Unsolvable Problems

Announcements

- Problem Set 5 graded, will be returned at end of lecture.
- Problem session tonight in 380-380X from 7PM – 7:50PM.
  - Optional, but highly recommended!
- CS Career Panel Tonight: 6PM in Gates 104.
  - Lots of cool people there!

Unsolvable Problems

Goals for Today

- Find concrete examples of problems that cannot be solved by computers.
- See how the procedure for finding languages that are not **R** or **RE** is fundamentally different from finding languages that are not regular or context-free.
- Set the stage for reductions and mapping reductions on Wednesday.

Recap from Friday

Major Ideas from Last Time

- Every TM can be converted into a string representation of itself.
  - The **encoding** of M is denoted $\langle M \rangle$.
- The **universal Turing machine** U_{TM} accepts an encoding $\langle M, w \rangle$ of a TM M and string w, then simulates the execution of M on w.
- The language of U_{TM} is the language $\mathbf{A_{TM}}$:

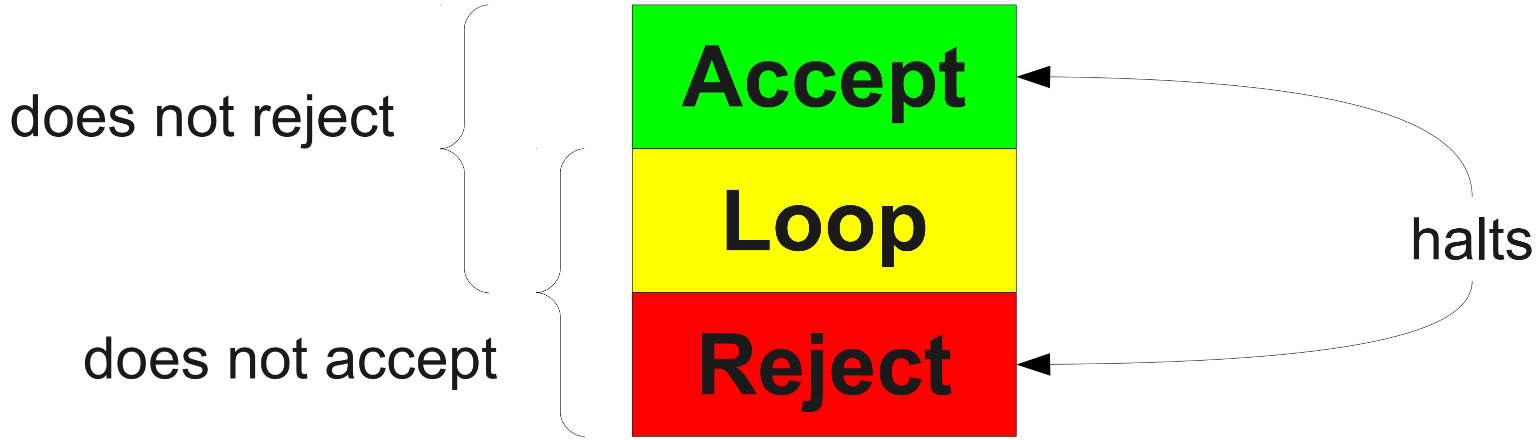
$$A_{TM} = \{ \langle M, w \rangle \mid M \text{ is a TM that accepts } w. \}$$

- Equivalently:

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

Major Ideas from Last Time

- A TM **accepts** a string *w* if it enters its accept state.
- A TM **rejects** a string *w* if it enters its reject state.
- A TM **loops** on a string *w* if neither of these happens.
- A TM **does not accept** a string *w* if it either rejects *w* or loops infinitely on *w*.
- A TM **does not reject** a string *w* if it either accepts *w* or loops infinitely on *w*.
- A TM **halts** if it accepts or rejects.



