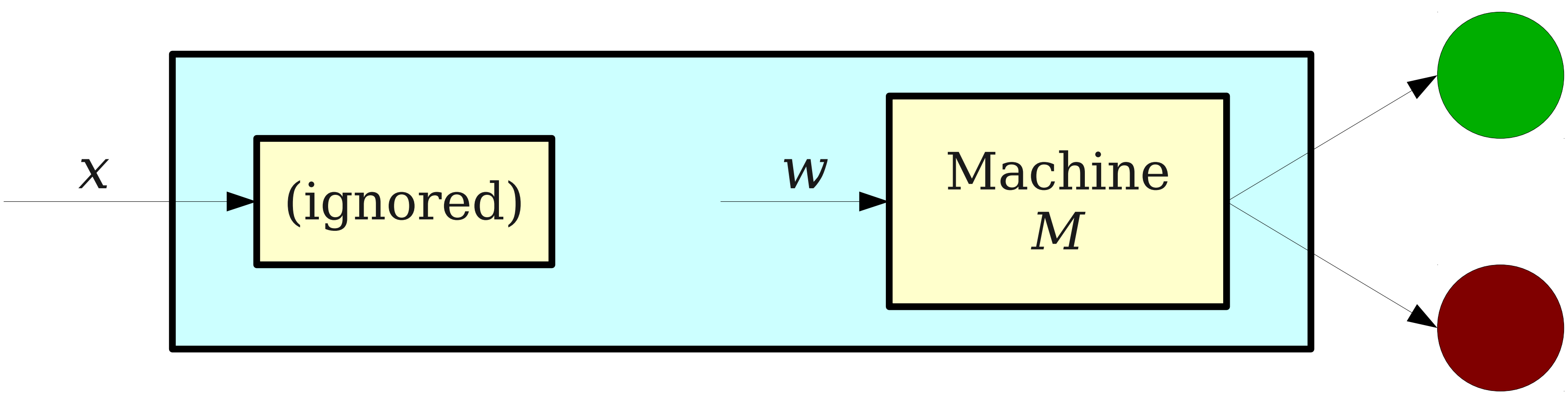
TMs in TMs

- As we've seen, Turing machines can run other Turing machines as subroutines.
- In order to reduce certain problems to one another, it is useful / necessary to embed Turing machines inside of one another.
  - We'll see an example in a second.
- One construction, in particular, is useful for reductions like these.

The Amplifier Machine

For any TM M and string w, let Amp(M, w) be this TM:

Amp(M, w) = "On input x:
Ignore x.
Run M on w.
If M accepts w, then Amp(M, w) accepts x.
If M rejects w, then Amp(M, w) rejects x."



The Amplifier Machine

For any TM M and string w, let $\mathrm{Amp}(M, w)$ be this TM:

$\mathrm{Amp}(M, w)$ = "On input x:
Ignore x.
Run M on w.
If M accepts w, then $\mathrm{Amp}(M, w)$ accepts x.
If M rejects w, then $\mathrm{Amp}(M, w)$ rejects x."

Theorem 1: If M accepts w, then $\mathscr{L}(\mathrm{Amp}(M, w)) = \Sigma^*$. If M does not accept w, then $\mathscr{L}(\mathrm{Amp}(M, w)) = \emptyset$.

Corollary 1: M accepts w iff $\mathscr{L}(\mathrm{Amp}(M, w)) = \Sigma^*$

Corollary 2: M does not accept w iff $\mathscr{L}(\mathrm{Amp}(M, w)) = \emptyset$.

Theorem 2: The function $f(\langle M, w\rangle) = \langle \mathrm{Amp}(M, w)\rangle$ is computable.

For any TM M and string w, let Amp(M, w) be the following TM:

Amp(M, w) = "On input x:
Ignore x.
Run M on w.
If M accepts w, then Amp(M, w) accepts x.
If M rejects w, then Amp(M, w) rejects x."

Theorem: If M accepts w, then $\mathscr{L}(\text{Amp}(M, w)) = \Sigma^*$. If M does not accept w, then $\mathscr{L}(\text{Amp}(M, w)) = \emptyset$.

