co-**RE** and Reducibility

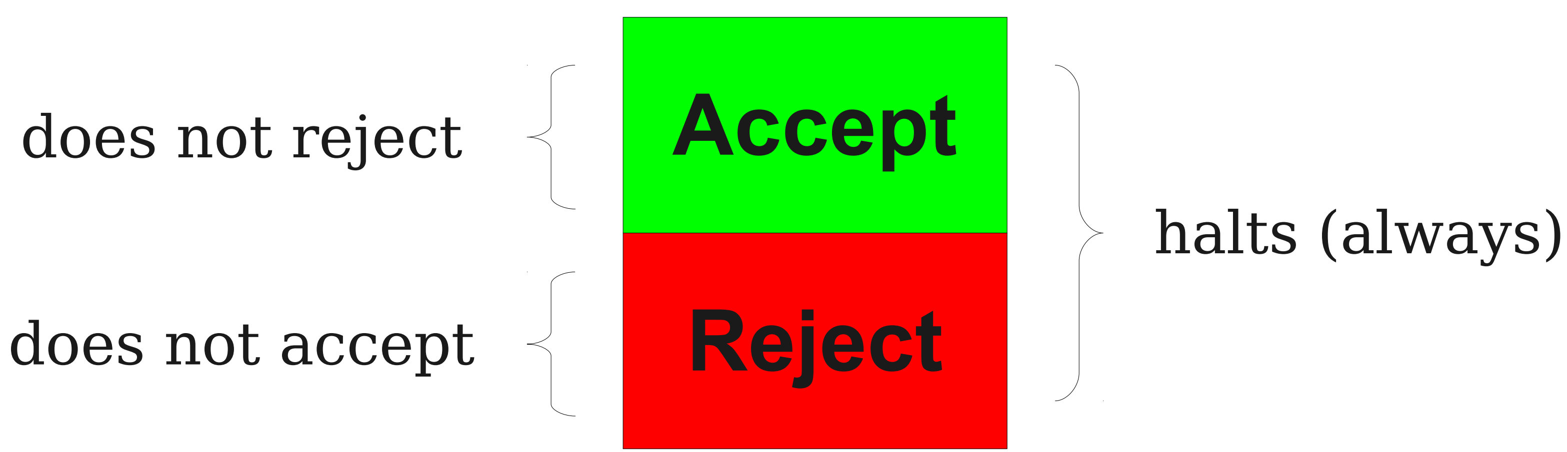
Friday Four Square!
Today at 4:15PM, Outside Gates

Announcements

- Problem Set 6 graded, will be returned at end of lecture.
  - Late submissions will be graded by Monday.
- Problem Set 7 due this Monday, March 4 at the start of lecture.
  - We are working on shuffling around OH for this weekend; we'll send out an email with updates.

Major Ideas from Last Time

- Some Turing machines always halt; they never go into an infinite loop.
- Turing machines of this sort are called **deciders**.
- For deciders, accepting is the same as not rejecting and rejecting is the same as not accepting.



Major Ideas from Last Time

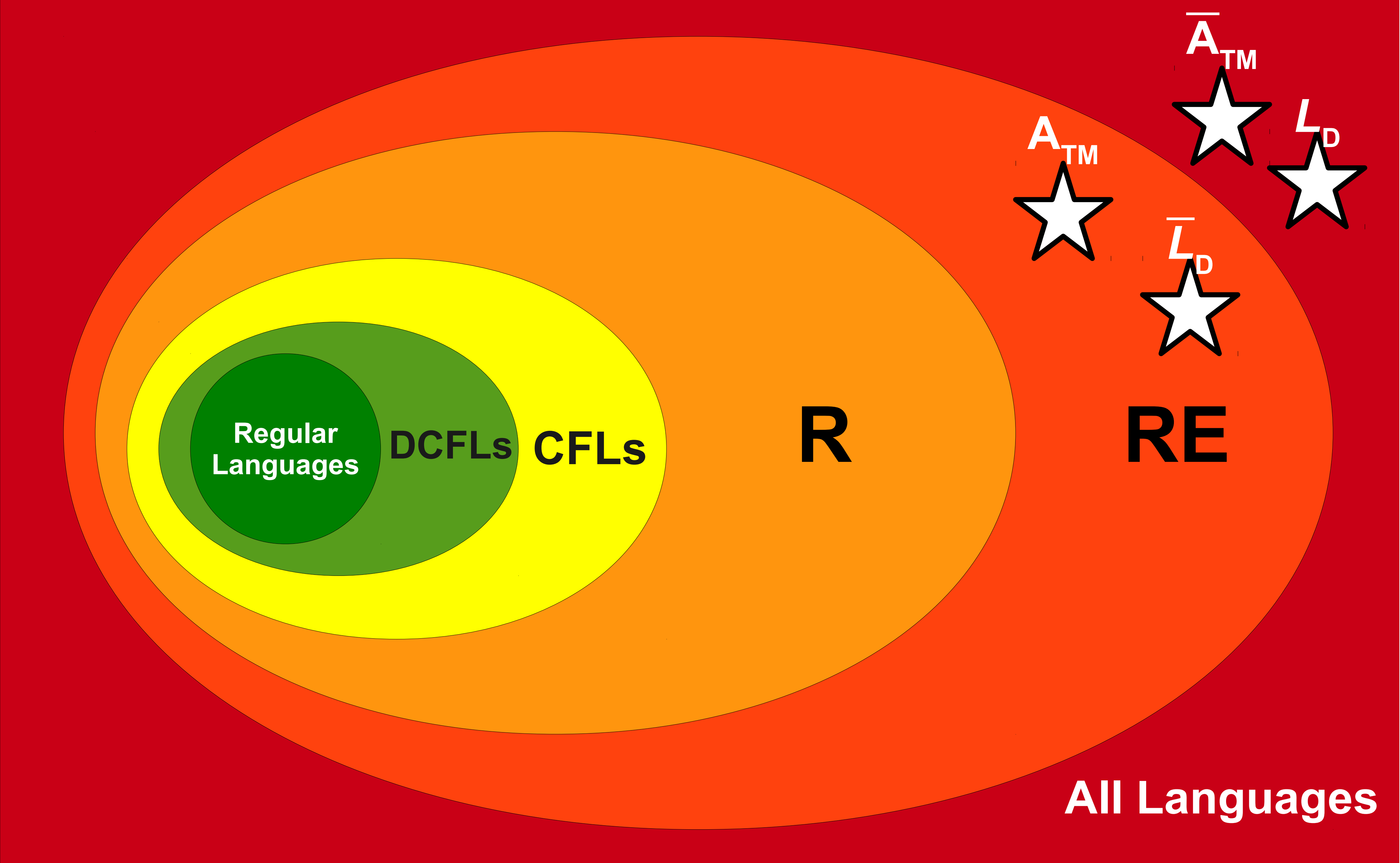
- A language L is called **decidable** iff there is a decider M such that $\mathscr{L}(M) = L$.
- Given a decider M, you *can* learn whether or not a string $w \in \mathscr{L}(M)$.
  - Run M on w.
  - Although it might take a staggeringly long time, M will eventually accept or reject w.
- The set $\mathbf{R}$ is the set of all decidable languages.

$$L \in \mathbf{R} \quad \text{iff} \quad L \text{ is decidable}$$

R and **RE** Languages

- Intuitively, a language is in **RE** if there is some way that you could exhaustively search for a proof that $w \in L$.
  - If you find it, accept!
  - If you don't find one, keep looking!
- Intuitively, a language is in **R** if there is a concrete algorithm that can determine whether $w \in L$.
  - It tends to be *much* harder to show that a language is in **R** than in **RE**.

The Limits of Computability



Outline for Today

- **The Halting Problem**
  - An important problem about TMs.
- **co-RE Languages**
  - Resolving a fundamental asymmetry.
- **Mapping Reductions**
  - A tool for finding unsolvable problems.

The Halting Problem

The Halting Problem

- The **halting problem** is the following problem:

Given a TM *M* and string *w*,
does *M* halt on *w*?

- Note that M doesn't have to *accept* w; it just has to *halt* on w.
- As a formal language:

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

- Is $HALT \in \mathbf{R}$? Is $HALT \in \mathbf{RE}$?

HALT is Recognizable

- Consider this Turing machine:

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

  Run M on w.

  If M accepts, accept.

  If M rejects, accept."
- Then H accepts $\langle M, w \rangle$ iff M halts on w.
- Thus $\mathscr{L}(H) = \mathit{HALT}$, so $\mathit{HALT} \in \mathbf{RE}$.

Theorem: $HALT \notin \mathbf{R}$.

(The halting problem is undecidable)

Proving $\mathit{HALT} \notin \mathbf{R}$

- Our proof will work as follows:
  - Suppose that $\mathit{HALT} \in \mathbf{R}$.
  - Using a decider for *HALT*, construct a decider for A_{TM}.
  - Reach a contradiction, since there is no decider for A_{TM} ($A_{TM} \notin \mathbf{R}$).
  - Conclude, therefore, that $\mathit{HALT} \notin \mathbf{R}$.

Accepting, Rejecting, and Looping

- Suppose we have a TM M and a string w.
- Then M either
  - **Accepts**, or
  - **Does not accept** (by rejecting or looping).
- What if M never rejects?
- Then M either
  - **Accepts**, or
  - **Does not accept** (by looping).

The Key Insight

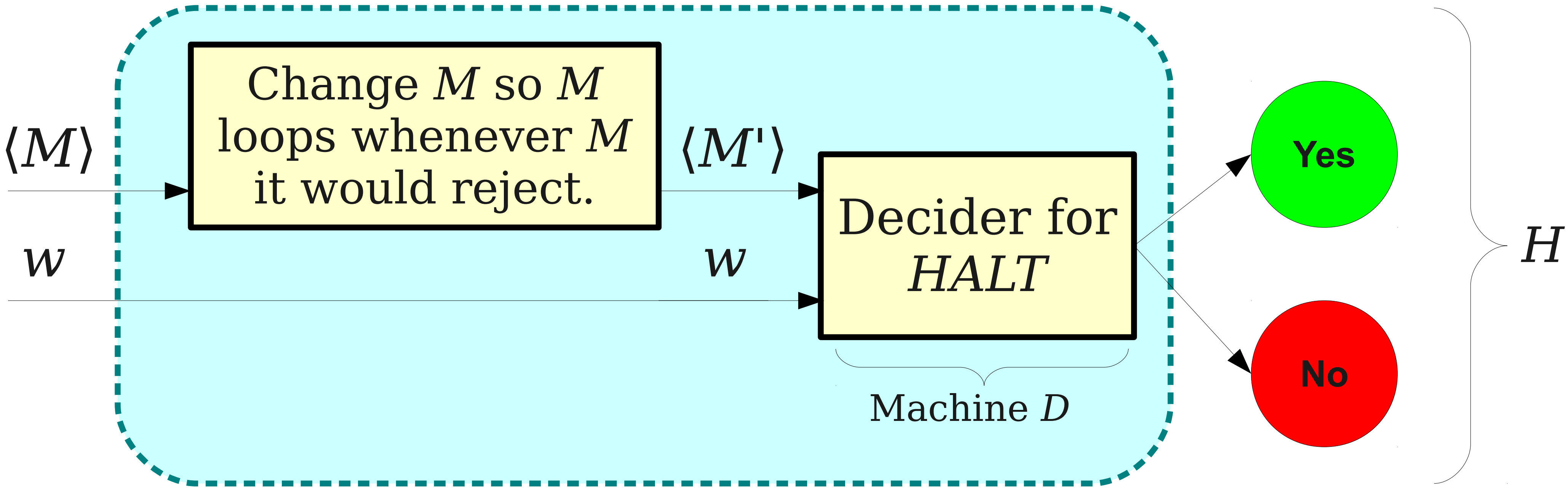
- If M never rejects, then

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

- In other words, if M never rejects, then

  $$\boldsymbol{\langle M, w\rangle \in \mathrm{A}_{\mathrm{TM}} \quad \text{iff} \quad \langle M, w\rangle \in HALT}$$

- If we can modify an arbitrary TM M so that M never rejects, then a decider for *HALT* can be made to decide A_{TM}.
  - Since $\mathrm{A}_{\mathrm{TM}} \notin \mathbf{R}$, this is a contradiction!