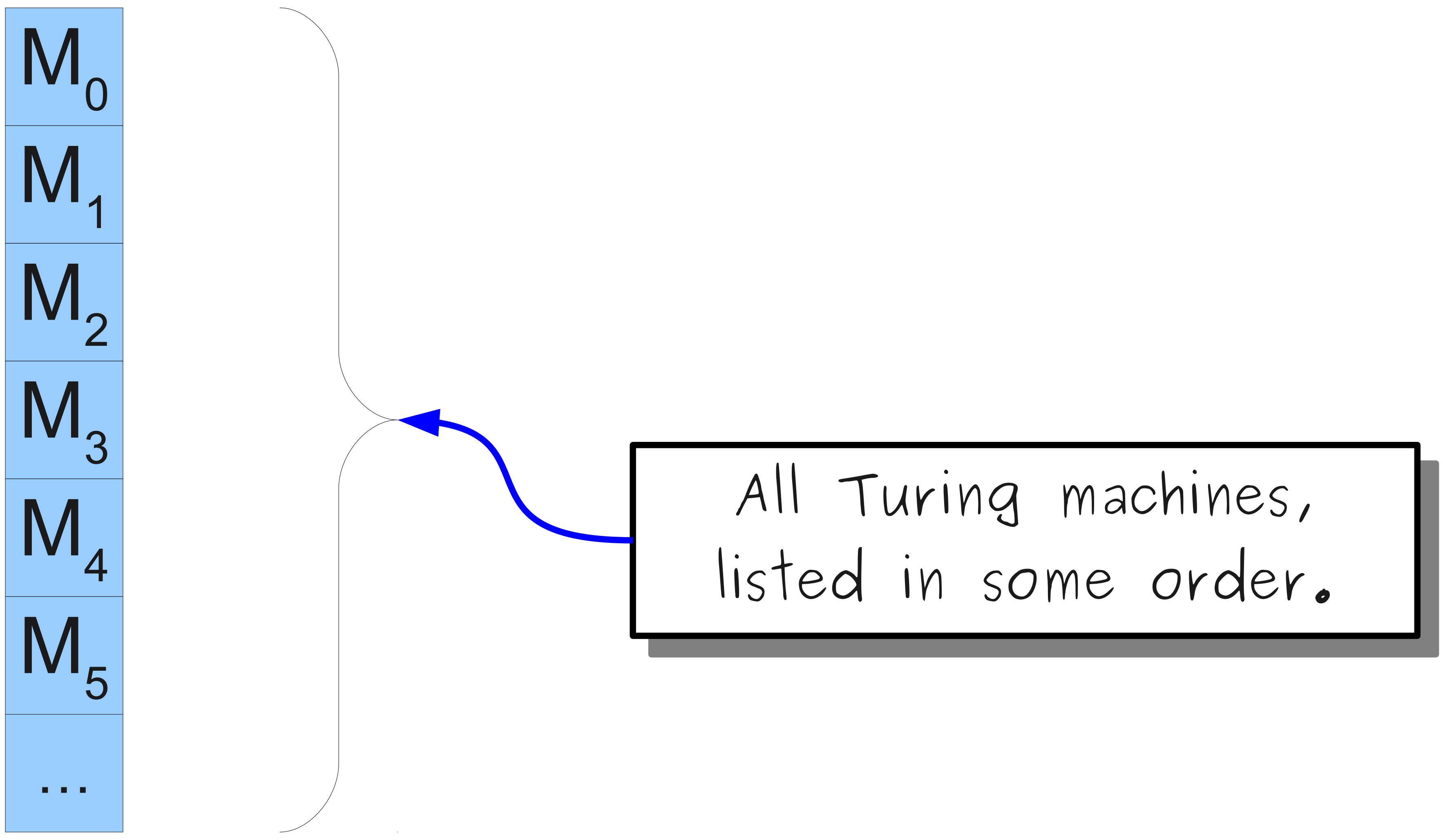
What happens when we run
a TM on a TM encoding?