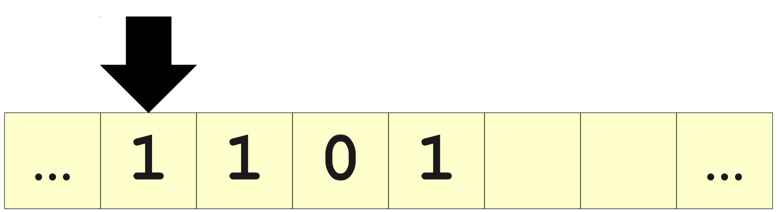
Proof: First, we consider what happens if M accepts w. In this case, consider what happens when we run Amp(M, w) on an arbitrary input string x. Amp(M, w) will run M on w, and since M accepts w, Amp(M, w) accepts x. Since our choice of x was arbitrary, we see that Amp(M, w) accepts any input, so $\mathscr{L}(\text{Amp}(M, w)) = \Sigma^*$.

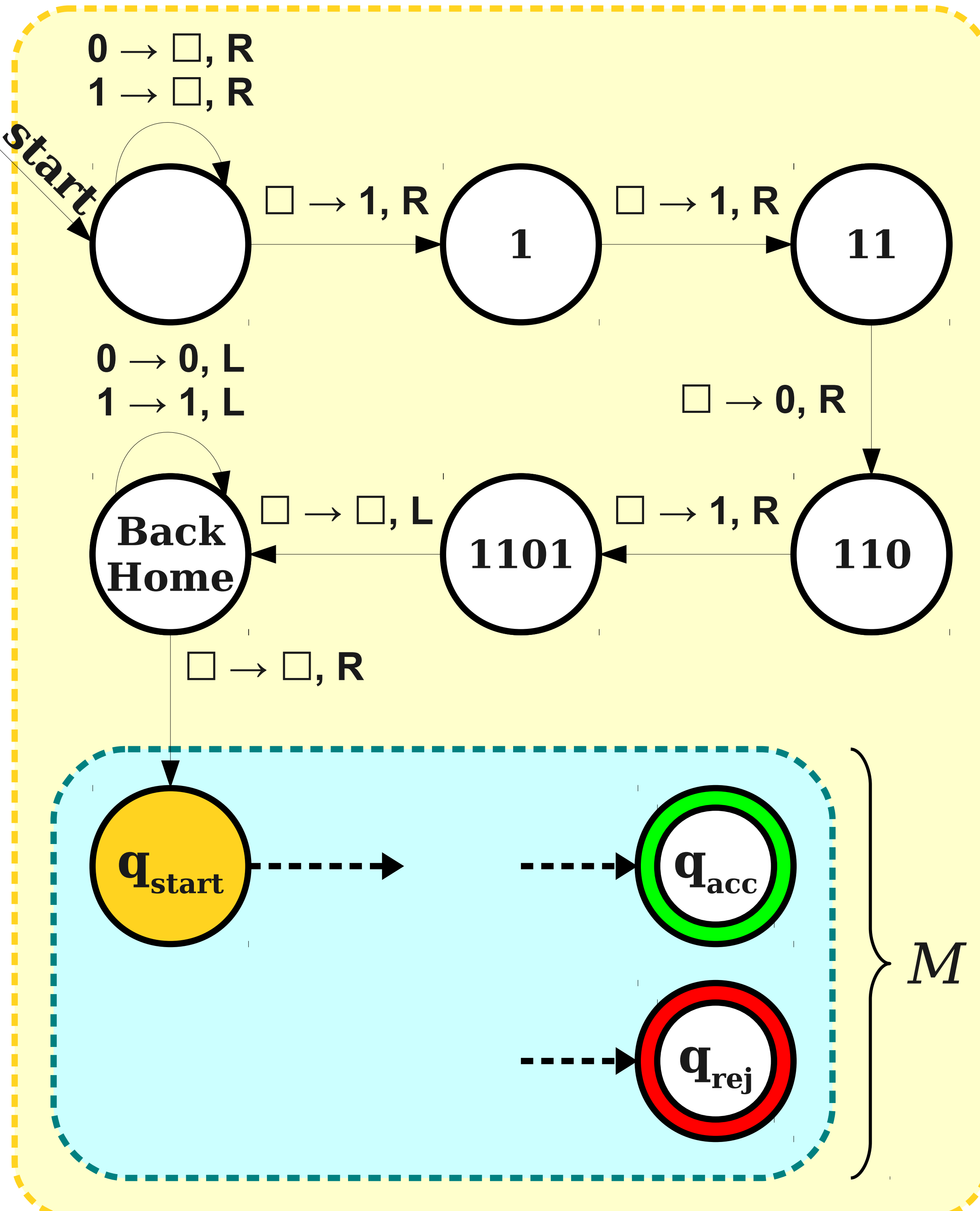
Otherwise, M does not accept w, so M rejects w or M loops on w. Consider the result of running Amp(M, w) on an arbitrary string x. If M rejects w, then Amp(M, w) rejects x. Otherwise, Amp(M, w) loops on x. In both cases, Amp(M, w) doesn't accept x. Since our choice of x was arbitrary, we see that Amp(M, w) never accepts any input, so $\mathscr{L}(\text{Amp}(M, w)) = \emptyset$. ■