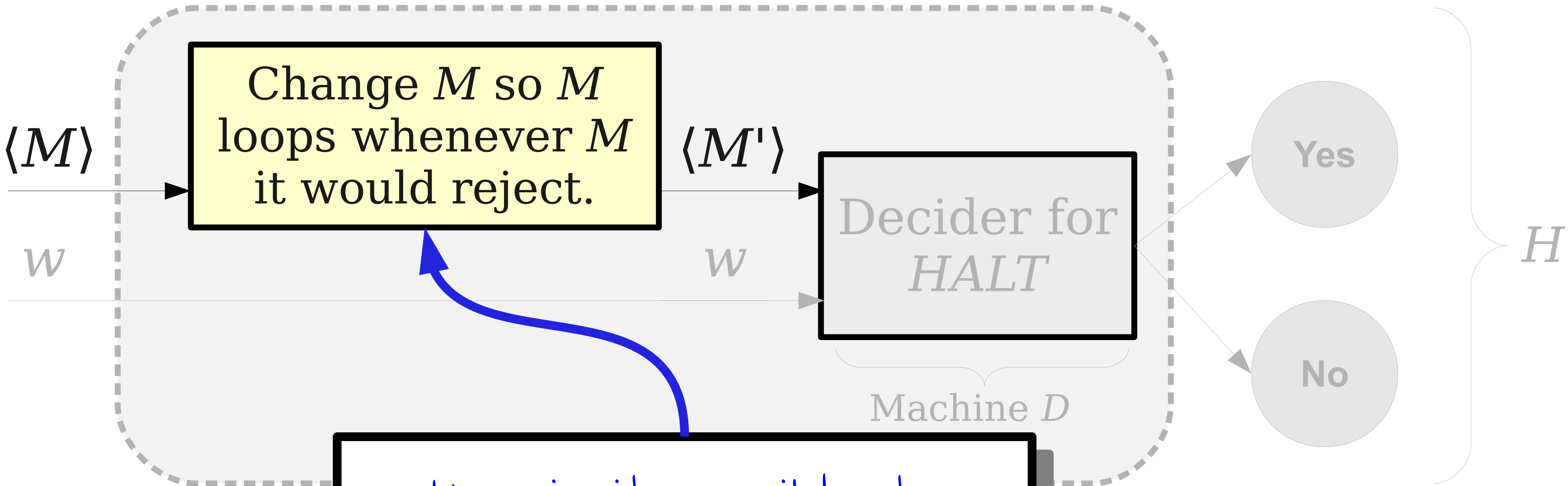
H = "On input $\langle M, w\rangle$:

- Transform M into M' by making M loop instead of rejecting.
- Run D on $\langle M', w\rangle$.
- If D accepts $\langle M', w\rangle$, then H accepts $\langle M, w\rangle$.
- If D rejects $\langle M', w\rangle$, then H rejects $\langle M, w\rangle$."

What happens if...

M accepts w?	**Accept**
M loops on w?	**Reject**
M rejects w?	**Reject**

Machine H is a decider for A_{TM}!



How is it possible to build this part of the machine?

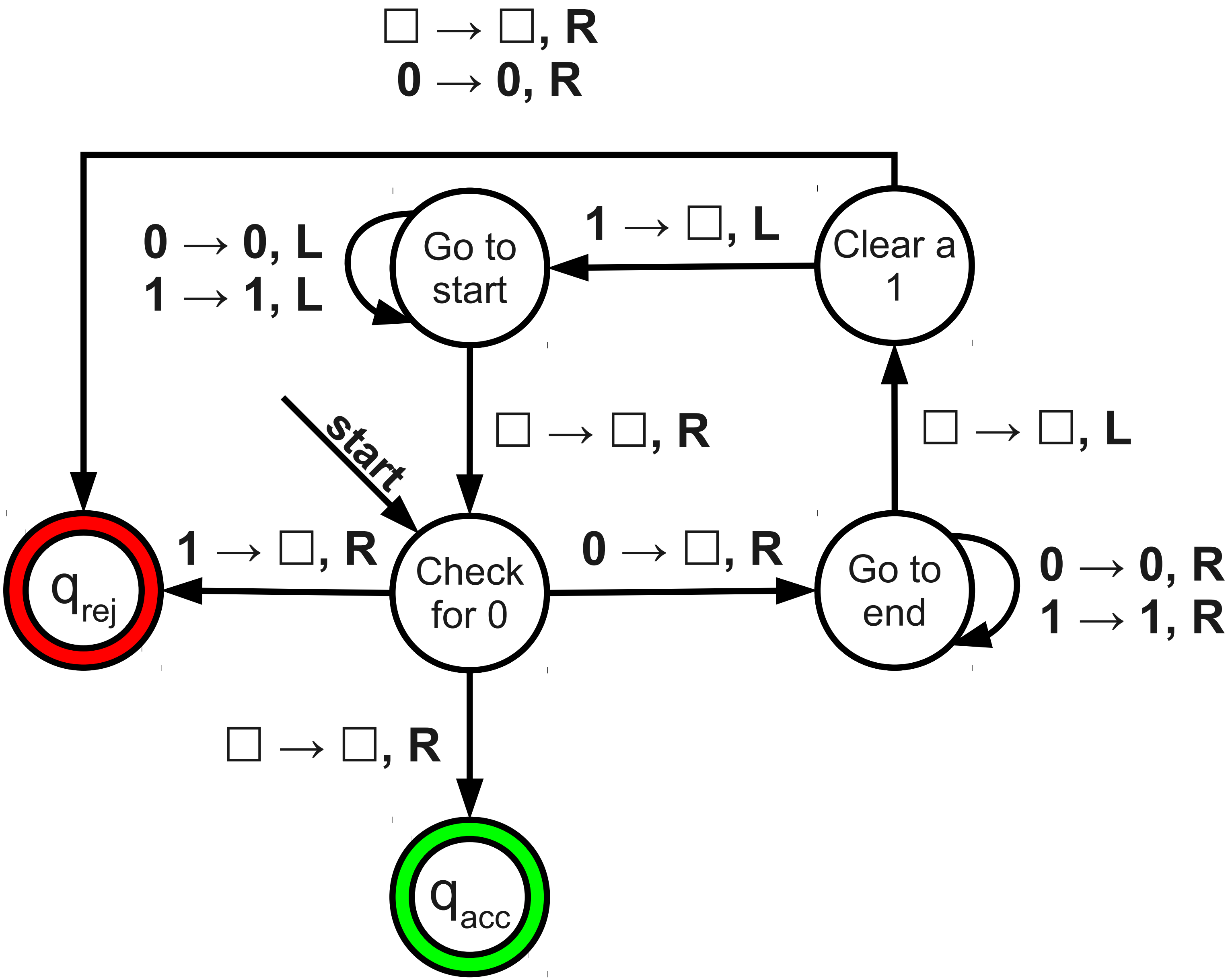
H = "On inpu

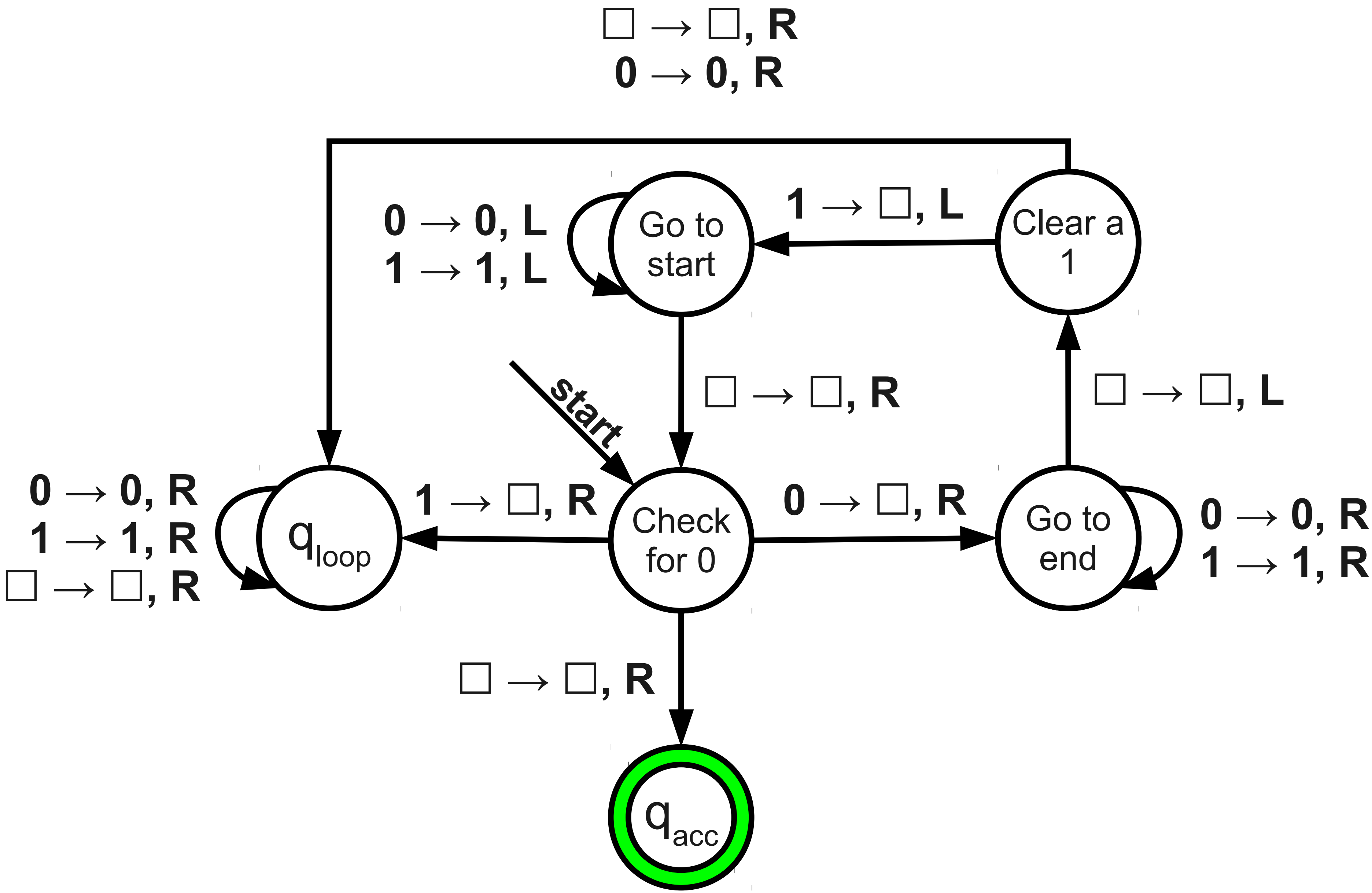
- Trans
  making
  rejecting.
- Run D on ⟨M', w⟩.
- If D accepts ⟨M', w⟩, then H accepts ⟨M, w⟩.
- If D rejects ⟨M', w⟩, then H rejects ⟨M, w⟩."