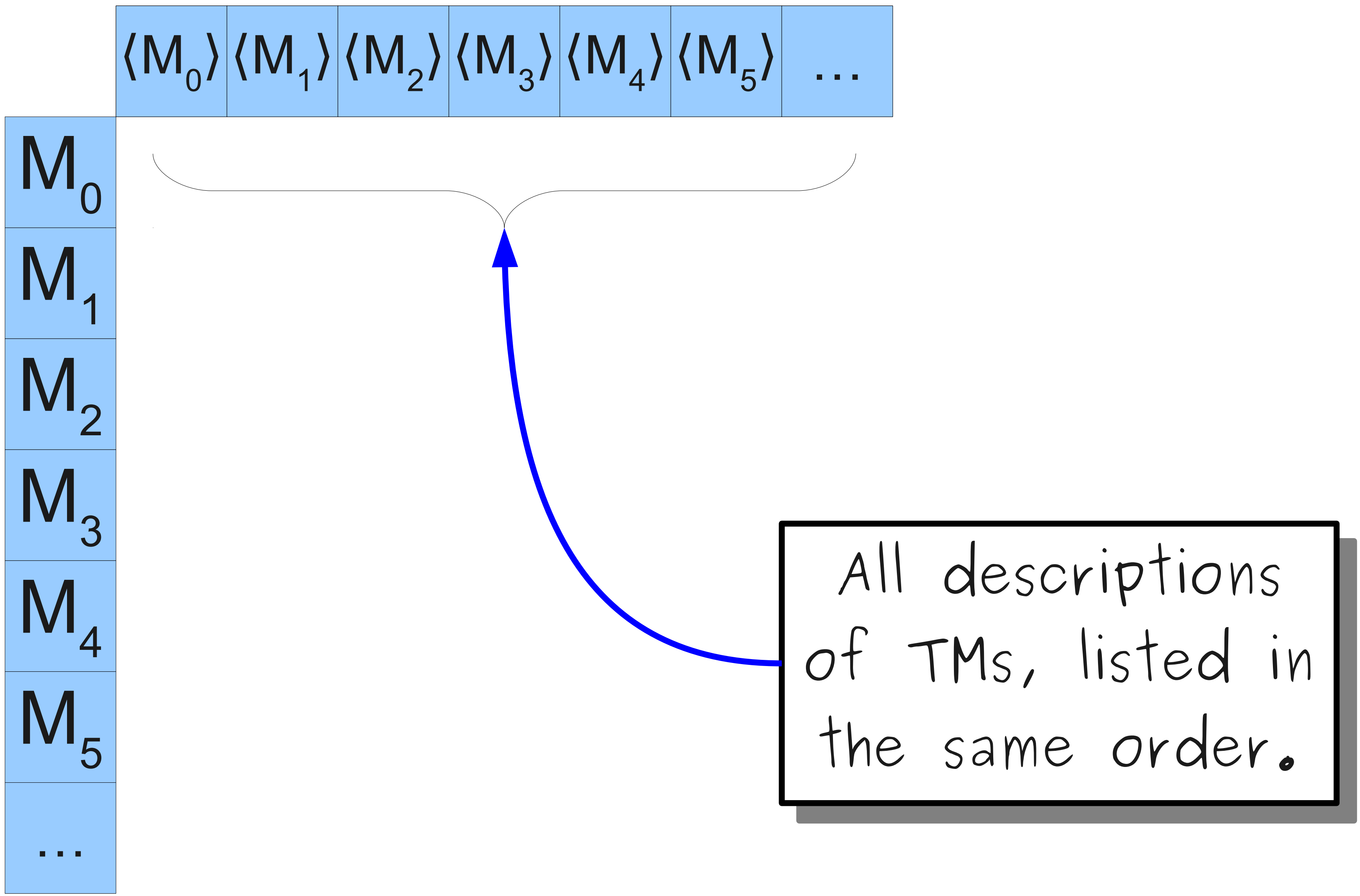
Languages, TMs, and TM Encodings

- Recall: The language of a TM M is the set

$$\mathscr{L}(M) = \{ w \in \Sigma^* \mid M \text{ accepts } w \}$$

- Some of the strings in this set might be descriptions of TMs.
- What happens if we just focus on the set of strings that are legal TM descriptions?


M_0
M_1
M_2
M_3
M_4
M_5
…
All Turing machines,
listed in some order.


⟨M0⟩ ⟨M1⟩ ⟨M2⟩ ⟨M3⟩ ⟨M4⟩ ⟨M5⟩ ...
M0
M1
M2
M3
M4
M5
...
All descriptions of TMs, listed in the same order.

	$\langle M_0 \rangle$	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\langle M_5 \rangle$	…
M_0	Acc	No	No	Acc	Acc	No	…
M_1	Acc	Acc	Acc	Acc	Acc	Acc	…
M_2	Acc	Acc	Acc	Acc	Acc	Acc	…
M_3	No	Acc	Acc	No	Acc	Acc	…
M_4	Acc	No	Acc	No	Acc	No	…
M_5	No	No	Acc	Acc	No	No	…
…	…	…	…	…	…	…	…

	$\langle M_0 \rangle$	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\langle M_5 \rangle$	…
M_0	Acc	No	No	Acc	Acc	No	…
M_1	Acc	Acc	Acc	Acc	Acc	Acc	…
M_2	Acc	Acc	Acc	Acc	Acc	Acc	…
M_3	No	Acc	Acc	No	Acc	Acc	…
M_4	Acc	No	Acc	No	Acc	No	…
M_5	No	No	Acc	Acc	No	No	…
…	…	…	…	…	…	…	…

Acc	Acc	Acc	No	Acc	No	…

	$\langle M_0 \rangle$	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\langle M_5 \rangle$	…
M_0	Acc	No	No	Acc	Acc	No	…
M_1	Acc	Acc	Acc	Acc	Acc	Acc	…
M_2	Acc	Acc	Acc	Acc	Acc	Acc	…
M_3	No	Acc	Acc	No	Acc	Acc	…
M_4	Acc	No	Acc	No	Acc	No	…
M_5	No	No	Acc	Acc	No	No	…
…	…	…	…	…	…	…	…

Acc	Acc	Acc	No	Acc	No	…

Flip all "accept" to "no" and vice-versa

	$\langle M_0 \rangle$	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\langle M_5 \rangle$	…
M_0	Acc	No	No	Acc	Acc	No	…
M_1	Acc	Acc	Acc	Acc	Acc	Acc	…
M_2	Acc	Acc	Acc	Acc	Acc	Acc	…
M_3	No	Acc	Acc	No	Acc	Acc	…
M_4	Acc	No	Acc	No	Acc	No	…
M_5	No	No	Acc	Acc	No	No	…
…	…	…	…	…	…	…	…

No	No	No	Acc	No	Acc	…

No TM has this behavior!

	$\langle M_0 \rangle$	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\langle M_5 \rangle$	…
M_0	Acc	No	No	Acc	Acc	No	…
M_1	Acc	Acc	Acc	Acc	Acc	Acc	…
M_2	Acc	Acc	Acc	Acc	Acc	Acc	…
M_3	No	Acc	Acc	No	Acc	Acc	…
M_4	Acc	No	Acc	No	Acc	No	…
M_5	No	No	Acc	Acc	No	No	…
…	…	…	…	…	…	…	…

No	No	No	Acc	No	Acc	…

	$\langle M_0 \rangle$	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\langle M_5 \rangle$	…
M_0	Acc	No	No	Acc	Acc	No	…
M_1	Acc	Acc	Acc	Acc	Acc	Acc	…
M_2	Acc	Acc	Acc	Acc	Acc	Acc	…
M_3	No	Acc	Acc	No	Acc	Acc	…
M_4	Acc	No	Acc	No	Acc	No	…
M_5	No	No	Acc	Acc	No	No	…
…	…	…	…	…	…	…	…

No	No	No	Acc	No	Acc	…

"The language of all TMs that do not accept their own description."

