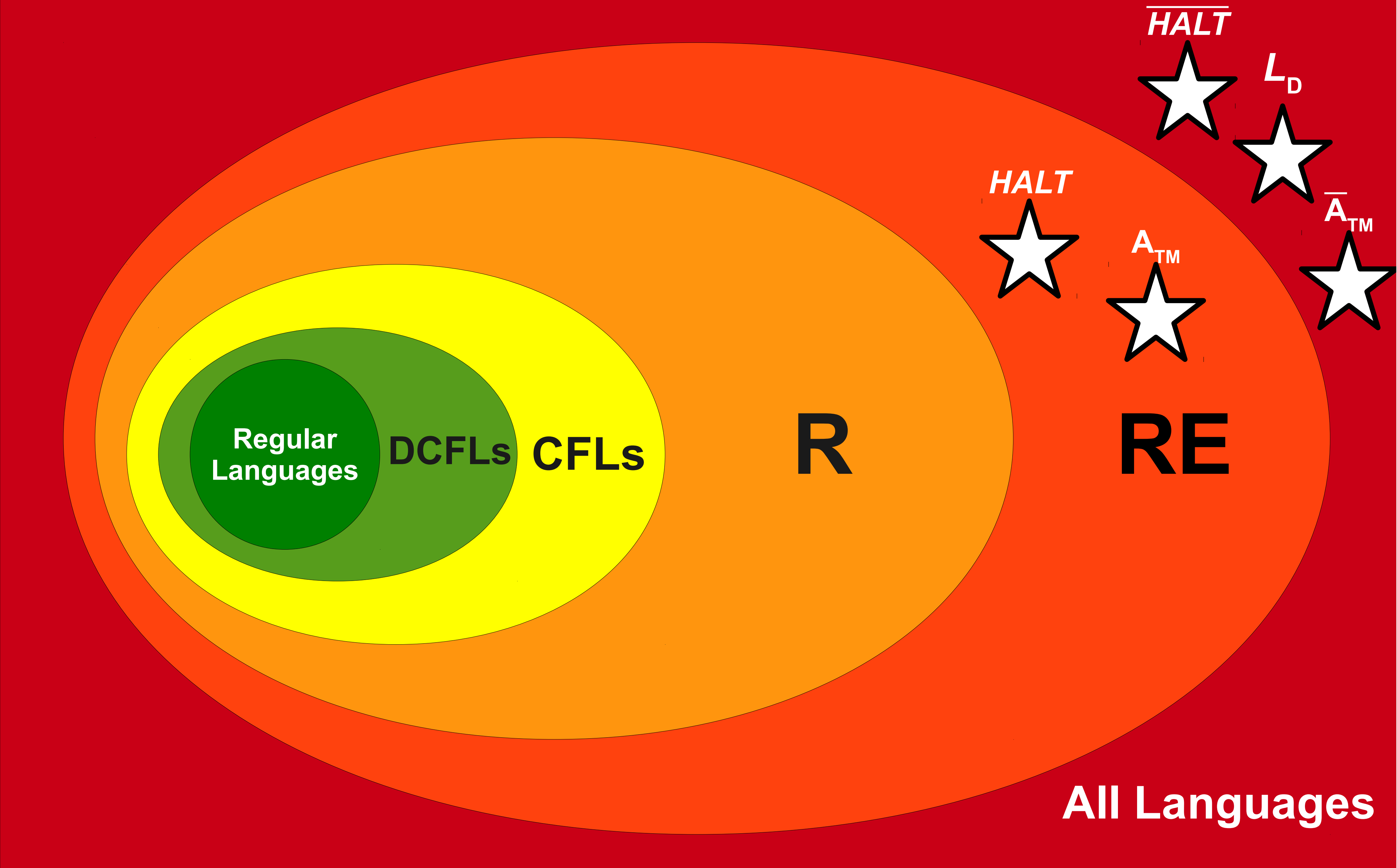
Reductions

The Limits of Computability



$HALT$ and $\overline{HALT}$

- The language $HALT$ is defined as

$$\{\langle M, w\rangle \mid M \text{ is a TM that halts on } w\}$$

- Equivalently:

$$\{x \mid x = \langle M, w\rangle \text{ for some TM } M \text{ and string } w\text{, and } M \text{ halts on } w\}$$

- Thus $\overline{HALT}$ is

$$\{x \mid x \neq \langle M, w\rangle \text{ for any TM } M \text{ and string } w\text{, or } M \text{ is a TM that does not halt on } w\}$$

Cheating With Math

- As a mathematical simplification, we will assume the following:

Every string can be decoded into any collection of objects.

- Every string is an encoding of some TM M.
- Every string is an encoding of some TM M and string w.
- Can do this as follows:
  - If the string is a legal encoding, go with that encoding.
  - Otherwise, pretend the string decodes to some predetermined group of objects.

Cheating With Math

- Example: Every string will be a valid C++ program.
- If it's already a C++ program, just compile it.
- Otherwise, pretend it's this program:

```
int main() {
    return 0;
}
```

$HALT$ and $\overline{HALT}$

- The language $HALT$ is defined as

  $$\{\langle M, w\rangle \mid M \text{ is a TM that halts on } w\}$$

- Thus $\overline{HALT}$ is the language

  $$\{\langle M, w\rangle \mid M \text{ is a TM that doesn't halt on } w\}$$

- Equivalently:

  $$\overline{HALT} = \{\langle M, w\rangle \mid M \text{ is a TM that loops on } w\}$$

The Takeaway Point

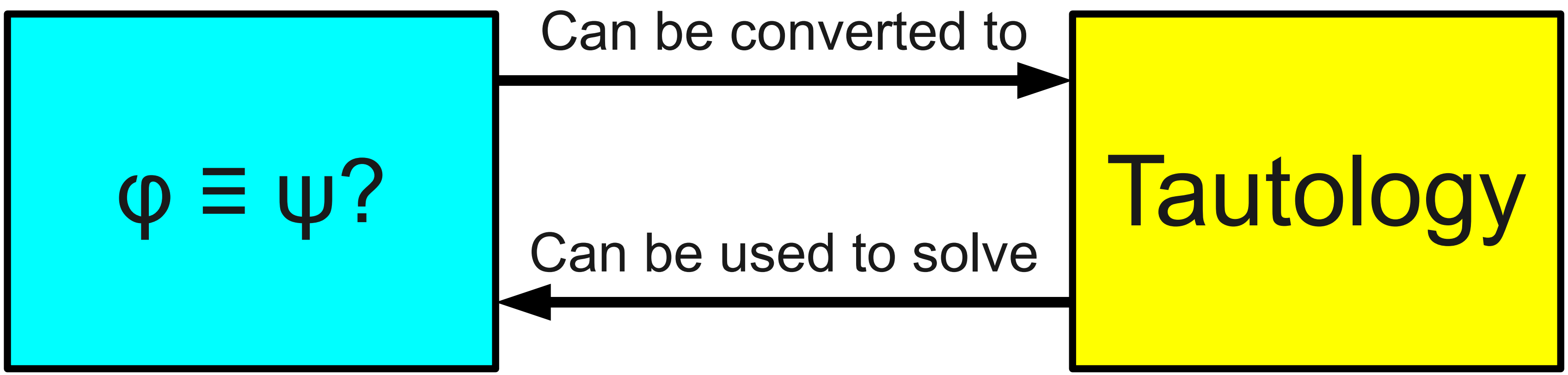
- When dealing with encodings, you don't need to consider strings that aren't valid encodings.
- This will keep our proofs *much* simpler than before.