hat happens if...

ccepts w? **Accept**
M loops on w? **Reject**
M rejects w? **Reject**

Machine H is a decider for A_{TM}!


□ → □, R
0 → 0, R
0 → 0, L
1 → 1, L
Go to start
1 → □, L
Clear a 1
□ → □, R
start
□ → □, L
q_rej
1 → □, R
Check for 0
0 → □, R
Go to end
0 → 0, R
1 → 1, R
□ → □, R
q_acc


□ → □, R
0 → 0, R
Go to start
0 → 0, L
1 → 1, L
1 → □, L
Clear a 1
start
□ → □, R
□ → □, L
0 → 0, R
1 → 1, R
□ → □, R
q_{loop}
1 → □, R
Check for 0
0 → □, R
Go to end
0 → 0, R
1 → 1, R
□ → □, R
q_{acc}

Theorem: $HALT \notin \mathbf{R}$.

Proof: By contradiction; assume $HALT \in \mathbf{R}$. Then there must be some decider D for $HALT$. Consider the following TM H:

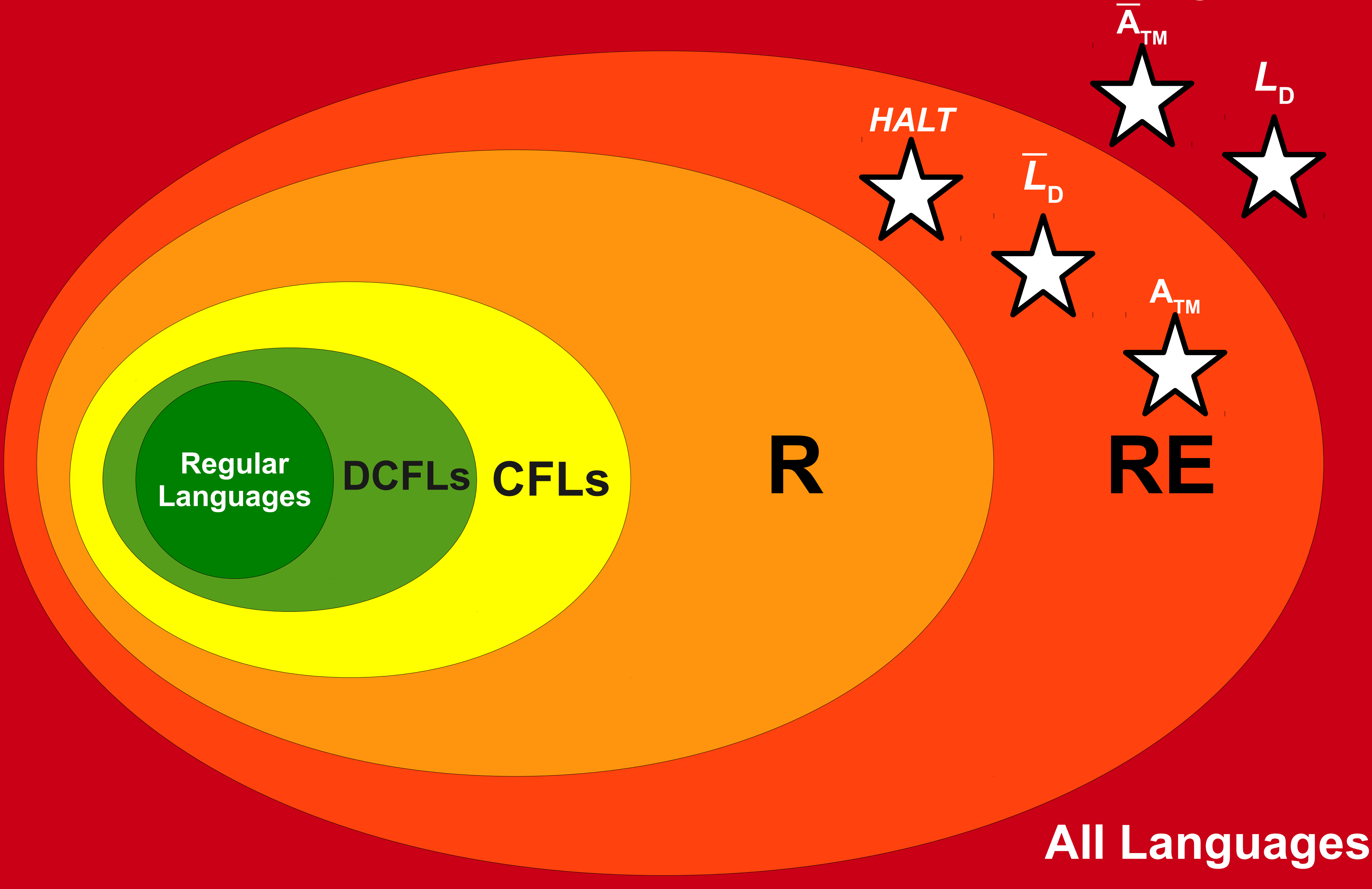
H = "On input $\langle M, w\rangle$, where M is a TM and w is a string:
Transform M into M' by making M' loop whenever M rejects.
Run D on $\langle M', w\rangle$.
If D accepts $\langle M', w\rangle$, then H accepts $\langle M, w\rangle$.
If D rejects $\langle M', w\rangle$, then H rejects $\langle M, w\rangle$."

We claim that H is a decider for A_{TM}. This means that $A_{TM} \in \mathbf{R}$, which contradicts the fact that $A_{TM} \notin \mathbf{R}$. This means our assumption was wrong, and so $HALT \notin \mathbf{R}$, as required.

First, we prove H is a decider. Note that on any input $\langle M, w\rangle$, H constructs the machine M' (which can be done in finite time), then runs D on $\langle M', w\rangle$. Since D is a decider, D always halts. Since H halts as soon as D halts, we know H halts on $\langle M, w\rangle$. Since our choice of $\langle M, w\rangle$ was arbitrary, this means that H halts on all inputs, so H is a decider.

Next, we prove that $\mathscr{L}(H) = A_{TM}$. To see this, note that H accepts $\langle M, w\rangle$ iff D accepts $\langle M', w\rangle$. Since D decides $HALT$, D accepts $\langle M', w\rangle$ iff M' halts on w. By construction, M' halts iff it accepts, so M' halts on w iff M' accepts w. Again by construction, M' accepts w iff M accepts w. Finally, M accepts w iff $\langle M, w\rangle \in A_{TM}$. Thus H accepts $\langle M, w\rangle$ iff $\langle M, w\rangle \in A_{TM}$, and so $\mathscr{L}(H) = A_{TM}$, as required. ■

The Limits of Computability

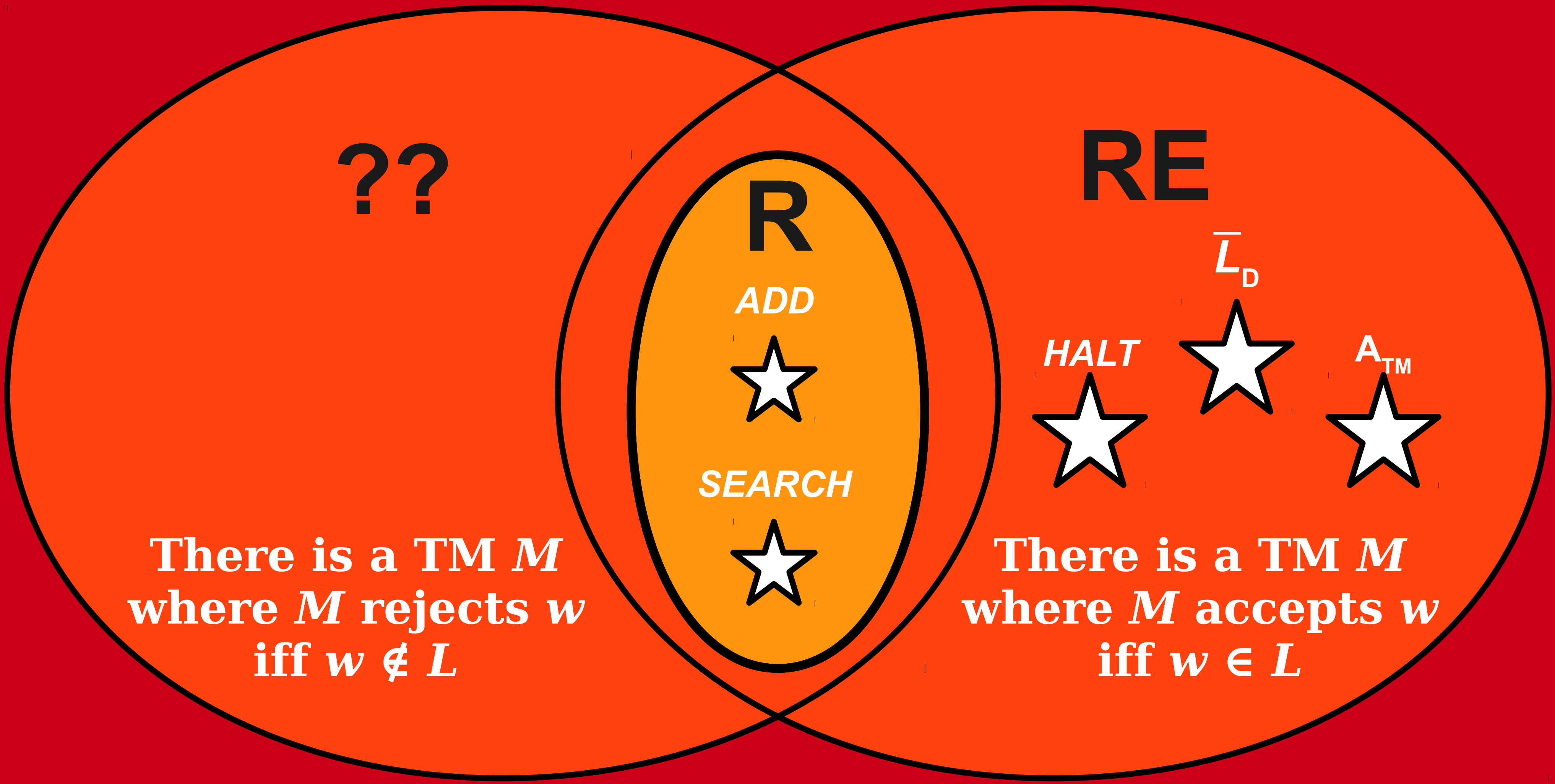


A_{TM} and *HALT*

- Both A_{TM} and *HALT* are undecidable.
  - There is no way to decide whether a TM will accept or eventually terminate.
- However, both A_{TM} and *HALT* are recognizable.
  - We can always run a TM on a string *w* and accept if that TM accepts or halts.
- **Intuition:** The only general way to learn what a TM will do on a given string is to run it and see what happens.