	$\langle M_0 \rangle$	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\langle M_5 \rangle$	…
M_0	Acc	No	No	Acc	Acc	No	…
M_1	Acc	Acc	Acc	Acc	Acc	Acc	…
M_2	Acc	Acc	Acc	Acc	Acc	Acc	…
M_3	No	Acc	Acc	No	Acc	Acc	…
M_4	Acc	No	Acc	No	Acc	No	…
M_5	No	No	Acc	Acc	No	No	…
…	…	…	…	…	…	…	…

No	No	No	Acc	No	Acc	…

{ ⟨*M*⟩ | *M* is a TM that does not accept ⟨*M*⟩ }

	$\langle M_0 \rangle$	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\langle M_5 \rangle$	…
M_0	Acc	No	No	Acc	Acc	No	…
M_1	Acc	Acc	Acc	Acc	Acc	Acc	…
M_2	Acc	Acc	Acc	Acc	Acc	Acc	…
M_3	No	Acc	Acc	No	Acc	Acc	…
M_4	Acc	No	Acc	No	Acc	No	…
M_5	No	No	Acc	Acc	No	No	…
…	…	…	…	…	…	…	…

No	No	No	Acc	No	Acc	…

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

Diagonalization Revisited

- The **diagonalization language** L_D is defined as

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

- That is, L_D is the set of descriptions of Turing machines that do not accept themselves.

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

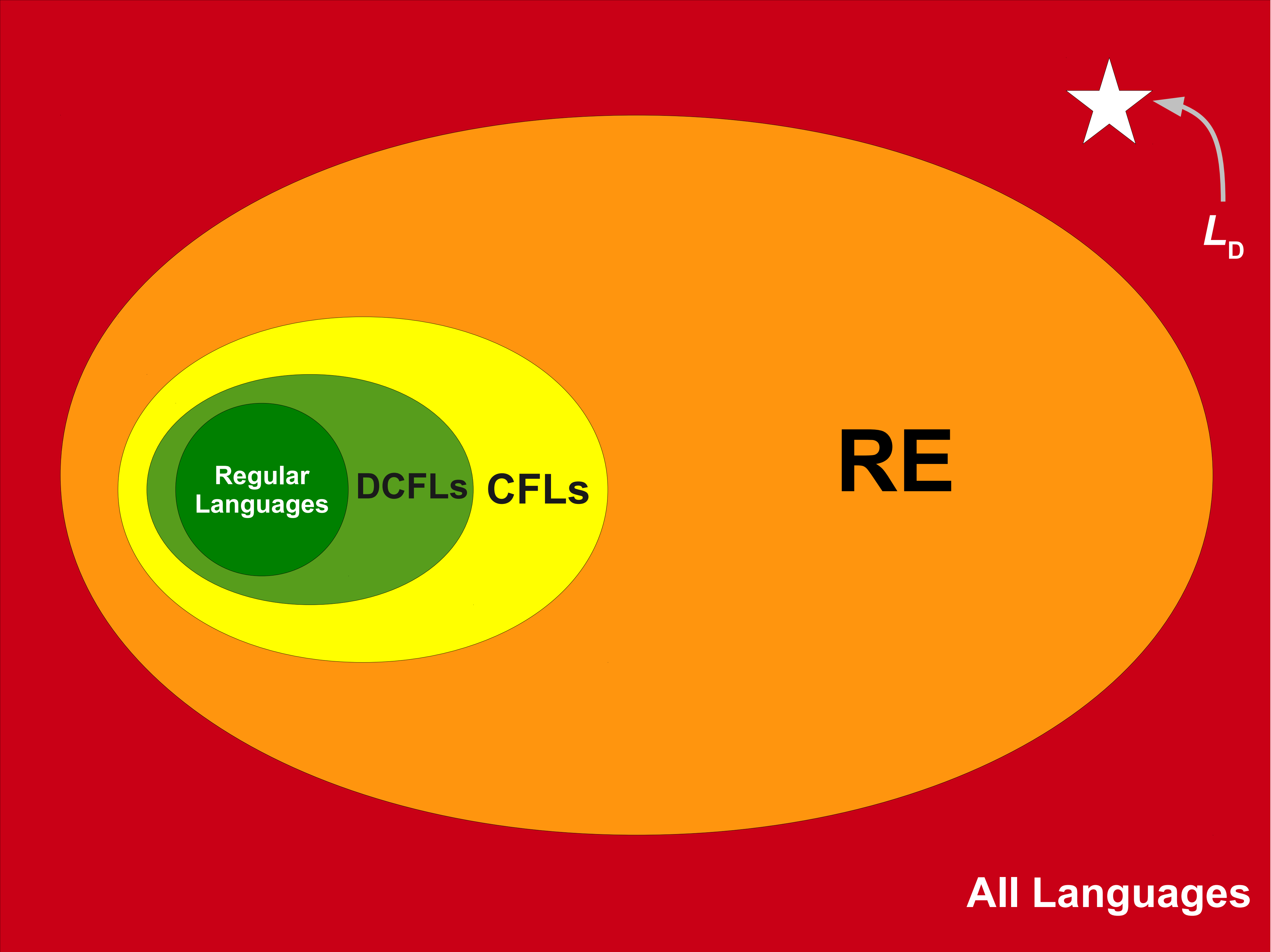
Theorem: $L_D \notin \mathbf{RE}$.