Reductions

Finding Unsolvable Problems

- Last time, we found five unsolvable problems.
- We proved that L_D was unrecognizable, then used this fact to show four other languages were either undecidable or unrecognizable.
- In general, to prove that a problem is unsolvable (not **R** or not **RE**), we don't directly show that it is unsolvable.
- Instead, we show how a solution to that problem would let us solve an unsolvable problem.

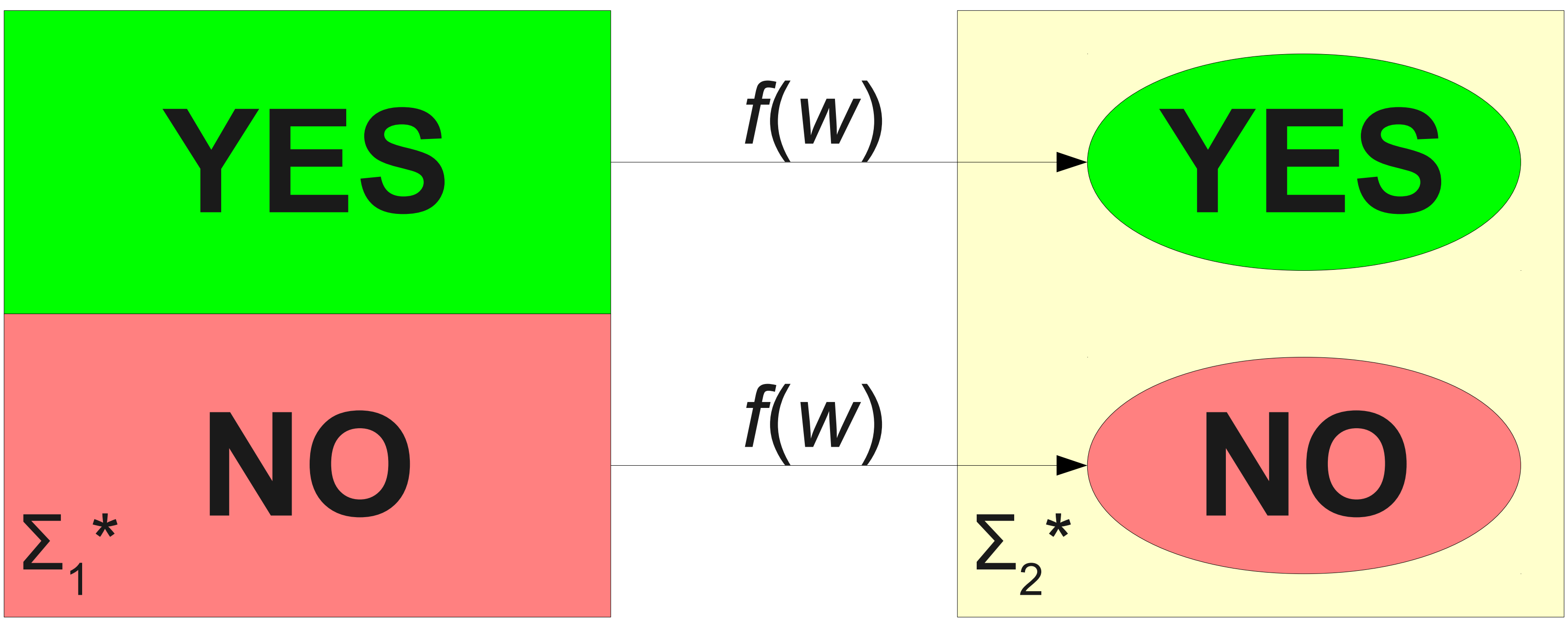
Reductions



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

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

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

Reducing $\varphi \equiv \psi$ to Tautology

- Let *EQUIV* be

$$\boldsymbol{EQUIV = \{ \langle \varphi, \psi \rangle \mid \varphi \equiv \psi \}}$$

- Let *TAUTOLOGY* be

$$\boldsymbol{TAUTOLOGY = \{ \langle \varphi \rangle \mid \varphi \text{ is a tautology} \}}$$

- To reduce *EQUIV* to *TAUTOLOGY*, we want a function *f* such that

$$\boldsymbol{\langle \varphi, \psi \rangle \in EQUIV \quad \text{iff} \quad f(\langle \varphi, \psi \rangle) \in TAUTOLOGY}$$

- One possible function we could use is

$$\boldsymbol{f(\langle \varphi, \psi \rangle) = \langle \varphi \leftrightarrow \psi \rangle}$$

Reducing any **RE** Language to A_{TM}

- Let L be any **RE** language, and let R be a recognizer for L.
- To reduce L to A_{TM}, we want a function f such that