Resolving an Asymmetry

The Limits of Computability



A New Complexity Class

- A language L is in **RE** iff there is a TM M such that
  - if $w \in L$, then M accepts w.
  - if $w \notin L$, then M does not accept w.
- A TM M of this sort is called a *recognizer*, and L is called *recognizable*.
- A language L is in co-**RE** iff there is a TM M such that
  - if $w \in L$, then M does not reject w.
  - if $w \notin L$, then M rejects w.
- A TM M of this sort is called a ***co-recognizer***, and L is called ***co-recognizable***.

RE and co-**RE**

- Intuitively, **RE** consists of all problems where a TM can exhaustively search for proof that $w \in L$.
  - If $w \in L$, the TM will find the proof.
  - If $w \notin L$, the TM cannot find a proof.
- Intuitively, co-**RE** consists of all problems where a TM can exhaustively search for a disproof that $w \in L$.
  - If $w \in L$, the TM cannot find the disproof.
  - If $w \notin L$, the TM will find the disproof.

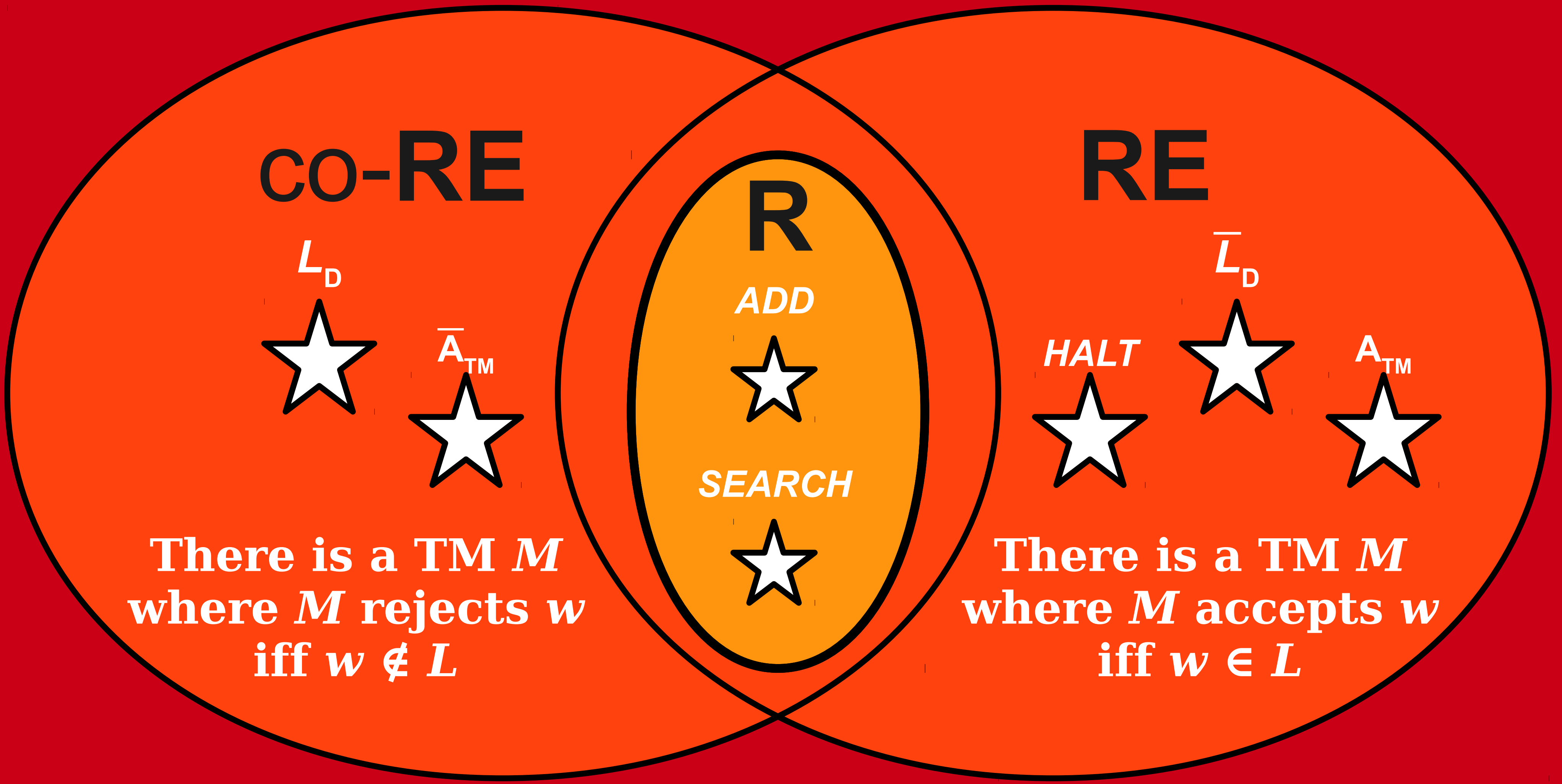
RE and co-**RE** Languages

- A_{TM} is an **RE** language:
  - Simulate the TM *M* on the string *w*.
  - If you find that *M* accepts *w*, accept.
  - If you find that *M* rejects *w*, reject.
  - (If *M* loops, we implicitly loop forever)
- $\overline{A}_{TM}$ is a co-**RE** language:
  - Simulate the TM *M* on the string *w*.
  - If you find that *M* accepts *w*, reject.
  - If you find that *M* rejects *w*, accept.
  - (If *M* loops, we implicitly loop forever)

RE and co-**RE** Languages

- $\overline{L}_{\mathrm{D}}$ is an **RE** language.
  - Simulate M on $\langle M \rangle$.
  - If you find that M accepts $\langle M \rangle$, accept.
  - If you find that M rejects $\langle M \rangle$, reject.
  - (If M loops, we implicitly loop forever)
- L_{D} is a co-**RE** language.
  - Simulate M on $\langle M \rangle$.
  - If you find that M accepts $\langle M \rangle$, reject.
  - If you find that M rejects $\langle M \rangle$, accept.
  - (If M loops, we implicitly loop forever)

The Limits of Computability



RE and co-**RE**

Theorem: $L \in$ **RE** iff $\overline{L} \in$ co-**RE**.

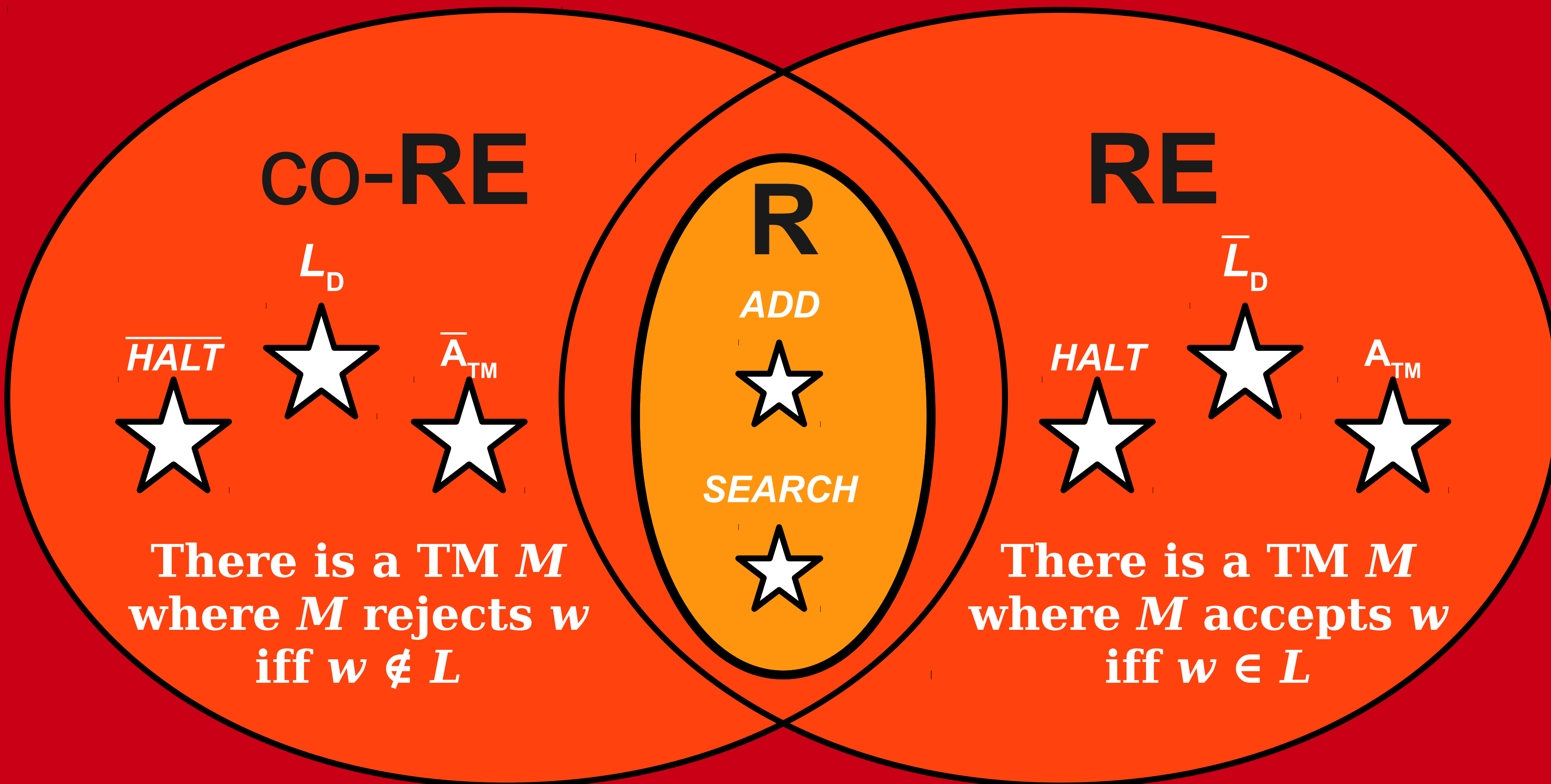
Proof Sketch: Start with a recognizer M for L. Then, flip its accepting and rejecting states to make machine M'. Then

M' rejects w
iff M accepts w
iff $w \in L$
iff $w \notin \overline{L}$.