Proof: By contradiction; assume that $L_D \in \mathbf{RE}$. Then there must be some TM R such that $\mathscr{L}(R) = L_D$. We know that either $\langle R \rangle \notin \mathscr{L}(R)$ or $\langle R \rangle \in \mathscr{L}(R)$. We consider each case separately:

Case 1: $\langle R \rangle \notin \mathscr{L}(R)$. By definition of L_D, since $\langle R \rangle \notin \mathscr{L}(R)$, we know that $\langle R \rangle \in L_D$. Since $\langle R \rangle \notin \mathscr{L}(R)$ and $\mathscr{L}(R) = L_D$, we know that $\langle R \rangle \notin L_D$. But this is impossible, since it contradicts the fact that $\langle R \rangle \in L_D$.

Case 2: $\langle R \rangle \in \mathscr{L}(R)$. By definition of L_D, since $\langle R \rangle \in \mathscr{L}(R)$, we know that $\langle R \rangle \notin L_D$. Since $\langle R \rangle \in \mathscr{L}(R)$ and $\mathscr{L}(R) = L_D$, we know that $\langle R \rangle \in L_D$. But this is impossible, since it contradicts the fact that $\langle R \rangle \notin L_D$.

In either case we reach a contradiction, so our assumption must have been wrong. Thus $L_D \notin \mathbf{RE}$. ■



Regular Languages
DCFLs
CFLs
RE
L_D
All Languages

What Just Happened?

$$\boldsymbol{L_{\mathrm{D}} = \{ \langle M \rangle \mid M \textbf{ is a TM and } \langle M \rangle \notin \mathscr{L}(M) \}}$$

- What is it about L_{D} that makes it impossible to solve with a Turing machine?

Indirect self-reference.

- Because TMs can be encoded as strings, TMs that compute over other TMs can be forced to compute some property of themselves *without realizing it*.
- The language L_{D} self-destructs given a Turing machine that recognizes L_{D} by stating "this machine accepts itself if and only if it does not accept itself."

Diagonalization Revisited

- In our original proof of Cantor's theorem, we constructed this diagonal set:

$$\boldsymbol{D = \{\ x \in S \mid x \notin f(x)\ \}}$$

- Note the similarity to the diagonalization language:

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

- We began this class by using Cantor's theorem to show the existence of an unsolvable problems.
- We have now used the exact same technique to single out a specific unsolvable problem.

An Undecidable Problem

Major Ideas from Last Time

- A Turing machine that halts on all inputs is called a **decider**.
- A language L is called **decidable** or **recursive** iff there is a decider M such that $\mathscr{L}(M) = L$.
- The Turing-decidable languages are, therefore, problems for which there is some computer that can always produce a yes or no answer.
- A problem is decidable precisely when there is some algorithm to solve it.
- **Decidability formalizes the definition of an algorithm.**

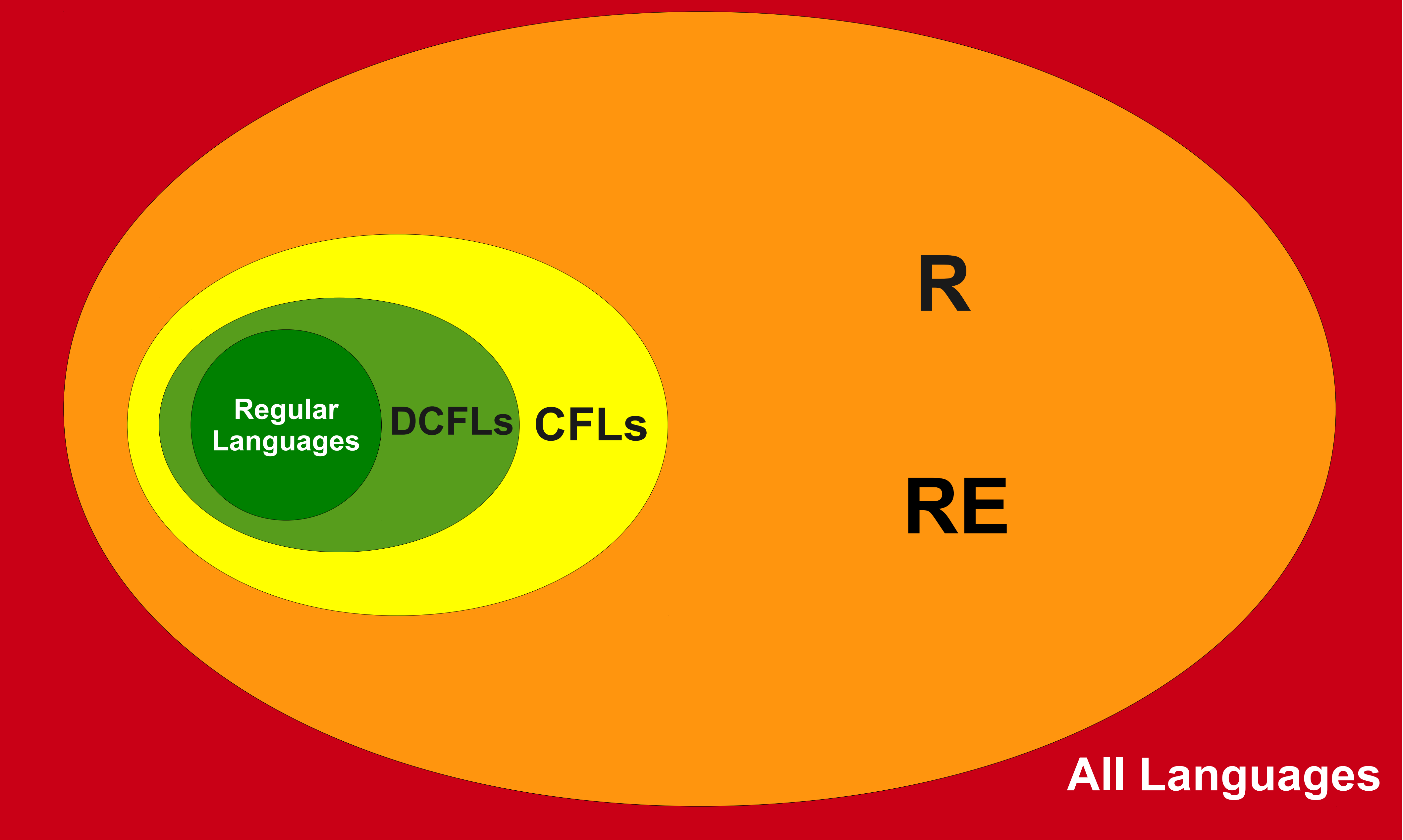
$\mathbf{R} \stackrel{?}{=} \mathbf{RE}$