$$\boldsymbol{w \in L \quad \text{iff} \quad f(w) \in A_{TM}}$$

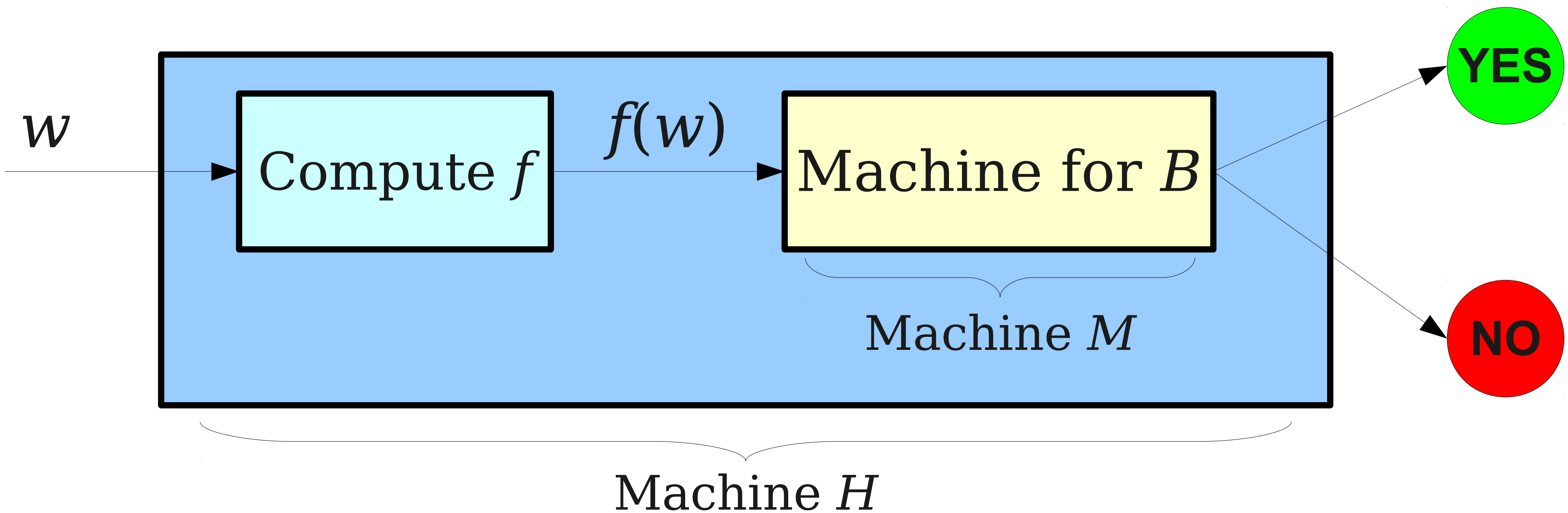
- One possible reduction is

$$\boldsymbol{f(w) = \langle R, w \rangle}$$

Why Reductions Matter

- If problem A reduces to problem B, we can use a recognizer/decider for B to recognize/decide problem A.
  - (There's a slight catch – we'll talk about this in a second).
- How is this possible?

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



H = "On input w:
Compute $f(w)$.
Run M on $f(w)$.
If M accepts $f(w)$, accept w.
If M rejects $f(w)$, reject w."

H* accepts *w

iff

***M* accepts *f*(*w*)**

iff

$f(w) \in B$

iff

$w \in A$

A Problem

- Recall: f is a reduction from A to B iff

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

- Under this definition, *any* language A reduces to *any* language B unless $B = \emptyset$ or Σ^*.
- Since $B \neq \emptyset$ and $B \neq \Sigma^*$, there is some $w_{yes} \in B$ and some $w_{no} \notin B$.
- Define $f : \Sigma_1^* \to \Sigma_2^*$ as follows:

$$\textbf{If } \boldsymbol{w \in A}\textbf{, then } \boldsymbol{f(w) = w_{yes}}$$

$$\textbf{If } \boldsymbol{w \notin A}\textbf{, then } \boldsymbol{f(w) = w_{no}}$$

- Then f is a reduction from A to B.

A Problem

- Example: let's reduce L_D to `0`*`1`*.
- Take w_{yes} = `01`, w_{no} = `10`.
- Then $f(w)$ is defined as
  - If $w \in L_D$, $f(w)$ = `01`.
  - If $w \notin L_D$, $f(w)$ = `10`.
- There is no TM that can actually evaluate the function $f(w)$ on all inputs, since no TM can decide whether or not $w \in L_D$.

Computable Functions

- This general reduction is mathematically well-defined, but might be impossible to actually compute!
- To fix our definition, we need to introduce the idea of a computable function.
- A function $f : \Sigma_1^* \to \Sigma_2^*$ is called a **computable function** if there is some TM M with the following behavior:

  “On input w:

    Determine the value of $f(w)$.

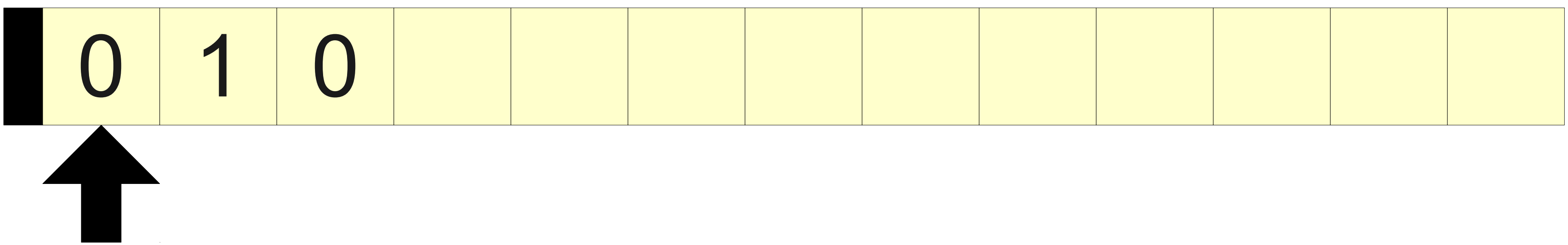
    Write $f(w)$ on the tape.

    Move the tape head back to the far left.

    Halt.”