M' does not reject w
iff M' accepts w or M' loops on w
iff M rejects w or M loops on w
iff $w \notin L$
iff $w \in \overline{L}$.

The same approach works if we flip the accept and reject states of a co-recognizer for $\overline{L}$. ■

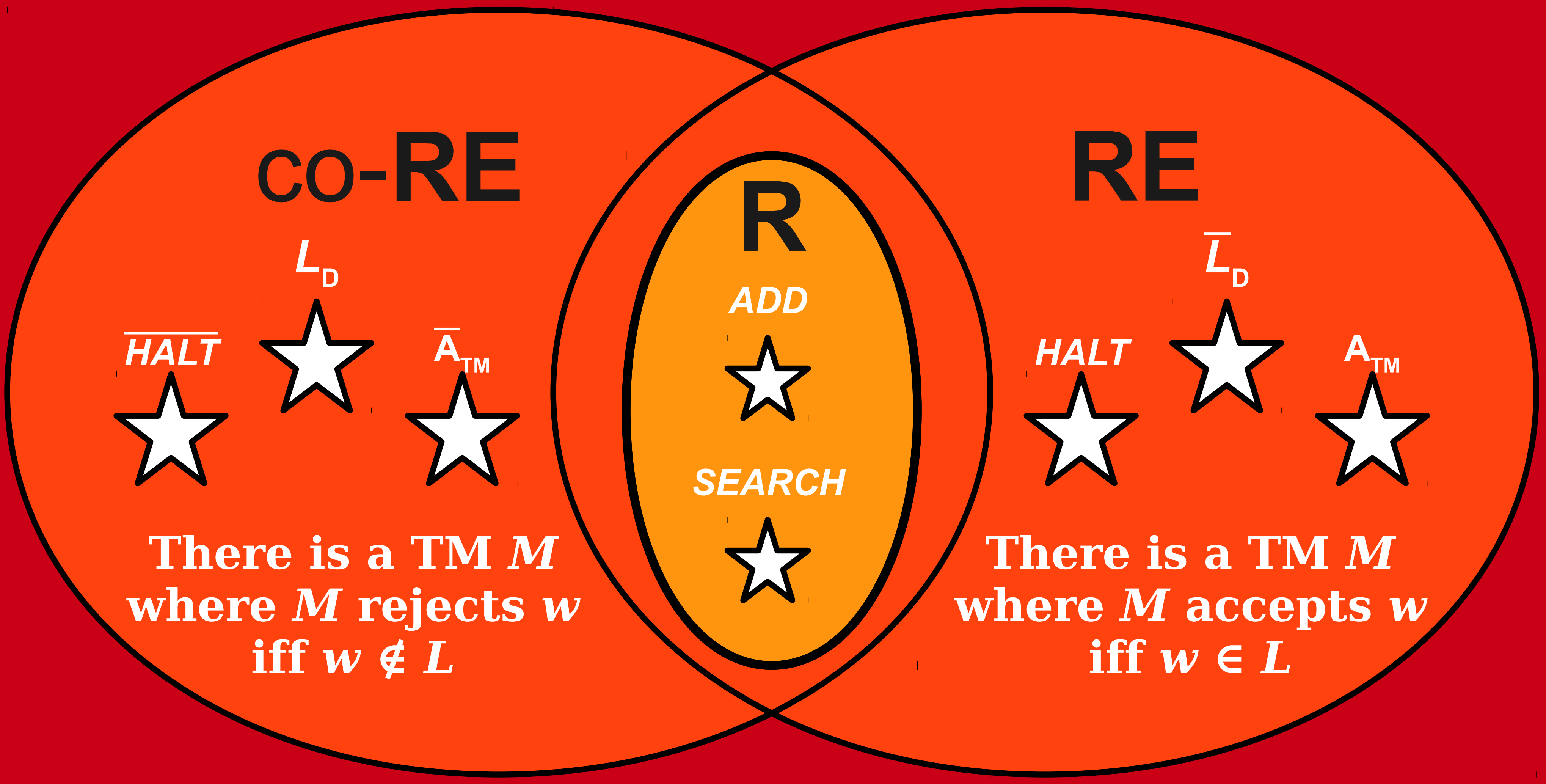
The Limits of Computability



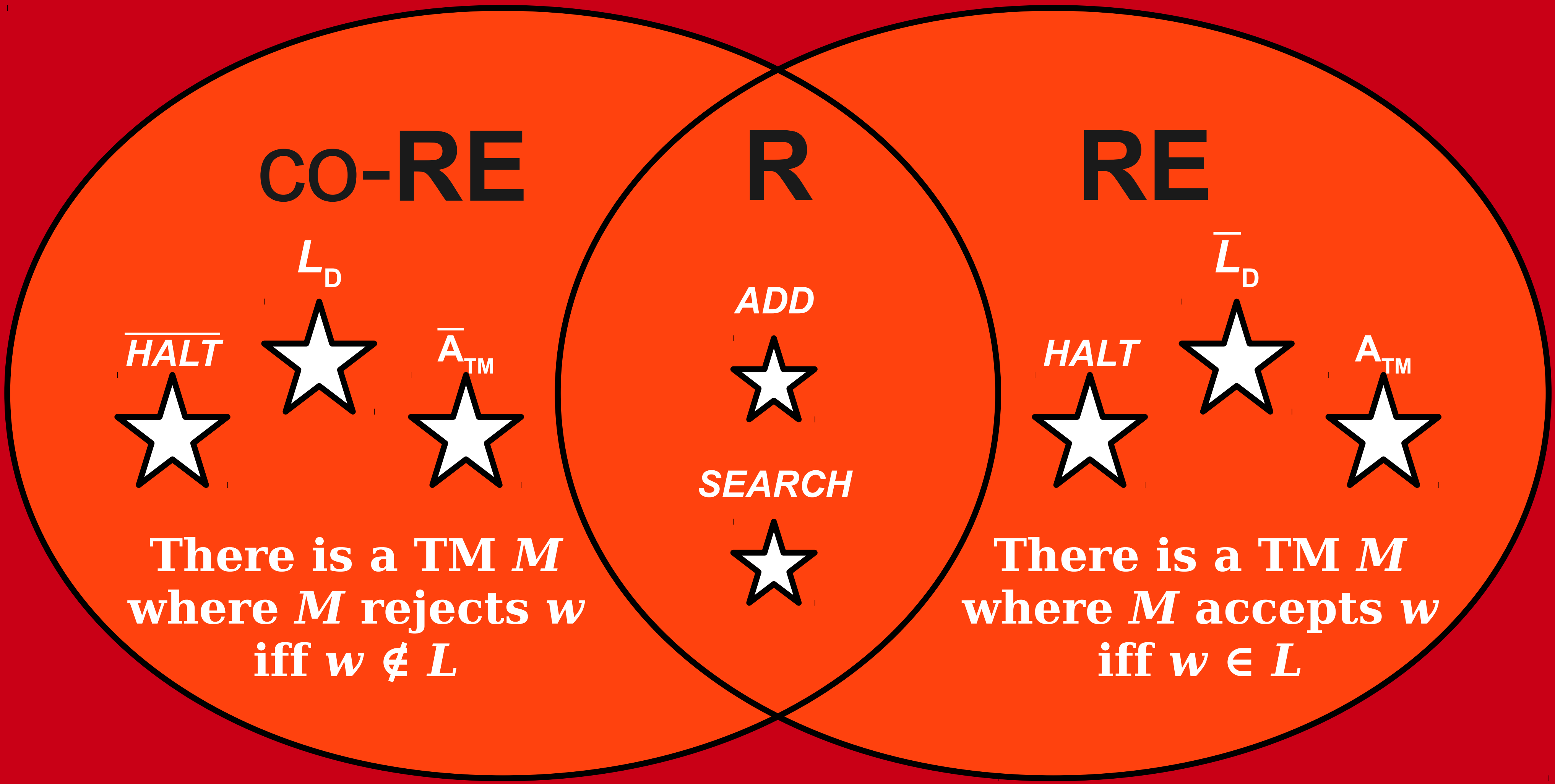
R, **RE**, and co-**RE**

- Every language in **R** is in both **RE** and co-**RE**.
- Why?
  - A decider for L accepts all $w \in L$ and rejects all $w \notin L$.
- In other words, $\mathbf{R} \subseteq \mathbf{RE} \cap \text{co-}\mathbf{RE}$.
- **Question:** Does $\mathbf{R} = \mathbf{RE} \cap \text{co-}\mathbf{RE}$?

Which Picture is Correct?



Which Picture is Correct?