- **R** is the set of all recursive languages.
- **RE** is the set of all recursively enumerable languages.
- Since all deciders are TMs, $\mathbf{R} \subseteq \mathbf{RE}$.

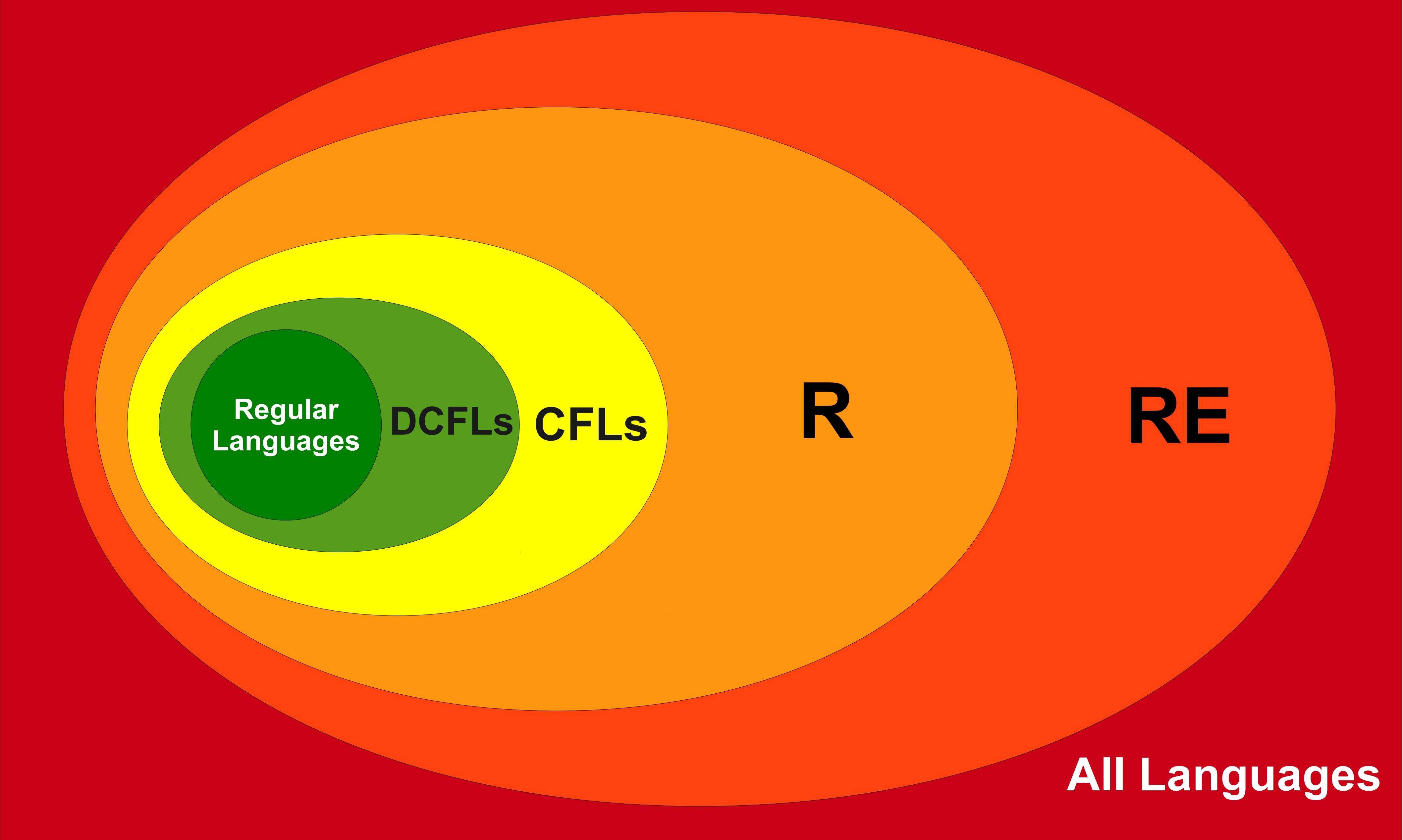
Question: Is R = RE?