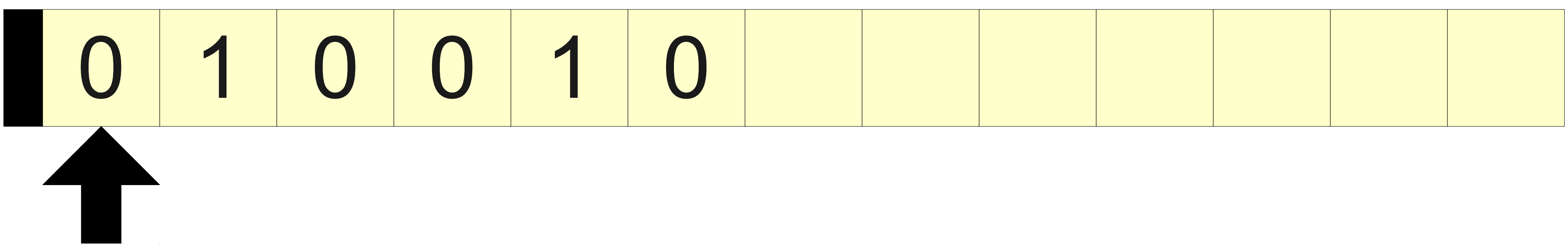
Computable Functions

$$f(w) = ww$$



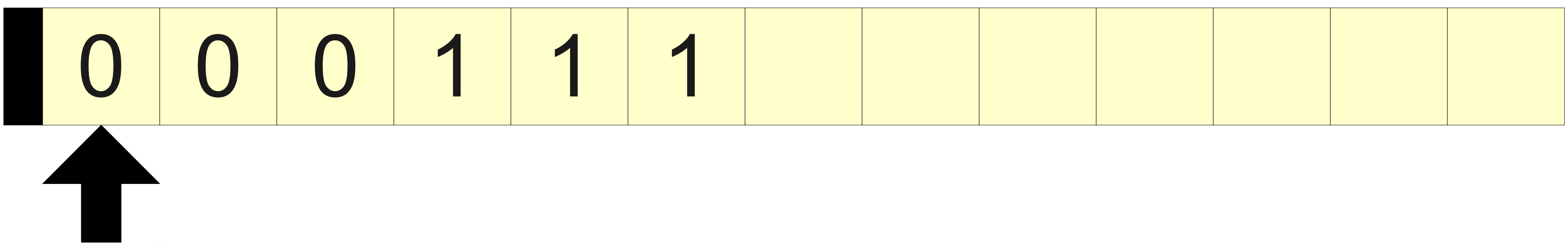
Computable Functions

$f(w) = ww$



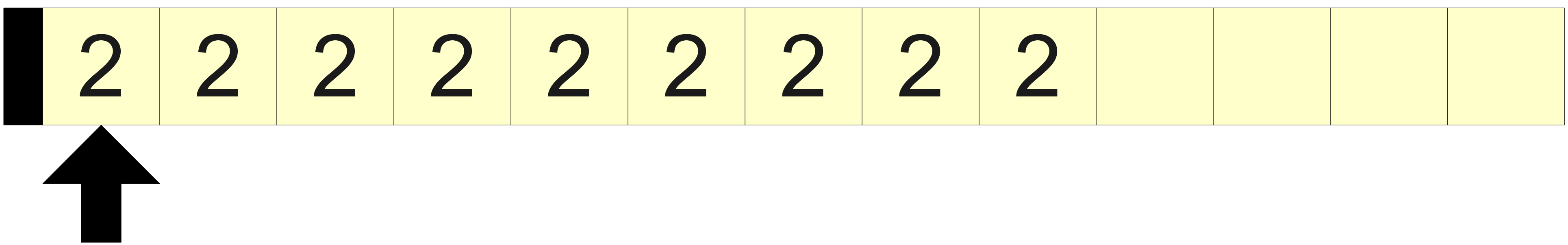
Computable Functions

$$f(w) = \begin{cases} 2^{nm} & \text{if } w = 0^n1^m \\ \varepsilon & \text{otherwise} \end{cases}$$



Computable Functions

$$f(w) = \begin{cases} 2^{nm} & \text{if } w = 0^n1^m \\ \varepsilon & \text{otherwise} \end{cases}$$



Mapping Reductions

- A function $f : \Sigma_1^* \rightarrow \Sigma_2^*$ 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 A to B, we say that A is **mapping reducible** to B.
- Notation: $\boldsymbol{A \leq_M B}$ iff A is mapping reducible to B.
- This is not a partial order (it's not antisymmetric), but it is reflexive and transitive. (*Why?*)

Why Mapping Reducibility Matters

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

Why Mapping Reducibility Matters

If this one is "easy" (**R** or **RE**)…

$$A \leq_M B$$

… then this one is "easy" (**R** or **RE**) too.

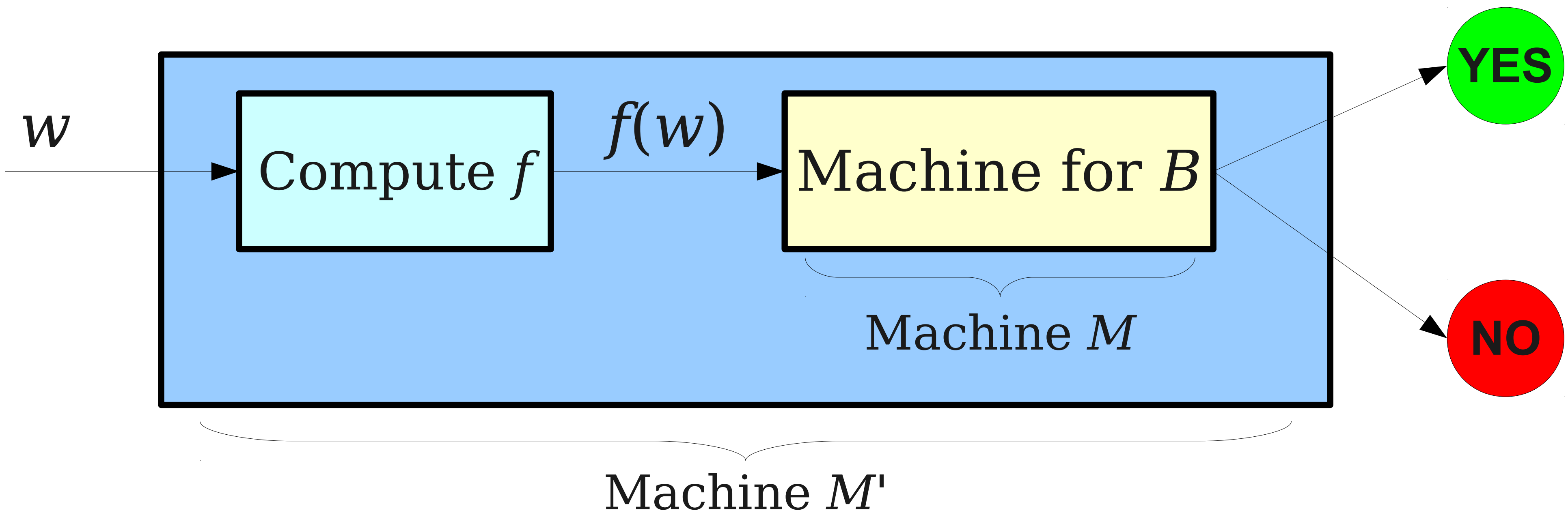
Why Mapping Reducibility Matters

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

$$A \leq_M B$$

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

$A \leq_M B$



M' = "On input w:
Compute $f(w)$.
Run M on $f(w)$.
If M accepts $f(w)$, accept w.
If M rejects $f(w)$, reject w."

M'* accepts *w

iff

***M* accepts *f*(*w*)**

iff

$f(w) \in B$

iff

$w \in A$

Using Reductions

Using Reductions

- Recall: The language A_{TM} is defined as

$$A_{TM} = \{ \langle M, w \rangle \mid M \text{ is a TM and } w \in \mathscr{L}(M) \}$$

- Last time, we proved that $A_{TM} \in \mathbf{RE} - \mathbf{R}$ (that is, $A_{TM} \in \mathbf{RE}$ but $A_{TM} \notin \mathbf{R}$) by showing that a decider for A_{TM} could be converted into a decider for the diagonalization language L_D.
- Let's see an alternate proof that A_{TM} is undecidable by using reductions.