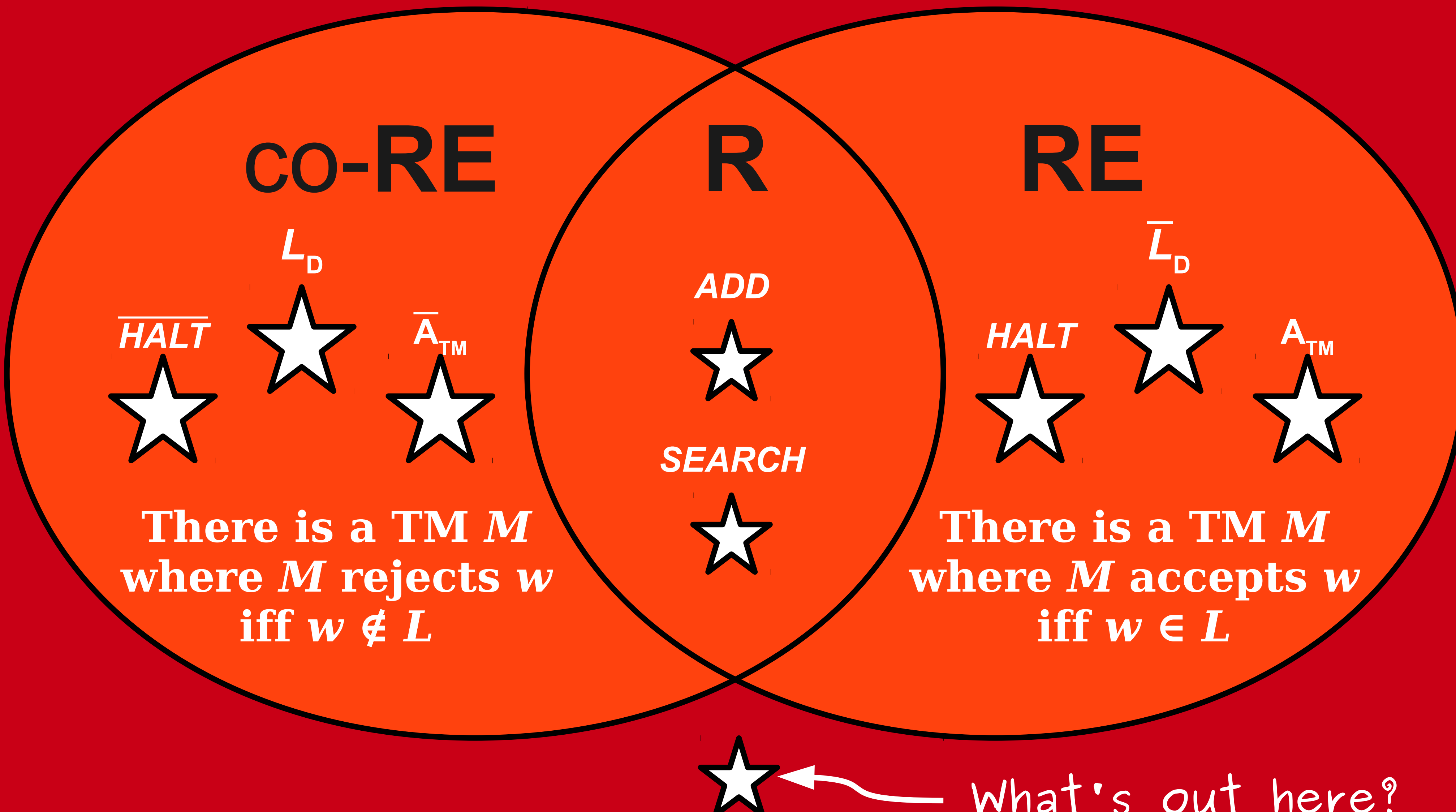
R, **RE**, and co-**RE**

- ***Theorem:*** If $L \in \mathbf{RE}$ and $L \in$ co-$\mathbf{RE}$, then $L \in \mathbf{R}$.
- ***Proof sketch:*** Since $L \in \mathbf{RE}$, there is a recognizer M for it. Since $L \in$ co-$\mathbf{RE}$, there is a co-recognizer $\overline{M}$ for it.

  This TM D is a decider for L:

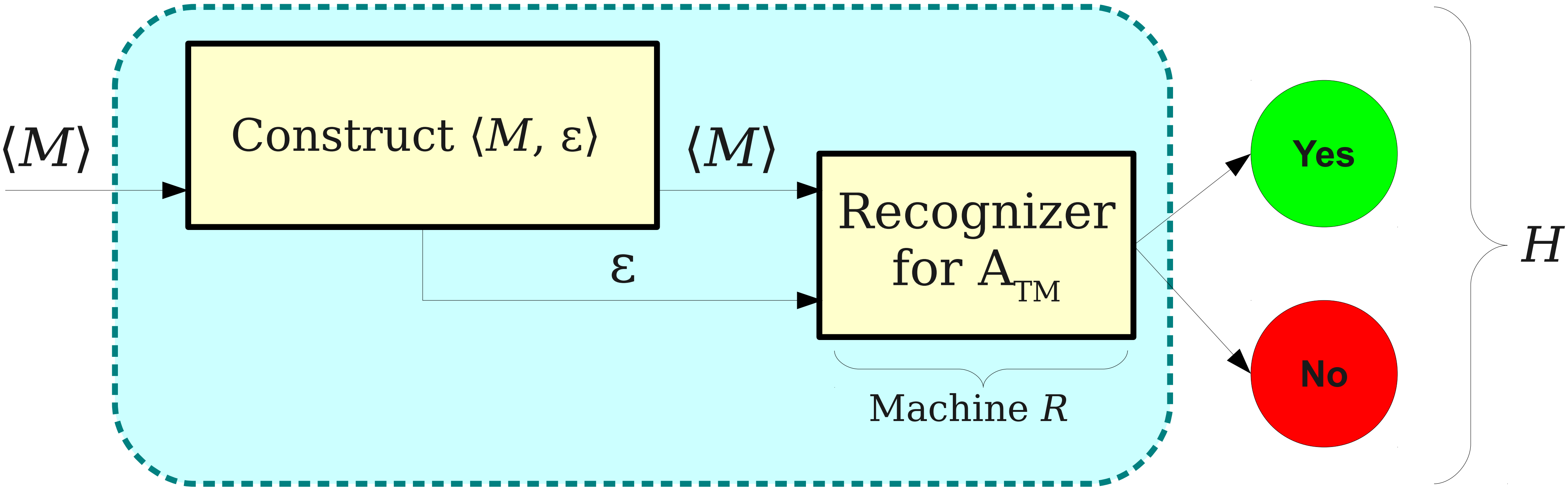
  > D = "On input w:
  > Run M on w and $\overline{M}$ on w in parallel.
  > If M accepts w, accept.
  > If $\overline{M}$ rejects w, reject.

The Limits of Computability



A Repeating Pattern

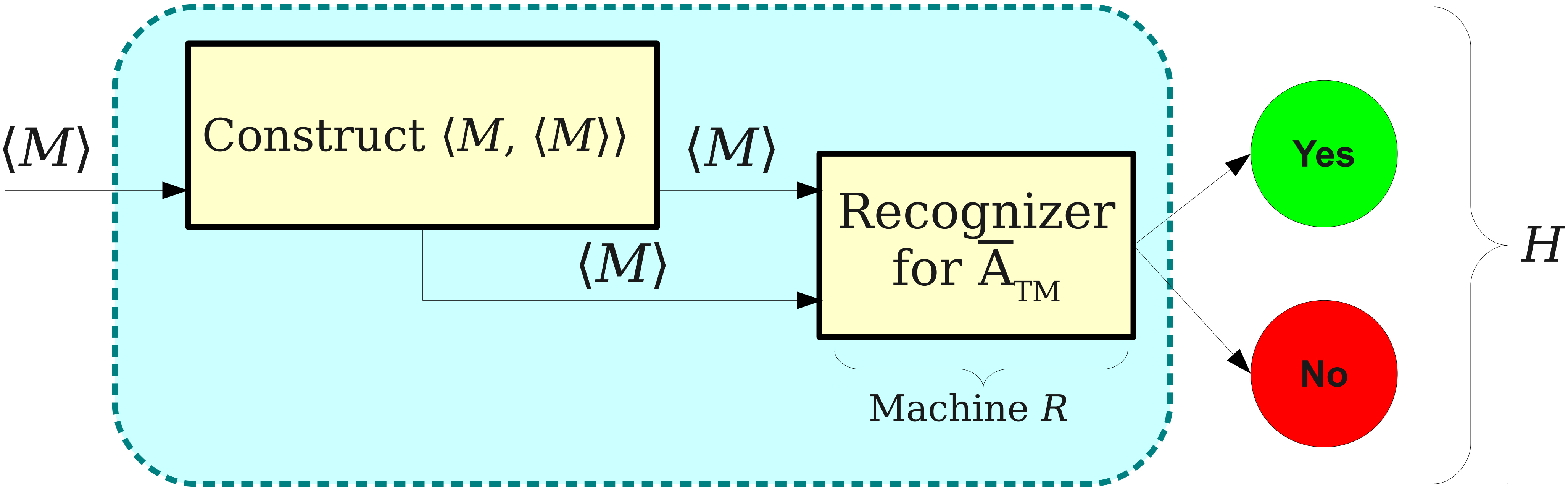
$L = \{ \langle M \rangle \mid M$ is a TM that accepts $\varepsilon \}$



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

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

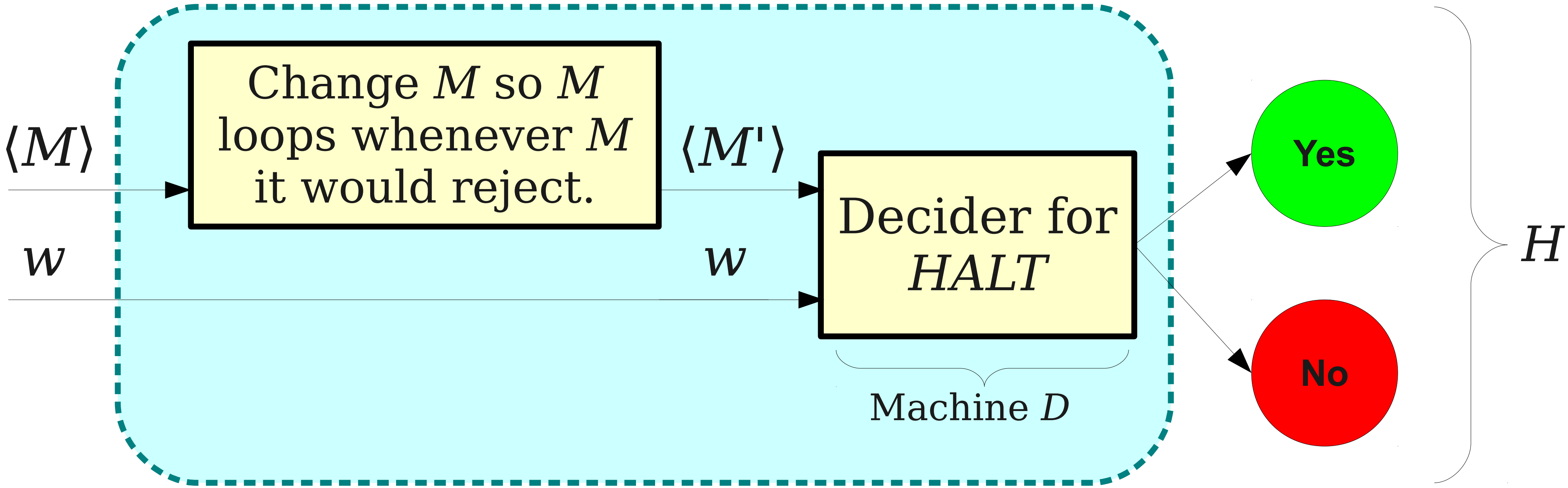
From $\overline{A}_{TM}$ to L_D



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 $\langle M, \langle M \rangle \rangle$.
- If R rejects $\langle M, \langle M \rangle \rangle$, then H rejects $\langle M, \langle M \rangle \rangle$."