"On input x:

- Ignore x.
- Run M on w.
- If M accepts w, we accept x.
- If M rejects w, we reject x."

Hypothetically, assume that w is the string `1101`.





Using the Amplifier

A More Elaborate Reduction

- Since $\overline{\mathrm{A}}_{\mathrm{TM}} \notin$ **RE**, there is no algorithm for determining whether a TM will not accept a given string.
- Could we check instead whether a TM *never* accepts a string?
- Consider the language

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

- How "hard" is L_e? Is it **R**, **RE**, co-**RE**, or none of these?

Building an Intuition

- Before we even try to prove how "hard" this language is, we should build an intuition for its difficulty.
- L_e is *probably* not in **RE**, since if we were convinced a TM never accepted, it would be hard to find positive evidence of this.
- L_e is *probably* in co-**RE**, since if we were convinced that a TM *did* accept some string, we could exhaustively search over all strings and try to find the string it accepts.
- Best guess: $L_e \in$ co-**RE** – **R**.

$\overline{\mathrm{A}}_{\mathrm{TM}} \leq_{\mathrm{M}} L_{\mathrm{e}}$

- We will prove that $L_{\mathrm{e}} \notin \mathbf{RE}$ by showing that $\overline{\mathrm{A}}_{\mathrm{TM}} \leq_{\mathrm{M}} L_{\mathrm{e}}$. (This also proves $L_{\mathrm{e}} \notin \mathbf{R}$).
- We want to find a function f such that

$$\langle M, w\rangle \in \overline{\mathrm{A}}_{\mathrm{TM}} \quad \textbf{iff} \quad f(\langle M, w\rangle) \in L_{\mathrm{e}}$$

- Since L_{e} is a language of TM descriptions, let's assume $f(\langle M, w\rangle) = \langle N\rangle$ for some TM N. Then

$$\langle M, w\rangle \in \overline{\mathrm{A}}_{\mathrm{TM}} \quad \textbf{iff} \quad \langle N\rangle \in L_{\mathrm{e}}$$

- Expanding out definitions, we get

$$M \textbf{ doesn't accept } w \quad \textbf{iff} \quad \mathscr{L}(N) = \varnothing$$

- How do we pick the machine N?

The Reduction

- Choose N such that this holds:

$$\boldsymbol{M} \textbf{ doesn't accept } \boldsymbol{w} \textbf{ iff } \mathscr{L}(\boldsymbol{N}) = \boldsymbol{\emptyset}$$

- We can pick $N = \mathrm{Amp}(M, w)$.
  - **Recall:** $\mathscr{L}(\mathrm{Amp}(M, w)) = \emptyset$ iff M doesn't accept w.
- Since $f(\langle M, w \rangle) = \langle \mathrm{Amp}(M, w) \rangle$ is computable, this is the mapping reduction we need!

Theorem: $L_e \notin \mathbf{RE}$

Proof: We will prove $\overline{A}_{TM} \leq_M L_e$. Since $\overline{A}_{TM} \notin \mathbf{RE}$, this proves that $L_e \notin \mathbf{RE}$, as required. To do so, we will exhibit a mapping reduction from $\overline{A}_{TM}$ to L_e. For any TM/string pair $\langle M, w\rangle$, let $f(\langle M, w\rangle) = \langle \mathrm{Amp}(M, w)\rangle$. By our earlier theorem, this function is computable.