The Complement of A_{TM}

- Recall: if $\mathrm{A}_{\mathrm{TM}} \in \mathbf{R}$, then $\overline{\mathrm{A}}_{\mathrm{TM}} \in \mathbf{R}$ as well.
- To show that A_{TM} is undecidable, we will prove that the *complement* of A_{TM} (denoted $\overline{\mathrm{A}}_{\mathrm{TM}}$) is undecidable.
- The language $\overline{\mathrm{A}}_{\mathrm{TM}}$ is the following:

$$\overline{\mathbf{A}}_{\mathbf{TM}} = \{ \langle M, w \rangle \mid M \textbf{ is a TM and } w \notin \mathscr{L}(M) \}$$

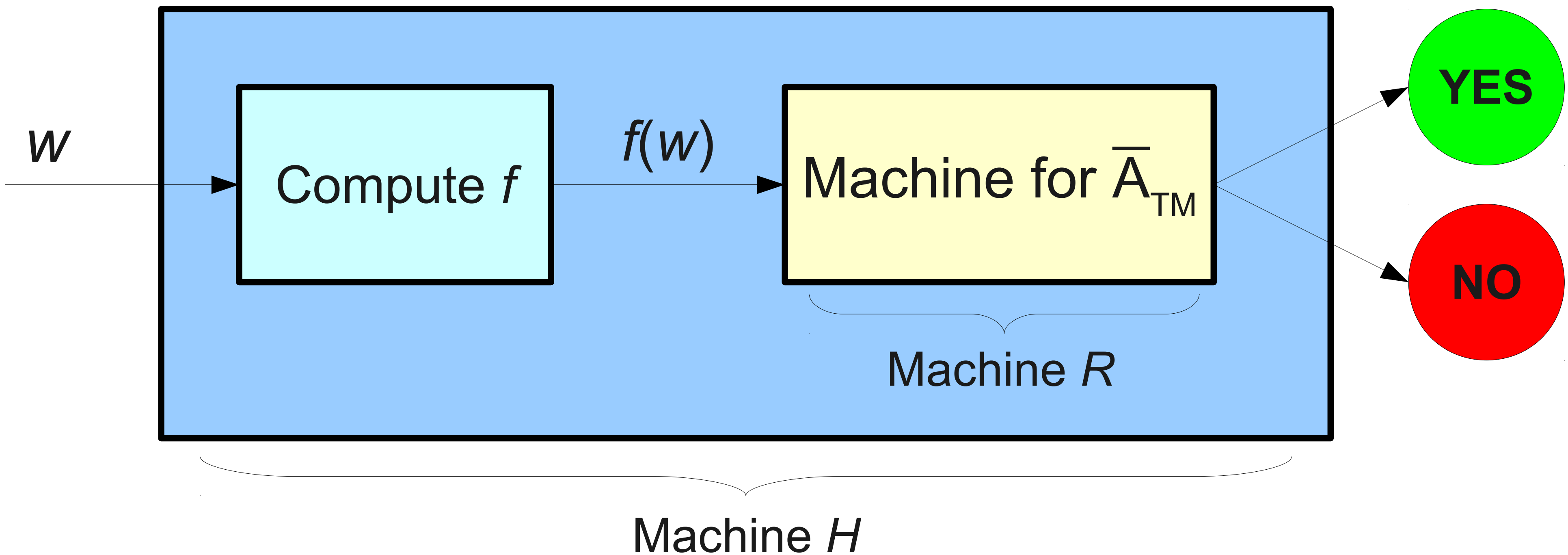
$L_D \leq_M \overline{A}_{TM}$

- Recall: The diagonalization language L_D is the language

  $$\mathbf{L_D = \{ \langle M \rangle \mid M \text{ is a TM and } \langle M \rangle \notin \mathscr{L}(M) \}}$$

- We directly established that $L_D \notin \mathbf{RE}$ using a diagonal argument.
- If we can show that $L_D \leq_M \overline{A}_{TM}$, then since $L_D \notin \mathbf{RE}$, we have proven that $\overline{A}_{TM} \notin \mathbf{RE}$.
- Therefore, $\overline{A}_{TM} \notin \mathbf{R}$, so $A_{TM} \notin \mathbf{R}$.

Where We're Going



Goal: Choose our function f(w) such that this machine H is a recognizer for L_D.

L_{D} and $\overline{\text{A}}_{\text{TM}}$

- L_{D} and $\overline{\text{A}}_{\text{TM}}$ are similar languages:

$$\langle M \rangle \in L_{\text{D}} \quad \textbf{iff} \quad \langle M \rangle \notin \mathscr{L}(M)$$

$$\langle M, w \rangle \in \overline{\text{A}}_{\text{TM}} \quad \textbf{iff} \quad w \notin \mathscr{L}(M)$$

- $\overline{\text{A}}_{\text{TM}}$ is more general than L_{D}:
  - L_{D} asks if a machine doesn't accept *itself*.
  - $\overline{\text{A}}_{\text{TM}}$ asks if a machine doesn't accept *some specific string*.

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

- Goal: Find a computable function f such that

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

- Simplifying this using the definition of L_D

$$\langle M \rangle \notin \mathscr{L}(M) \quad \textbf{iff} \quad f(\langle M \rangle) \in \overline{A}_{TM}$$

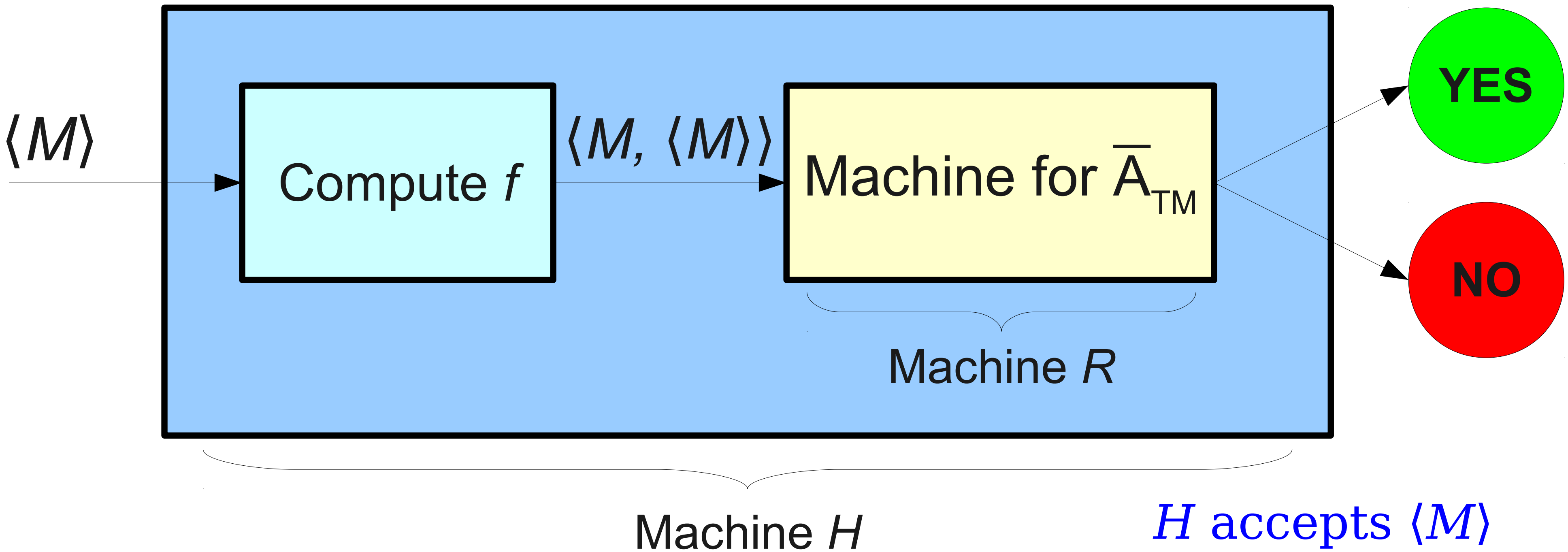
- Let's assume that $f(\langle M \rangle)$ has the form $\langle M', w \rangle$ for some TM M' and string w. This means that

$$\langle M \rangle \notin \mathscr{L}(M) \quad \textbf{iff} \quad \langle M', w \rangle \in \overline{A}_{TM}$$

$$\langle M \rangle \notin \mathscr{L}(M) \quad \textbf{iff} \quad w \notin \mathscr{L}(M')$$

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

What We Just Did



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

Compute $\langle M, \langle M \rangle \rangle$.