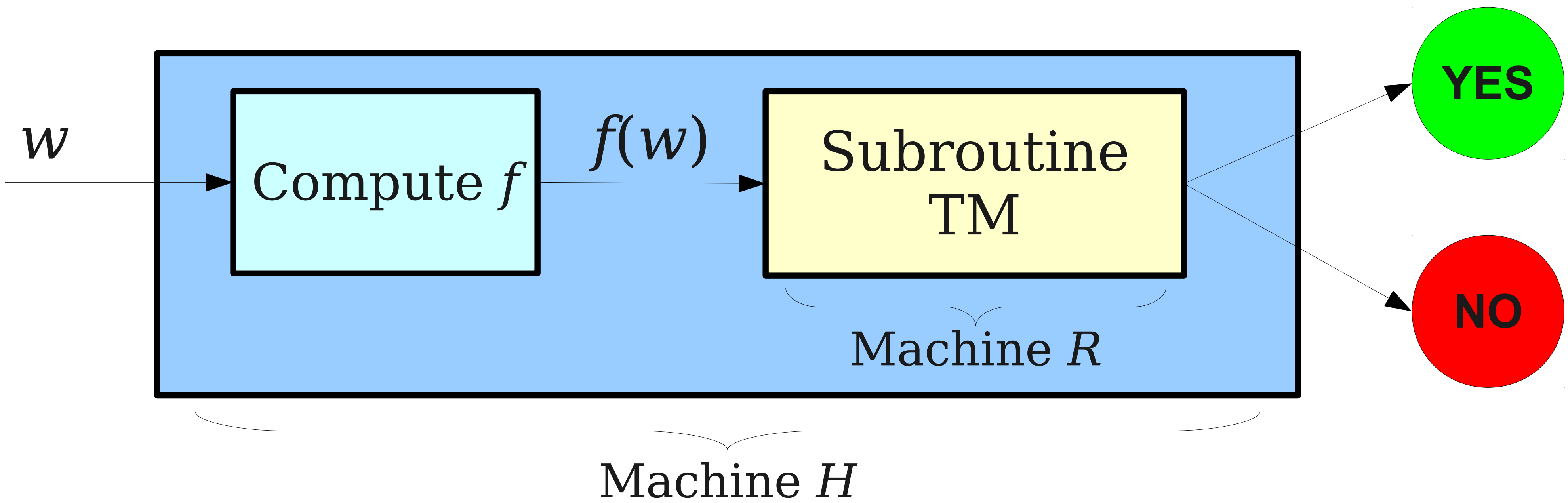
From *HALT* to A_{TM}



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

- Build M into M' so M' loops when M rejects.
- Run D on $\langle M', w\rangle$.
- If D accepts $\langle M', w\rangle$, then H accepts $\langle M, w\rangle$.
- If D rejects $\langle M', w\rangle$, then H rejects $\langle M, w\rangle$."

The General Pattern

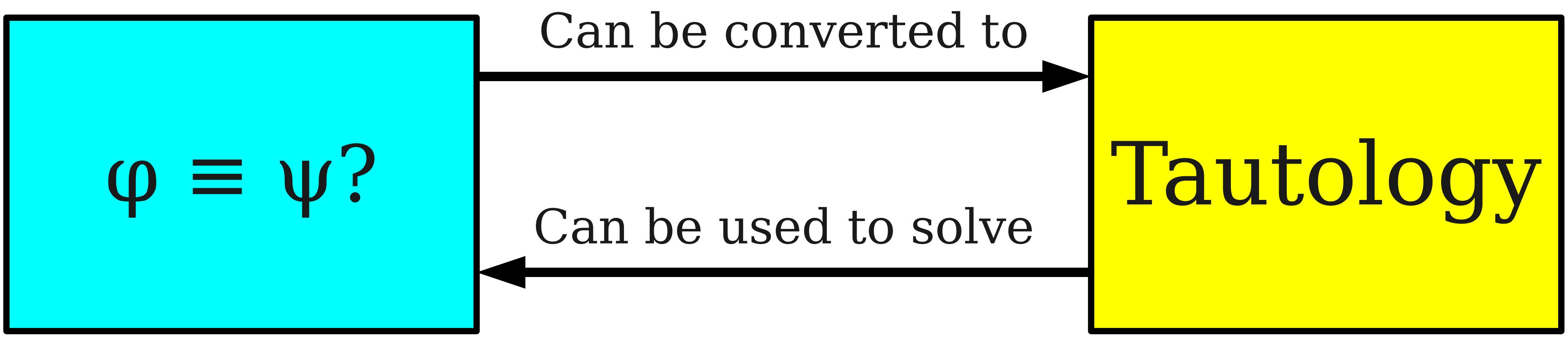


H = "On input w:

- 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."

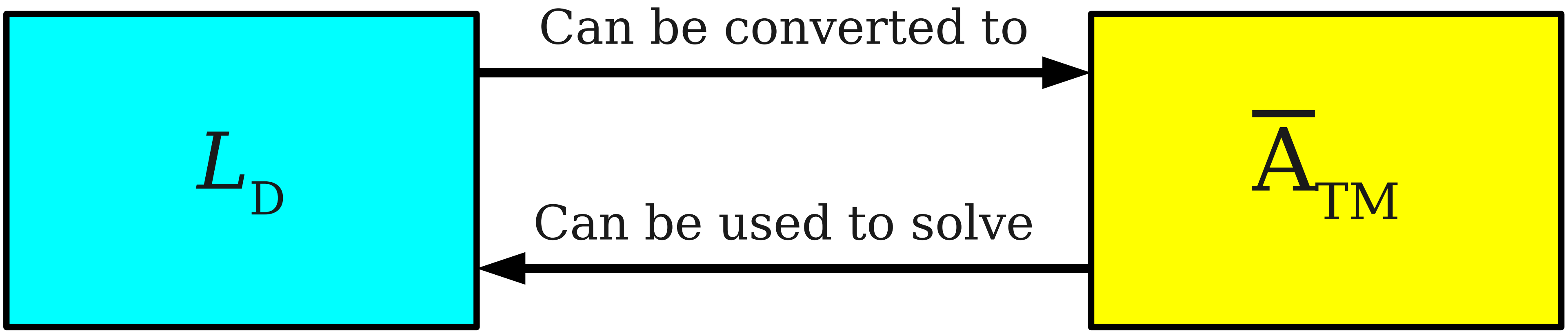
Reductions

- Intuitively, problem A **reduces** to problem B iff a solver for B can be used to solve problem A.



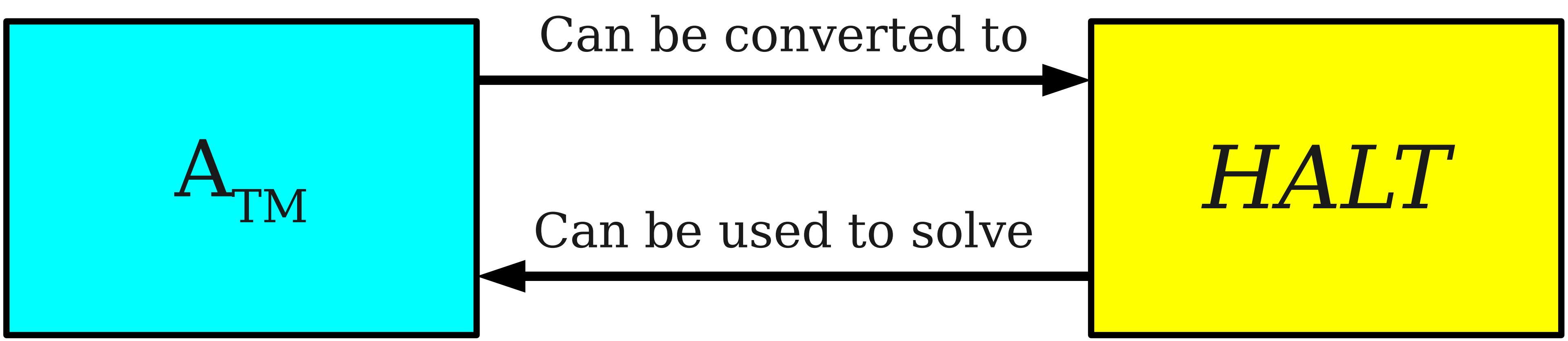
Reductions

- Intuitively, problem A **reduces** to problem B iff a solver for B can be used to solve problem A.



Reductions

- Intuitively, problem *A* **reduces** to problem *B* iff a solver for *B* can be used to solve problem *A*.



Reductions

- Intuitively, problem A **reduces** to problem B iff a solver for B can be used to solve problem A.
- Reductions can be used to show certain problems are "solvable:"

If *A* reduces to *B* and *B* is "solvable," then *A* is "solvable."

- Reductions can be used to show certain problems are "*unsolvable*:"

If *A* reduces to *B* and *A* is "unsolvable," then *B* is "unsolvable."

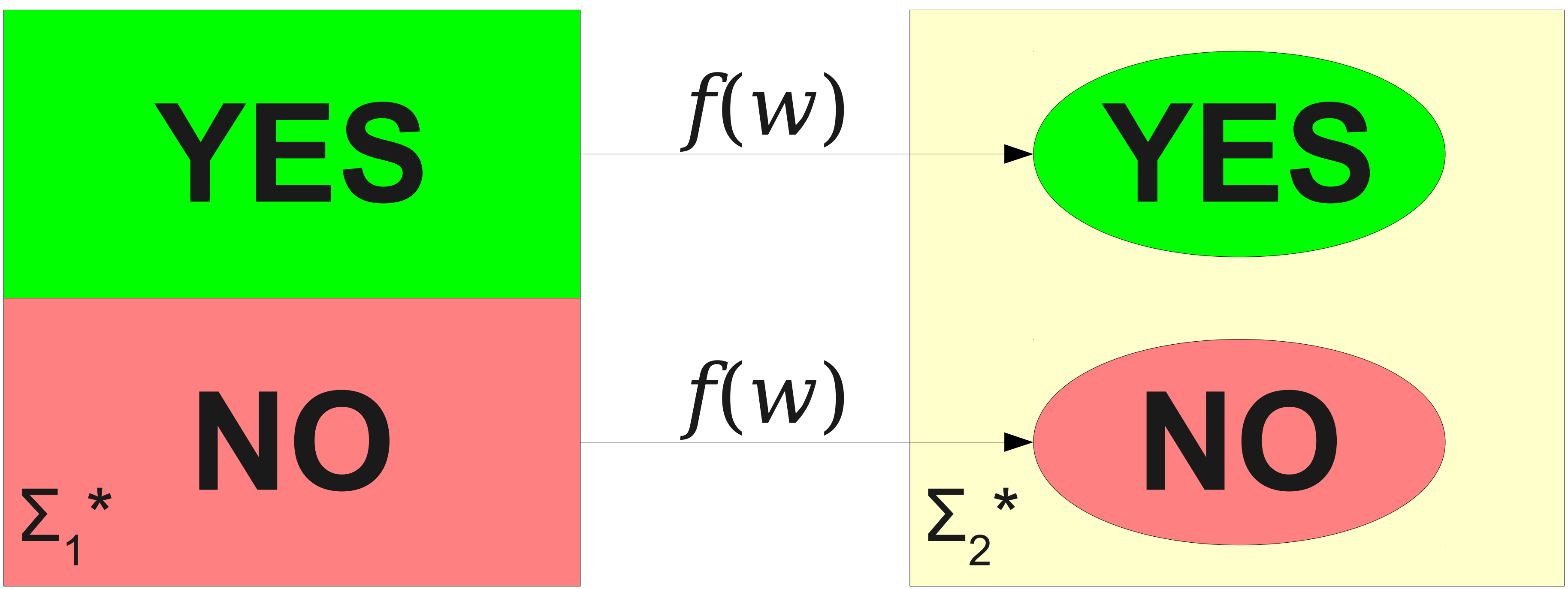
Formalizing Reductions

- In order to make the previous intuition more rigorous, we need to formally define reductions.
- There are many ways to do this; we'll explore two:
  - **Mapping reducibility** (today / Monday), and
  - **Polynomial-time reducibility** (next week).