We claim this is a mapping reduction from $\overline{A}_{TM}$ to L_e. To prove this, we will prove that $\langle M, w\rangle \in \overline{A}_{TM}$ iff $\langle \mathrm{Amp}(M, w)\rangle \in L_e$. By definition of $\overline{A}_{TM}$, we see $\langle M, w\rangle$ iff M does not accept w. By our earlier theorem, M does not accept w iff $\mathscr{L}(\mathrm{Amp}(M, w)) = \emptyset$. Finally, by definition of L_e, we see $\mathscr{L}(\mathrm{Amp}(M, w)) = \emptyset$ iff $\langle \mathrm{Amp}(M, w)\rangle \in L_e$. Taken together, we see that $\langle M, w\rangle \in \overline{A}_{TM}$ iff $\langle \mathrm{Amp}(M, w)\rangle \in L_e$, so f is a mapping reduction from $\overline{A}_{TM}$ to L_e. Therefore, we see $\overline{A}_{TM} \leq_M L_e$, as required. ■

A Math Joke

Time-Out For Announcements

Problem Set 6 Graded

- On-time Problem Set 6's have all been graded and should be returned after lecture today.
  - Online submissions: contact us if you don't hear back soon.
- Late Problem Set 6's will be returned this Wednesday.

Problem Set 8 Out

- Problem Set 8 goes out right now. It's due the Monday after Thanksgiving break (December 2).
- Some contradictory information:
  - This is the last problem set on which you can use a late period.
  - We *strongly* recommend that you don't, since you'll be pinched trying to finish Problem Set 9 if you do.
- TAs and I will figure out an OH schedule during Thanksgiving week.

Your Questions

“The fact we can't create a TM for $\overline{A}_{TM}$ and L_D is very cool. But it is tough to see why we would want to solve those problems in the first place – what are problems that we actually want to solve but can't, because of limits of computability?”

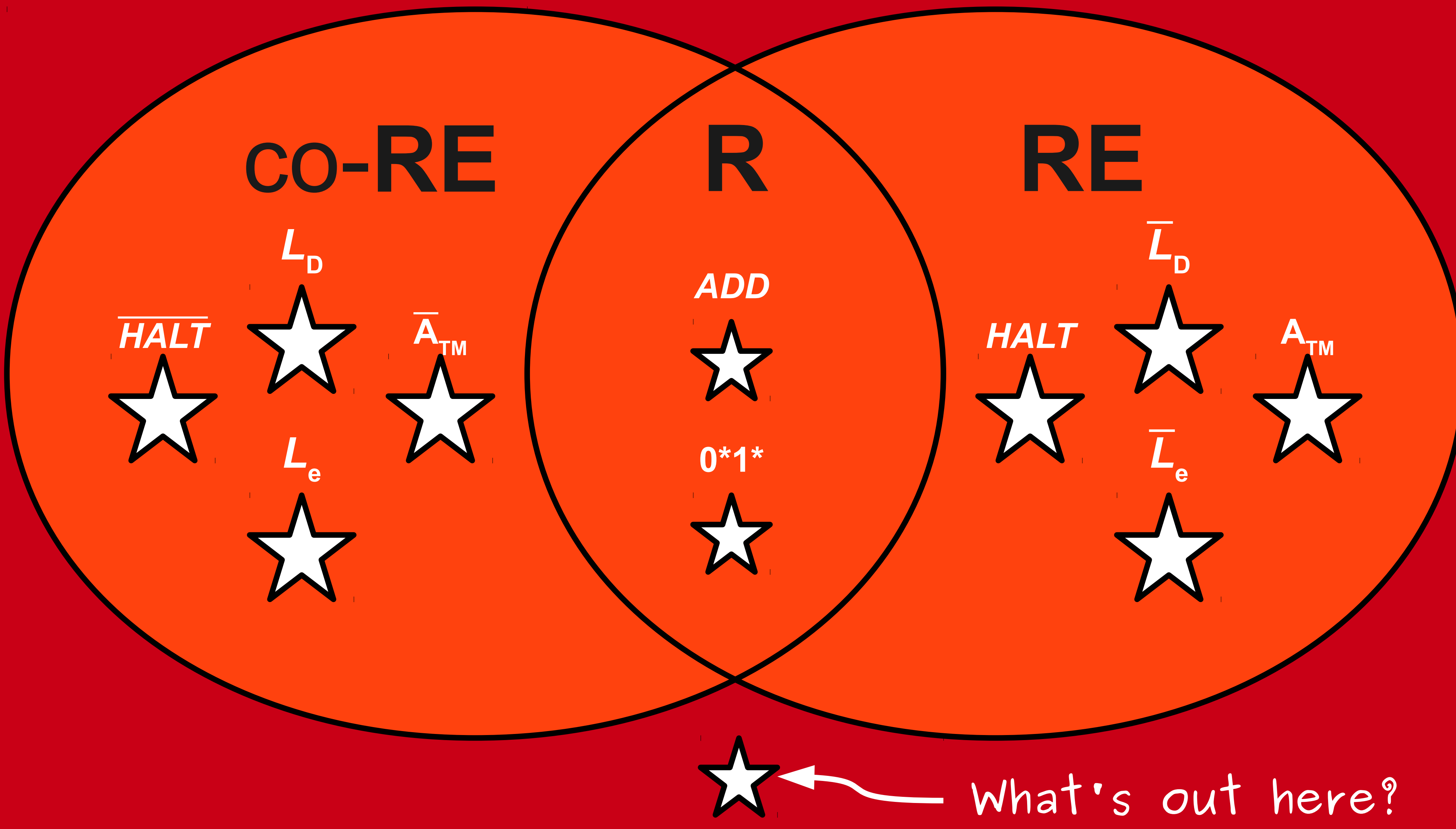
“Aren't there some cases where we can know a TM is infinite looping? Couldn't we modify the U_{TM} so it keeps a record of IDs and then if it sees the same one twice know it was in a loop? This doesn't guarantee to find all loops, but would it be useful?”