Run R on $\langle M, \langle M \rangle \rangle$.

If R accepts $\langle M, \langle M \rangle \rangle$, accept $\langle M \rangle$.

If R rejects $\langle M, \langle M \rangle \rangle$, reject $\langle M \rangle$."

H accepts $\langle M \rangle$

iff

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

iff

$\langle M, \langle M \rangle \rangle \in \overline{A}_{TM}$

iff

$\langle M \rangle \notin \mathscr{L}(M)$

iff

$\langle M \rangle \in L_D$

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

- The final version of our function *f* is defined here:

$$\boldsymbol{f(\langle M \rangle) = \langle M, \langle M \rangle \rangle}$$

- It's reasonable to assume that *f* is computable; details are left as an exercise.
- If we can formally prove that $\langle M \rangle \in L_D$ iff $f(\langle M \rangle) \in \overline{A}_{TM}$, then we have that $L_D \leq_M \overline{A}_{TM}$. Thus $\overline{A}_{TM} \notin$ **RE**.

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

Proof: We exhibit a mapping reduction f from L_{D} to $\overline{\mathrm{A}}_{\mathrm{TM}}$.

Consider the function f defined as follows:

$$f(\langle M \rangle) = \langle M, \langle M \rangle \rangle$$

We claim that f can be computed by a TM and omit the details from this proof. We will prove that $\langle M \rangle \in L_{\mathrm{D}}$ iff $f(\langle M \rangle) \in \overline{\mathrm{A}}_{\mathrm{TM}}$. Note that $f(\langle M \rangle) = \langle M, \langle M \rangle \rangle$, so $f(\langle M \rangle) \in \overline{\mathrm{A}}_{\mathrm{TM}}$ iff $\langle M, \langle M \rangle \rangle \in \overline{\mathrm{A}}_{\mathrm{TM}}$. By definition of $\overline{\mathrm{A}}_{\mathrm{TM}}$, $\langle M, \langle M \rangle \rangle \in \overline{\mathrm{A}}_{\mathrm{TM}}$ iff $\langle M \rangle \notin \mathscr{L}(M)$. Finally, note that $\langle M \rangle \notin \mathscr{L}(M)$ iff $\langle M \rangle \in L_{\mathrm{D}}$. Thus $f(\langle M \rangle) \in \overline{\mathrm{A}}_{\mathrm{TM}}$ iff $\langle M \rangle \in L_{\mathrm{D}}$, so f is a mapping reduction from L_{D} to $\overline{\mathrm{A}}_{\mathrm{TM}}$.

Since f is a mapping reduction from L_{D} to $\overline{\mathrm{A}}_{\mathrm{TM}}$, we have $L_{\mathrm{D}} \leq_{\mathrm{M}} \overline{\mathrm{A}}_{\mathrm{TM}}$. Since $L_{\mathrm{D}} \notin \mathbf{RE}$ and $L_{\mathrm{D}} \leq_{\mathrm{M}} \overline{\mathrm{A}}_{\mathrm{TM}}$, this means $\overline{\mathrm{A}}_{\mathrm{TM}} \notin \mathbf{RE}$, as required. ■

The Halting Problem

- Recall the definition of *HALT*:

  $\boldsymbol{HALT = \{ \langle M, w \rangle \mid M \textbf{ is a TM that halts on } w \}}$

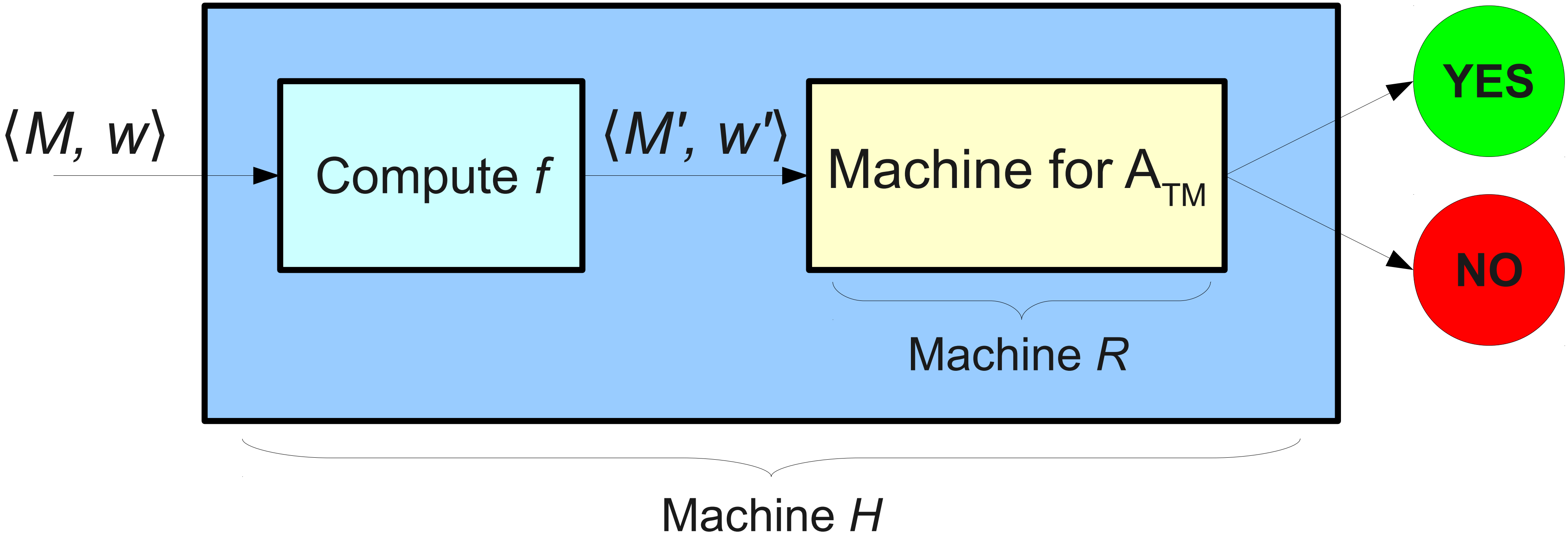
- That is, the set of TM / string pairs where the TM *M* either accepts or rejects the string *w*.
- Last time, we proved that *HALT* ∈ **RE** – **R** by building a TM for it, then by showing a decider for *HALT* could be turned into a decider for A_{TM}.
- Let's explore an alternate proof using mapping reductions.