Defining Reductions

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

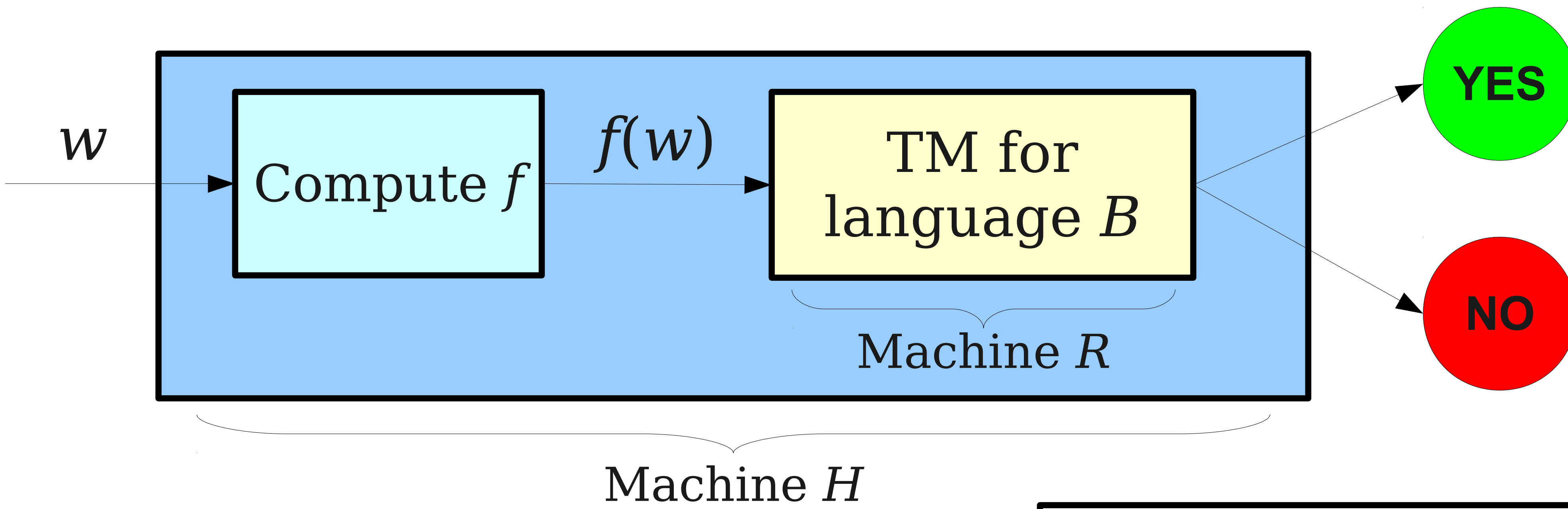
  For any $w \in \Sigma_1^*$, $w \in A$ iff $f(w) \in B$

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

Why Reductions Matter

- If language A reduces to language B, we can use a recognizer / co-recognizer / decider for B to recognize / co-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:

- 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 $f(w)$

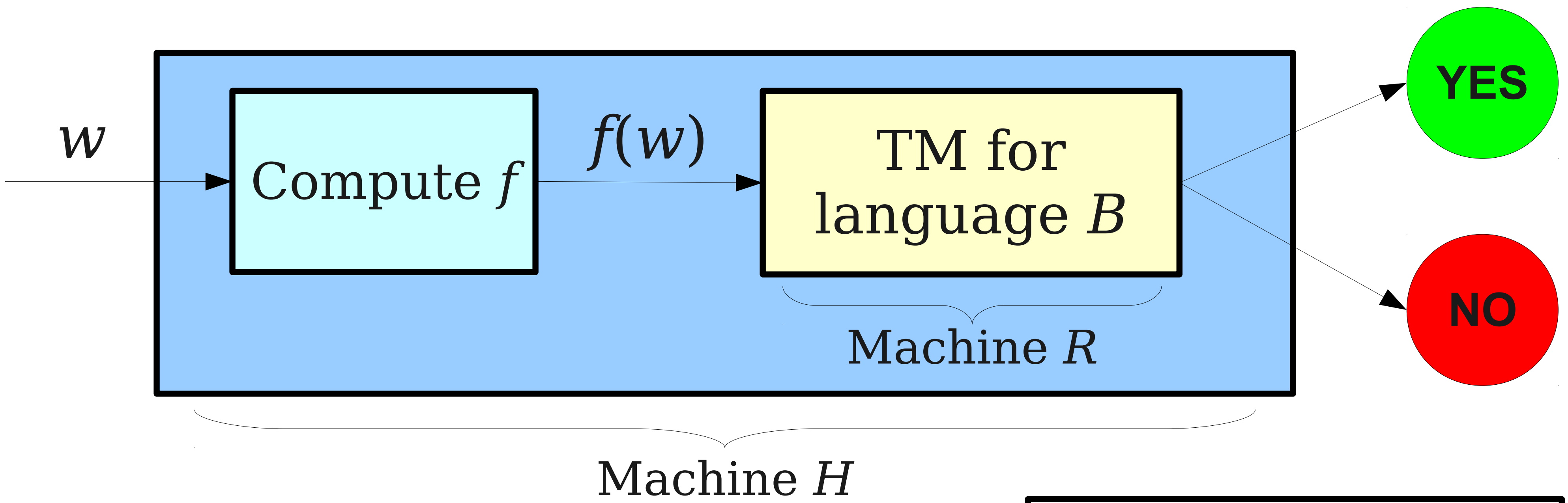
iff

$f(w) \in B$

iff

$w \in A$

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



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."

$$\mathscr{L}(H) = A$$

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 $w \in L_D$.

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:

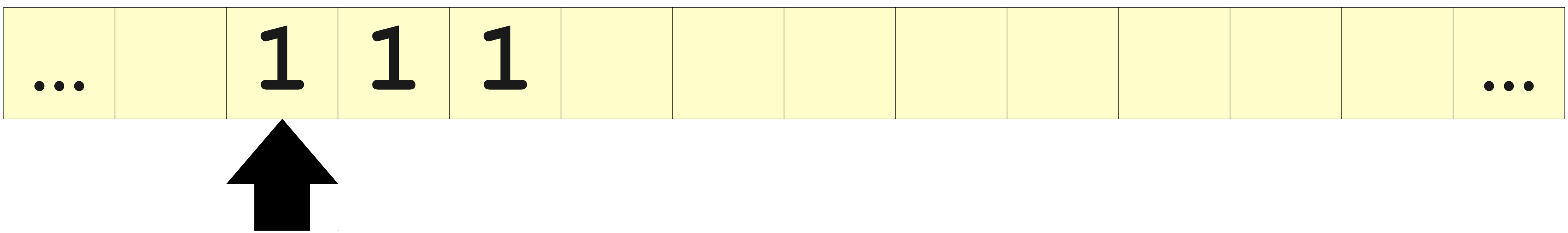
  Compute $f(w)$ and write it on the tape.

  Move the tape head to the start of $f(w)$.

  Halt."

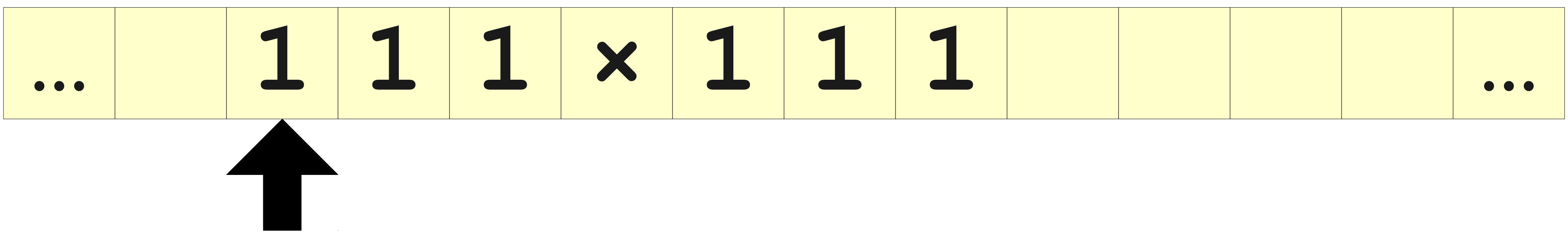
Computable Functions

$$f(\mathbf{1}^n) = \mathbf{1}^{3n+1}$$



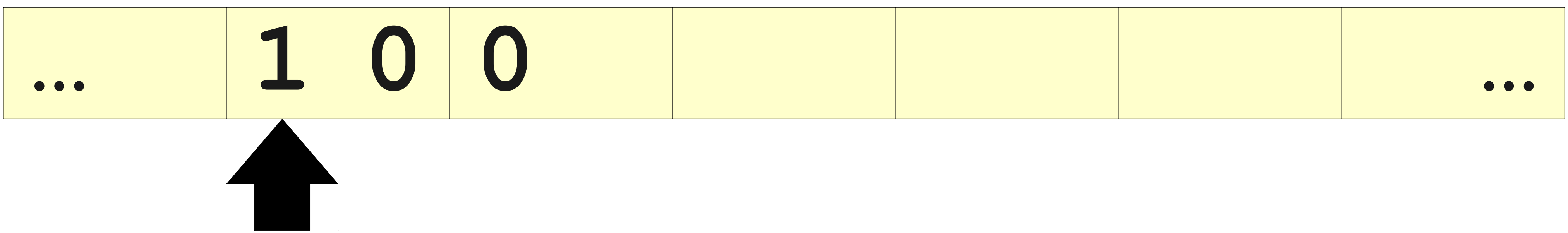
Computable Functions

$$f(w) = \begin{cases} 1^{mn} & \text{if } w = 1^n \times 1^m \\ \varepsilon & \text{otherwise} \end{cases}$$



Computable Functions

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



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.