“What's the difference between a language being decidable and having a decider for a language?”

“The generalized hailstone sequence terminating is proven to be undecidable (http://link.springer.com/chapter/10.1007%2F978-3-540-72504-6_49). What purpose is there to prove something as undecidable? Is undecidable better than not solvable?”

Back to CS103

The Limits of Computability



RE $\cup$ co-**RE** is Not Everything

- Using the same reasoning as the first day of lecture, we can show that there must be problems that are neither **RE** nor co-**RE**.
- There are more sets of strings than TMs.
- There are more sets of strings than twice the number of TMs.
- What do these languages look like?

TM Equality

- There are infinitely many pairs of Turing machines with the same language as one another.
  - Good exercise: think about why this is.
- Consider the following language:

$$\mathbf{EQ_{TM}} = \{\ \langle \boldsymbol{M_1}, \boldsymbol{M_2} \rangle \mid \boldsymbol{M_1} \textbf{ and } \boldsymbol{M_2} \textbf{ are TMs} \textbf{ and } \mathscr{L}(\boldsymbol{M_1}) = \mathscr{L}(\boldsymbol{M_2})\ \}$$

- Questions:
  - Is $\mathrm{EQ}_{\mathrm{TM}} \in$ co-**RE**?
  - Is $\mathrm{EQ}_{\mathrm{TM}} \in$ **RE**?

Is $\mathrm{EQ}_{\mathrm{TM}} \in$ co-**RE**?

- Intuitively, would we expect $\mathrm{EQ}_{\mathrm{TM}}$ to be a co-**RE** language?
- Suppose TM M_1 accepts a string w. We'd need to know whether M_2 accepts w as well.
- Co-recognizing this would require us to have a corecognizer that detects whether $\langle M_2, w\rangle \in \mathrm{A}_{\mathrm{TM}}$, but that's not an co-**RE** language!
- Our guess: *$\mathrm{EQ}_{\mathrm{TM}}$ is probably not co-**RE***.

Proving $\text{EQ}_{\text{TM}} \notin \text{co-}\mathbf{RE}$

- To prove that $\text{EQ}_{\text{TM}} \notin \text{co-}\mathbf{RE}$, we can try to find a language L where
  - $L \notin \text{co-}\mathbf{RE}$, and
  - $L \leq_{\text{M}} \text{EQ}_{\text{TM}}$
- A good candidate would be something like A_{TM}, which is a "canonical" non-co-**RE** languages.
- **Goal:** Prove $\text{A}_{\text{TM}} \leq_{\text{M}} \text{EQ}_{\text{TM}}$.