HALT is **RE**

- Recall: $A_{TM} \in \mathbf{RE}$.
- To prove that *HALT* is **RE**, we will show that $HALT \leq_M A_{TM}$.
- Since $A_{TM} \in \mathbf{RE}$, this proves $HALT \in \mathbf{RE}$.
- Idea: we need to find some function f such that

$$\langle M, w \rangle \in HALT \quad \textbf{iff} \quad f(\langle M, w \rangle) \in A_{TM}$$

Where We're Going



Goal: Choose our function $f(w)$ such that this machine H is a recognizer for $HALT$.

$HALT \leq_M A_{TM}$

- Goal: Find a function f such that

$$\langle M, w\rangle \in HALT \quad \textbf{iff} \quad f(\langle M, w\rangle) \in A_{TM}$$

- Substituting the definitions:

$$M \textbf{ halts on } w \quad \textbf{iff} \quad f(\langle M, w\rangle) \in A_{TM}.$$

- Assume that $f(\langle M, w\rangle) = \langle M', w'\rangle$ for some TM M' and string w'. Then we have

$$M \textbf{ halts on } w \quad \textbf{iff} \quad \langle M', w'\rangle \in A_{TM}$$

$$M \textbf{ halts on } w \quad \textbf{iff} \quad w' \in \mathscr{L}(M')$$

$$M \textbf{ halts on } w \quad \textbf{iff} \quad M' \textbf{ accepts } w'$$

Choosing M' and w'

- We need to find M' and w' such that

  ***M* halts on *w* iff *M*' accepts *w*'.**

- This is the creative step of the proof – how do we choose an M' and w' with that property?
- **Key idea that shows up in almost all major reduction proofs**: Construct a machine M' and string w' so that running M' on w' runs M on w.
- This causes the behavior of M' running on w' to depend on what M does on w.

Choosing M' and w'

- Here is one possible choice of M' and w' we can make:

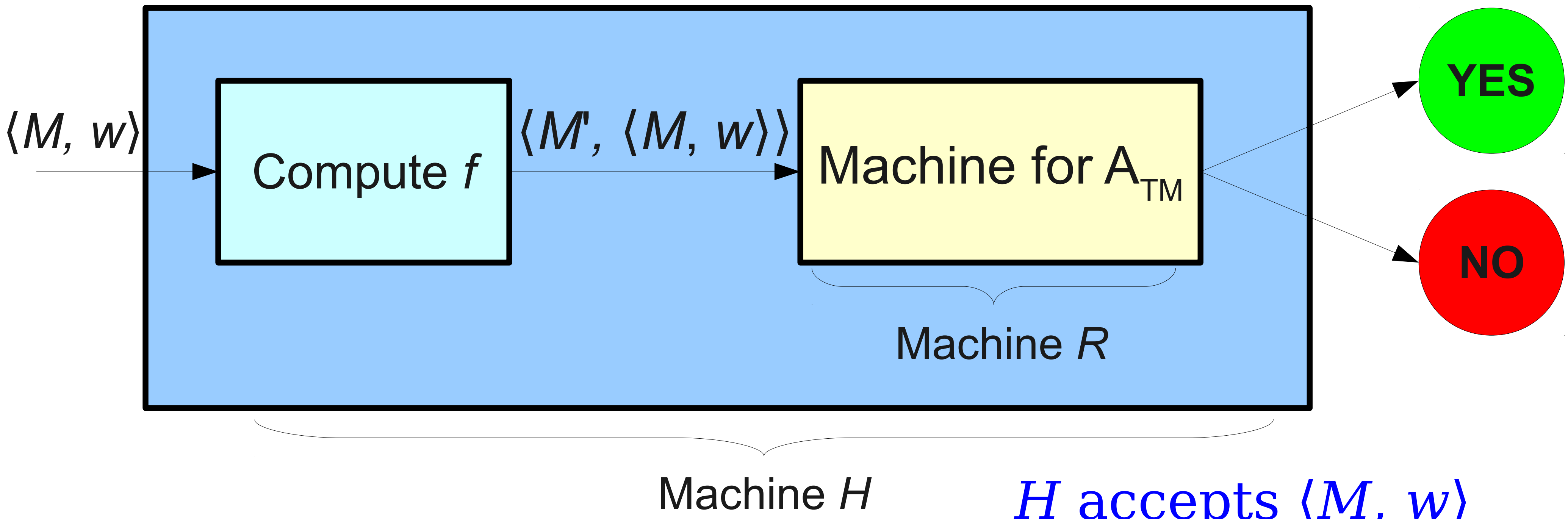
  M' = "On input $\langle N, z\rangle$:

  Run N on z.

  If N halts on z, accept."

  $w' = \langle M, w\rangle$

- Now, running M' on w' runs M on w. If M halts on w, then M' accepts w'. If M loops on w, then M' does not accept w'.



H accepts $\langle M, w\rangle$
iff
R accepts $\langle M', \langle M, w\rangle\rangle$
iff
$\langle M', \langle M, w\rangle\rangle \in A_{TM}$
iff
M' accepts $\langle M, w\rangle$
iff
M halts on w
iff
$\langle M, w\rangle \in HALT$

M' = "On input $\langle N, z\rangle$:
Run N on z.
If N halts, accept."

H = "On input $\langle M, w\rangle$:
Compute $\langle M', \langle M, w\rangle\rangle$.
Run R on $\langle M', \langle M, w\rangle\rangle$.
If R accepts $\langle M', \langle M, w\rangle\rangle$, accept.
If R rejects $\langle M', \langle M, w\rangle\rangle$, reject."

Theorem: $HALT \leq_M A_{TM}$.

Proof: We exhibit a mapping reduction f from $HALT$ to A_{TM}.

Let the machine M' be defined as follows:

M' = "On input $\langle N, z\rangle$:
Run N on z.
If N halts on z, accept."

Then let $f(\langle M, w\rangle) = \langle M', \langle M, w\rangle\rangle$. We claim that f is computable and omit the details from this proof. We further claim that $\langle M, w\rangle \in HALT$ iff $f(\langle M, w\rangle) \in A_{TM}$. To see this, note that $f(\langle M, w\rangle) = \langle M', \langle M, w\rangle\rangle \in A_{TM}$ iff M' accepts $\langle M, w\rangle$. By construction, M' accepts $\langle M, w\rangle$ iff M halts on w. Finally, note that M halts on w iff $\langle M, w\rangle \in HALT$. Thus $\langle M, w\rangle \in HALT$ iff $f(\langle M, w\rangle) \in A_{TM}$. Therefore, f is a mapping reduction from $HALT$ to A_{TM}, so $HALT \leq_M A_{TM}$. ■

HALT is Undecidable

- We proved *HALT* $\in$ **RE** by showing that *HALT* $\leq_M A_{TM}$.
- We can prove *HALT* $\notin$ **R** by showing that $A_{TM} \leq_M$ *HALT*.
- Note that this has to be a completely separate reduction! We're transforming A_{TM} into *HALT* this time, not the other way around.