- If we can verify a "yes" answer to a problem, can we necessarily solve that problem directly to obtain a yes/no answer?

Which Picture is Correct?



Which Picture is Correct?



Attacking this Problem

- To prove that **R = RE**, we need to show that for any recognizer, there was some equivalent decider.
- To prove that **R ≠ RE**, we need to find a single recognizable language that is undecidable.

Revisiting A_{TM}

- Recall that A_{TM} is the language

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

- $A_{TM} \in \mathbf{RE}$, because it is the language of the universal Turing machine U_{TM}.
- Important theorem:

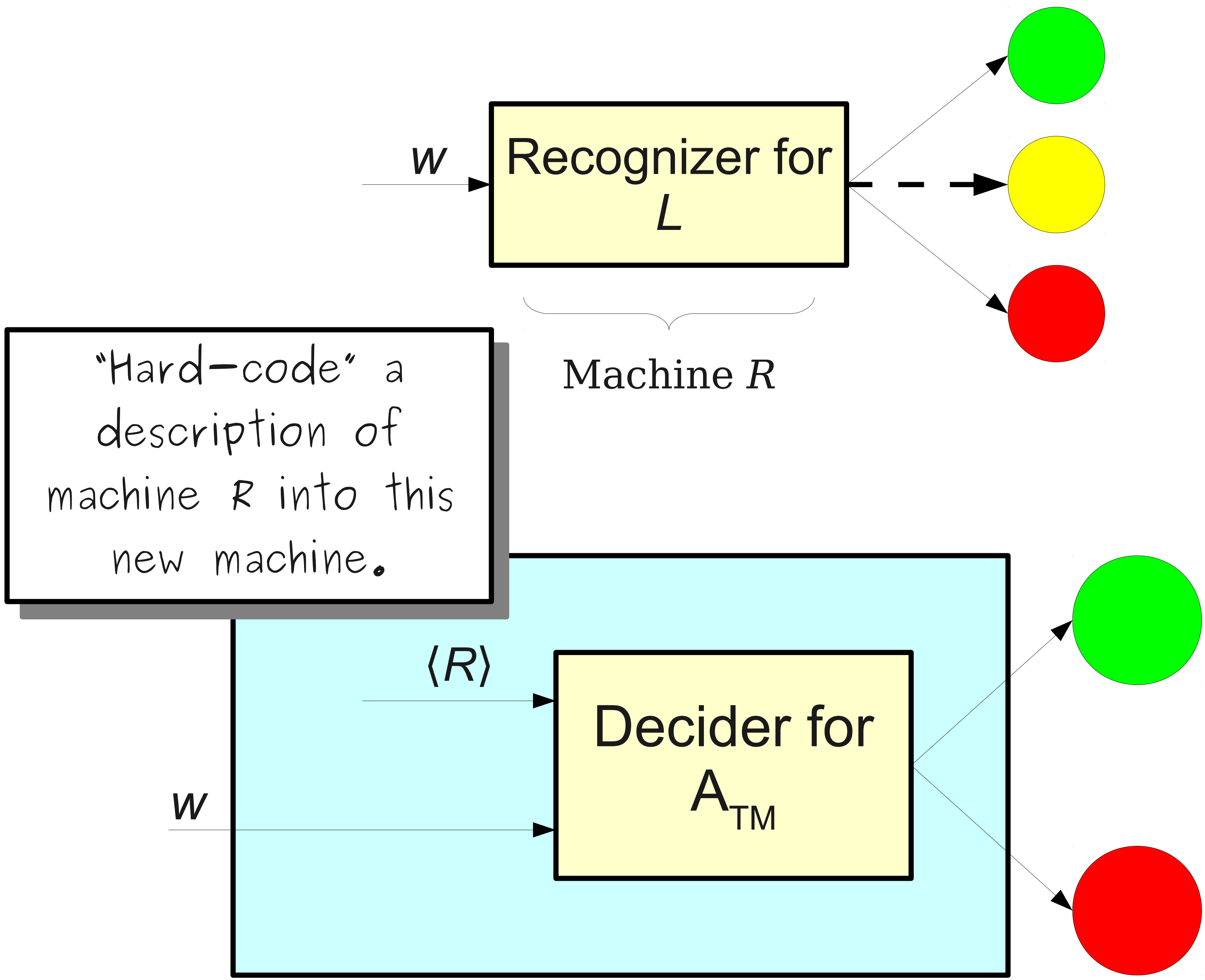
$$\mathbf{R} = \mathbf{RE} \quad \text{iff} \quad A_{TM} \in \mathbf{R}$$