Proving $A_{TM} \leq_M EQ_{TM}$

- Goal: Find a computable function f where

$$\boldsymbol{\langle M, w\rangle \in A_{TM} \text{ iff } f(\langle M, w\rangle) \in EQ_{TM}}$$

- Since EQ_{TM} is a language of pairs of TMs, let's assume $f(\langle M\rangle) = \langle M_1, M_2\rangle$. Then we want to pick M_1 and M_2 such that

$$\boldsymbol{\langle M, w\rangle \in A_{TM} \text{ iff } \langle M_1, M_2\rangle \in EQ_{TM}}$$

- Substituting definitions, we want

$$\boldsymbol{M \text{ accepts } w \text{ iff } \mathscr{L}(M_1) = \mathscr{L}(M_2)}$$

- What do we do now?

Using the Amplifier

- We want

$$\boldsymbol{M \textbf{ accepts } w \textbf{ iff } \mathscr{L}(M_1) = \mathscr{L}(M_2)}$$

- What happens if we pick M_1 to be Amp(M, w)?
  - If M accepts w, then $\mathscr{L}(M_1) = \Sigma^*$.
  - If M does not accept w, then $\mathscr{L}(M_1) = \emptyset$.
- Choose M_1 to be the amplifier machine and M_2 to be any TM with language Σ^*. Then the above statement is true!

What's Going On?

- Suppose we have an oracle for EQ_{TM}.
- We want to know whether M accepts w.
- To do this:
  - Find a TM S we know has language Σ^*.
  - Ask the oracle "does TM Amp(M, w) have the same language as TM S?"
  - If so, then M accepts w.
  - If not, then M does not accept w.

Theorem: $\mathrm{EQ}_{\mathrm{TM}} \notin$ co-**RE**.

Proof: We will prove $\mathrm{A}_{\mathrm{TM}} \leq_{\mathrm{M}} \mathrm{EQ}_{\mathrm{TM}}$. Since $\mathrm{A}_{\mathrm{TM}} \notin$ co-**RE**, this proves that $\mathrm{EQ}_{\mathrm{TM}} \notin$ co-**RE**. To show $\mathrm{A}_{\mathrm{TM}} \leq_{\mathrm{M}} \mathrm{EQ}_{\mathrm{TM}}$, we will exhibit a mapping reduction from A_{TM} to $\mathrm{EQ}_{\mathrm{TM}}$.

For any TM/string pair $\langle M, w\rangle$, define $f(\langle M, w\rangle)$ to be the pair of TMs $\langle \mathrm{Amp}(M, w), S\rangle$, where S is the TM "On input x, accept x." This function is computable, and note that $\mathscr{L}(S) = \Sigma^*$.

We claim that $\langle M, w\rangle \in \mathrm{A}_{\mathrm{TM}}$ iff $\langle \mathrm{Amp}(M, w), E\rangle \in \mathrm{EQ}_{\mathrm{TM}}$. To see this, note by definition of A_{TM} that $\langle M, w\rangle \in \mathrm{A}_{\mathrm{TM}}$ iff M accepts w. By our earlier theorem, M accepts w iff $\mathscr{L}(\mathrm{Amp}(M, w)) = \Sigma^*$. Since $\mathscr{L}(S) = \Sigma^*$, we see M accepts w iff $\mathscr{L}(\mathrm{Amp}(M, w)) = \mathscr{L}(S)$. Finally, by definition of $\mathrm{EQ}_{\mathrm{TM}}$, $\mathscr{L}(\mathrm{Amp}(M, w)) = \mathscr{L}(S)$ iff $\langle \mathrm{Amp}(M, w), S\rangle \in \mathrm{EQ}_{\mathrm{TM}}$. Collectively, we see $\langle M, w\rangle \in \mathrm{A}_{\mathrm{TM}}$ iff $\langle \mathrm{Amp}(M, w), S\rangle \in \mathrm{EQ}_{\mathrm{TM}}$.

Thus f is a mapping reduction from A_{TM} to $\mathrm{EQ}_{\mathrm{TM}}$, so $\mathrm{A}_{\mathrm{TM}} \leq_{\mathrm{M}} \mathrm{EQ}_{\mathrm{TM}}$, as required. ■

Is $\mathrm{EQ}_{\mathrm{TM}} \in \mathbf{RE}$?

- Intuitively, would we expect $\mathrm{EQ}_{\mathrm{TM}}$ to be a **RE** language?
- Suppose TM M_1 doesn't accept a string w. We'd need to know whether M_2 also doesn't accept w.
- Recognizing this would require us to have a recognizer that detects whether $\langle M_2, w\rangle \in \overline{\mathrm{A}}_{\mathrm{TM}}$, but that's not an **RE** language!
- Our guess: *EQ_{TM} is probably not* ***RE***.