$A_{TM} \leq_M HALT$

- We want to find a computable function f such that

$$\boldsymbol{\langle M, w\rangle \in A_{TM} \quad \textbf{iff} \quad f(\langle M, w\rangle) \in HALT.}$$

- Assume $f(\langle M, w\rangle)$ has the form $\langle M', w'\rangle$ for some TM M' and string w'.
- We want

$$\boldsymbol{\langle M, w\rangle \in A_{TM} \quad \textbf{iff} \quad \langle M', w'\rangle \in HALT.}$$

- Substituting definitions:

$$\boldsymbol{M \text{ accepts } w \quad \textbf{iff} \quad M' \text{ halts on } w'.}$$

- How might we design M' and w'?

$A_{TM} \leq_M HALT$

- We need to choose a TM/string pair M' and w' such that M' halts on w' iff M accepts w.
- Repeated idea: Construct M' and w' such that running M' on w' simulates M on w and bases its decision on what happens.
- One option:

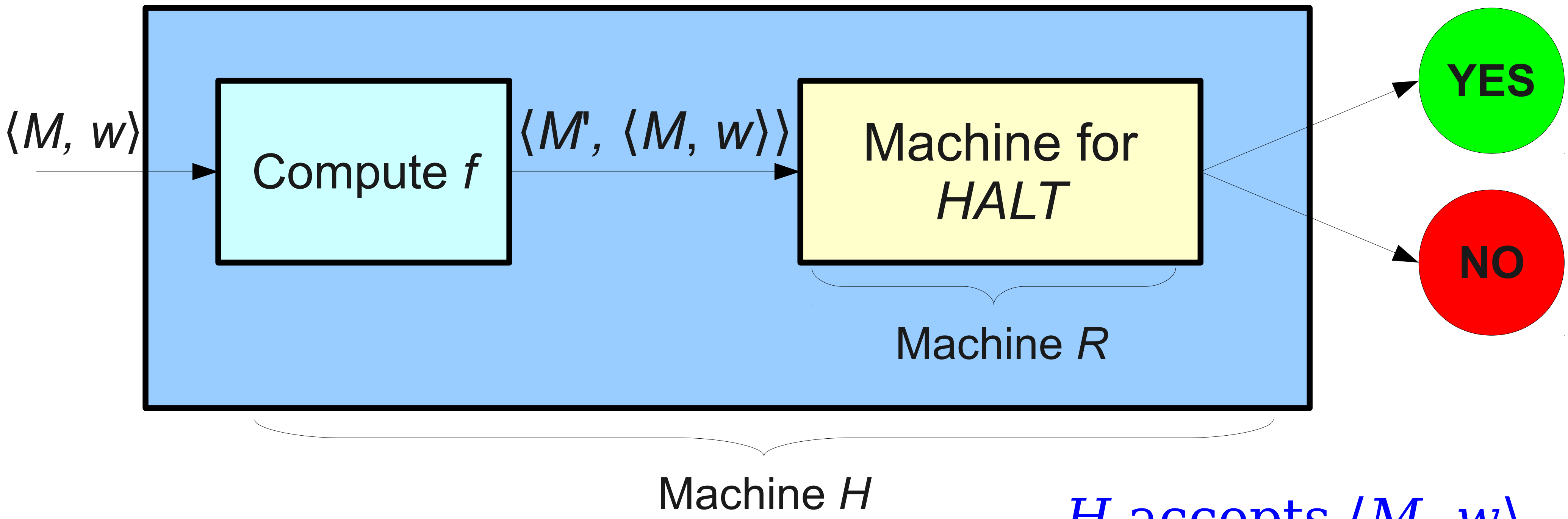
M' = "On input $\langle N, z\rangle$:

Run N on z.

If N accepts z, accept.

If N rejects z, loop infinitely."

$w' = \langle M, w\rangle$



M' = "On input $\langle N, z\rangle$:

Run N on z.

If N accepts, accept.

If N rejects, loop infinitely."

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

Compute $\langle M', \langle M, w\rangle\rangle$.

Run R on $\langle M', \langle M, w\rangle\rangle$.

If R accepts $\langle M', \langle M, w\rangle\rangle$, accept.

If R rejects $\langle M', \langle M, w\rangle\rangle$, reject."

H accepts $\langle M, w\rangle$

iff

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

iff

$\langle M', \langle M, w\rangle\rangle \in HALT$

iff

M' halts on $\langle M, w\rangle$

iff

M accepts w

iff

$\langle M, w\rangle \in A_{TM}$

An Important Detail

- In the course of this reduction, we construct a new machine *M*'.
- We never actually run the machine *M*'! That might loop forever.
- We instead just build a description of that machine and fed it into our machine for *HALT*.
- The answer given back by this machine about what *M*' *would do if we were to run it* can then be used to solve A_{TM}.

Theorem: $A_{TM} \leq_M HALT$.

Proof: We exhibit a mapping reduction from A_{TM} to $HALT$.

Let M' be the following TM:

M' = "On input $\langle N, z\rangle$:
Run N on z.
If N accepts, accept.
If N rejects, loop infinitely."

Then let $f(\langle M, w\rangle) = \langle M', \langle M, w\rangle\rangle$. We claim that f is computable and omit the details from this proof. We further claim that $\langle M, w\rangle \in A_{TM}$ iff $f(\langle M, w\rangle) \in HALT$. To see this, note that $f(\langle M, w\rangle) = \langle M', \langle M, w\rangle\rangle \in HALT$ iff M' halts on $\langle M, w\rangle$. By construction, M' halts on $\langle M, w\rangle$ iff M accepts w. Finally, M accepts w iff $\langle M, w\rangle \in A_{TM}$. Thus we have that $\langle M, w\rangle \in A_{TM}$ iff $f(\langle M, w\rangle) \in HALT$. Therefore, f is a mapping reduction from A_{TM} to $HALT$, so $A_{TM} \leq_M HALT$. ■

A Note on Directionality

Note the Direction

- To show that a language A is **RE**, reduce it to something that is known to be **RE**:

$$A \leq_{\mathrm{M}} \textit{some-}\boldsymbol{RE}\textit{-problem}$$

- To show that a language A is *not* **R**, reduce a problem that is known not to be **R** to A:

$$\textit{some-non-}\boldsymbol{R}\textit{-problem} \leq_{\mathrm{M}} A$$

- **The single most common mistake with reductions is doing the reduction in the wrong direction.**

Next Time

- **co-RE and Beyond**
  - What lies outside of **RE**? How much of it can be solved by computers?
- **More Reductions**
  - More examples of mapping reductions.