Proving $\overline{A}_{TM} \leq_M EQ_{TM}$

- Goal: Find a computable function f where

$$\boldsymbol{\langle M, w\rangle \in \overline{A}_{TM} \text{ iff } f(\langle M, w\rangle) \in EQ_{TM}}$$

- Since EQ_{TM} is a language of pairs of TMs, let's assume $f(\langle M\rangle) = \langle M_1, M_2\rangle$. Then we want to pick M_1 and M_2 such that

$$\boldsymbol{\langle M, w\rangle \in \overline{A}_{TM} \text{ iff } \langle M_1, M_2\rangle \in EQ_{TM}}$$

- Substituting definitions, we want

$$\boldsymbol{M \text{ does not accept } w \text{ iff } \mathscr{L}(M_1) = \mathscr{L}(M_2)}$$

- What do we do now?

Using the Amplifier

- We want

  ***M* does not accept *w* iff $\mathscr{L}(M_1) = \mathscr{L}(M_2)$**

- What happens if we pick M_1 to be Amp(M, w)?
  - If M accepts w, then $\mathscr{L}(M_1) = \Sigma^*$.
  - If M does not accept w, then $\mathscr{L}(M_1) = \emptyset$.
- Choose M_1 to be the amplifier machine and M_2 to be any TM with language $\emptyset$. Then the above statement is true!

What's Going On?

- Suppose we have an oracle for EQ_{TM}.
- We want to know whether M accepts w.
- To do this:
  - Find a TM E we know has language Ø.
  - Ask the oracle “does TM Amp(M, w) have the same language as TM E?”
  - If so, then M does not accept w.
  - If not, then M accepts w.

Theorem: $\mathrm{EQ_{TM}} \notin \mathbf{RE}$.

Proof: We will prove $\overline{\mathrm{A}}_{\mathrm{TM}} \leq_{\mathrm{M}} \mathrm{EQ_{TM}}$. Since $\overline{\mathrm{A}}_{\mathrm{TM}} \notin \mathbf{RE}$, this proves that $\mathrm{EQ_{TM}} \notin \mathbf{RE}$. To show $\overline{\mathrm{A}}_{\mathrm{TM}} \leq_{\mathrm{M}} \mathrm{EQ_{TM}}$, we will exhibit a mapping reduction from $\overline{\mathrm{A}}_{\mathrm{TM}}$ to $\mathrm{EQ_{TM}}$.

For any TM/string pair $\langle M, w\rangle$, define $f(\langle M, w\rangle)$ to be the pair of TMs $\langle \mathrm{Amp}(M, w), E\rangle$, where E is the TM "On input x, reject x." This function is computable, and note that $\mathscr{L}(E) = \emptyset$.

We claim that $\langle M, w\rangle \in \overline{\mathrm{A}}_{\mathrm{TM}}$ iff $\langle \mathrm{Amp}(M, w), E\rangle \in \mathrm{EQ_{TM}}$. To see this, note by definition of $\overline{\mathrm{A}}_{\mathrm{TM}}$ that $\langle M, w\rangle \in \overline{\mathrm{A}}_{\mathrm{TM}}$ iff M does not accept w. By our theorem, M does not accept w iff $\mathscr{L}(\mathrm{Amp}(M, w)) = \emptyset$. Since $\mathscr{L}(E) = \emptyset$, we see M does not accept w iff $\mathscr{L}(\mathrm{Amp}(M, w)) = \mathscr{L}(E)$. Finally, by definition of $\mathrm{EQ_{TM}}$, $\mathscr{L}(\mathrm{Amp}(M, w)) = \mathscr{L}(E)$ iff $\langle \mathrm{Amp}(M, w), E\rangle \in \mathrm{EQ_{TM}}$. Collectively, we see $\langle M, w\rangle \in \overline{\mathrm{A}}_{\mathrm{TM}}$ iff $\langle \mathrm{Amp}(M, w), E\rangle \in \mathrm{EQ_{TM}}$.

Thus f is a mapping reduction from $\overline{\mathrm{A}}_{\mathrm{TM}}$ to $\mathrm{EQ_{TM}}$, so $\overline{\mathrm{A}}_{\mathrm{TM}} \leq_{\mathrm{M}} \mathrm{EQ_{TM}}$, as required. ■

The Limits of Computability