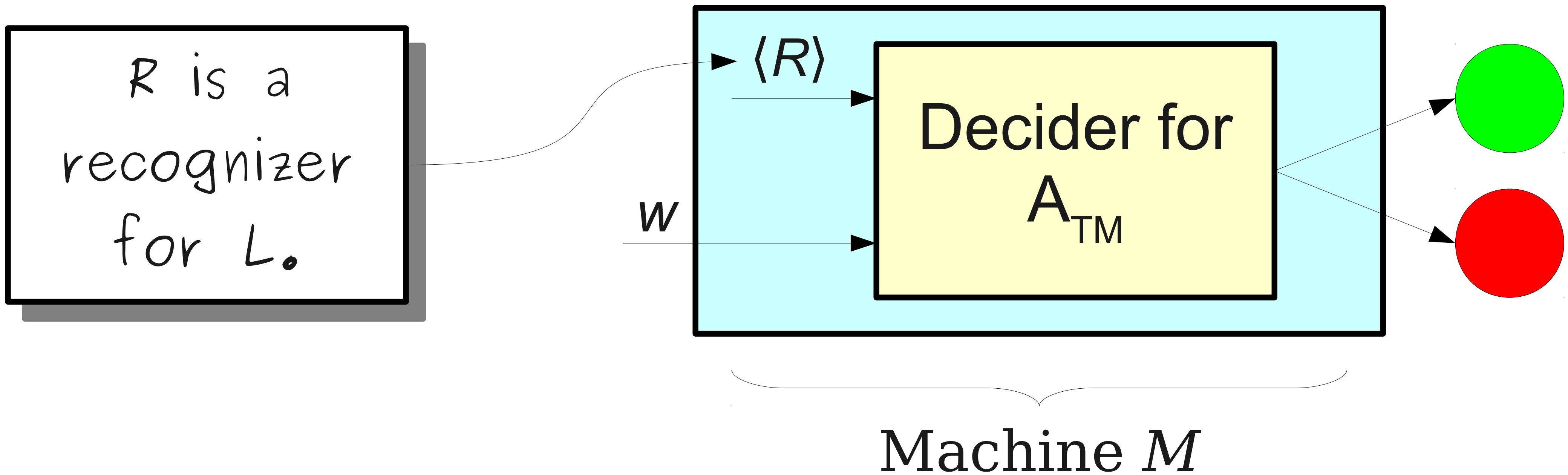
Lemma: If **R** = **RE**, then A_{TM} ∈ **R**.

Proof: Assume **R** = **RE**. Since A_{TM} ∈ **RE**, this means that A_{TM} ∈ **R**. Therefore, A_{TM} ∈ **R**. ■

The Other Direction

- We want to prove that if $A_{TM} \in \mathbf{R}$, then $\mathbf{R} = \mathbf{RE}$.
- We will show that if A_{TM} is decidable, then given any recognizer for a language L, we can construct a decider for L.


w
Recognizer for L
Machine R
"Hard-code" a description of machine R into this new machine.
⟨R⟩
w
Decider for A_TM



M accepts w

iff

The decider for A_{TM} accepts $\langle R, w\rangle$

iff

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

iff

R accepts w

iff

$w \in \mathscr{L}(R)$

M is a decider for $\mathscr{L}(R)$, so $\mathscr{L}(R) \in \mathbf{R}$.

Theorem: If $A_{TM} \in \mathbf{R}$, then $\mathbf{R} = \mathbf{RE}$.

Proof: Assume that $A_{TM} \in \mathbf{R}$. Then there must be a decider D such that $\mathscr{L}(D) = A_{TM}$. Consider any language $L \in \mathbf{RE}$; we show that $L \in \mathbf{R}$. Since our choice of L was arbitrary, this shows that $\mathbf{RE} \subseteq \mathbf{R}$. Since $\mathbf{R} \subseteq \mathbf{RE}$, this proves $\mathbf{R} = \mathbf{RE}$.

Since $L \in \mathbf{RE}$, there is some recognizer for L; call it R. Then consider the following TM M:

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

We prove that $\mathscr{L}(M) = L$ and that M is a decider. To see that $\mathscr{L}(M) = L$, consider any string w. Then M accepts w iff D accepts $\langle R, w\rangle$. Note that D accepts $\langle R, w\rangle$ iff R accepts w. Finally, R accepts w iff $w \in \mathscr{L}(R) = L$. Thus M accepts w iff $w \in L$, so $\mathscr{L}(M) = L$.

To show that M is a decider, consider what happens when we run M on an arbitrary string w. M first runs D on $\langle R, w\rangle$. Since D is a decider, D eventually halts. If D accepts $\langle R, w\rangle$, then M accepts. If D rejects $\langle R, w\rangle$, then M rejects. Thus M halts on all inputs, so it is a decider.

Since M is a decider for L, this proves $L \in \mathbf{R}$ as required. ■

$\mathbf{R} = \mathbf{RE}$ iff $A_{TM} \in \mathbf{R}$.

So, is $A_{TM} \in \mathbf{R}$?

If A_{TM} is Decidable...

- Let $P(n) \equiv$ "Every tournament graph with n players has a winner."
- For any fixed n, we can check whether $P(n)$ is true by listing all tournament graphs and then seeing if they have a tournament winner.
- Consider this TM:

  "On input w:

  Ignore w.

  For $n = 1$ to ∞:

  If $P(n)$ is false, accept."
- This TM accepts any string w iff there is some tournament graph with no winner.
- Using A_{TM}, we could decide whether the theorem is true by deciding whether this program accepts or rejects some string w.

If A_{TM} is Decidable...

- Consider the following TM:

  "On input φ, where φ is a formula in first-order logic:

    Nondeterministically guess a proof of φ.

    Deterministically verify that this proof is valid.

    If so, accept.

    Otherwise, reject."

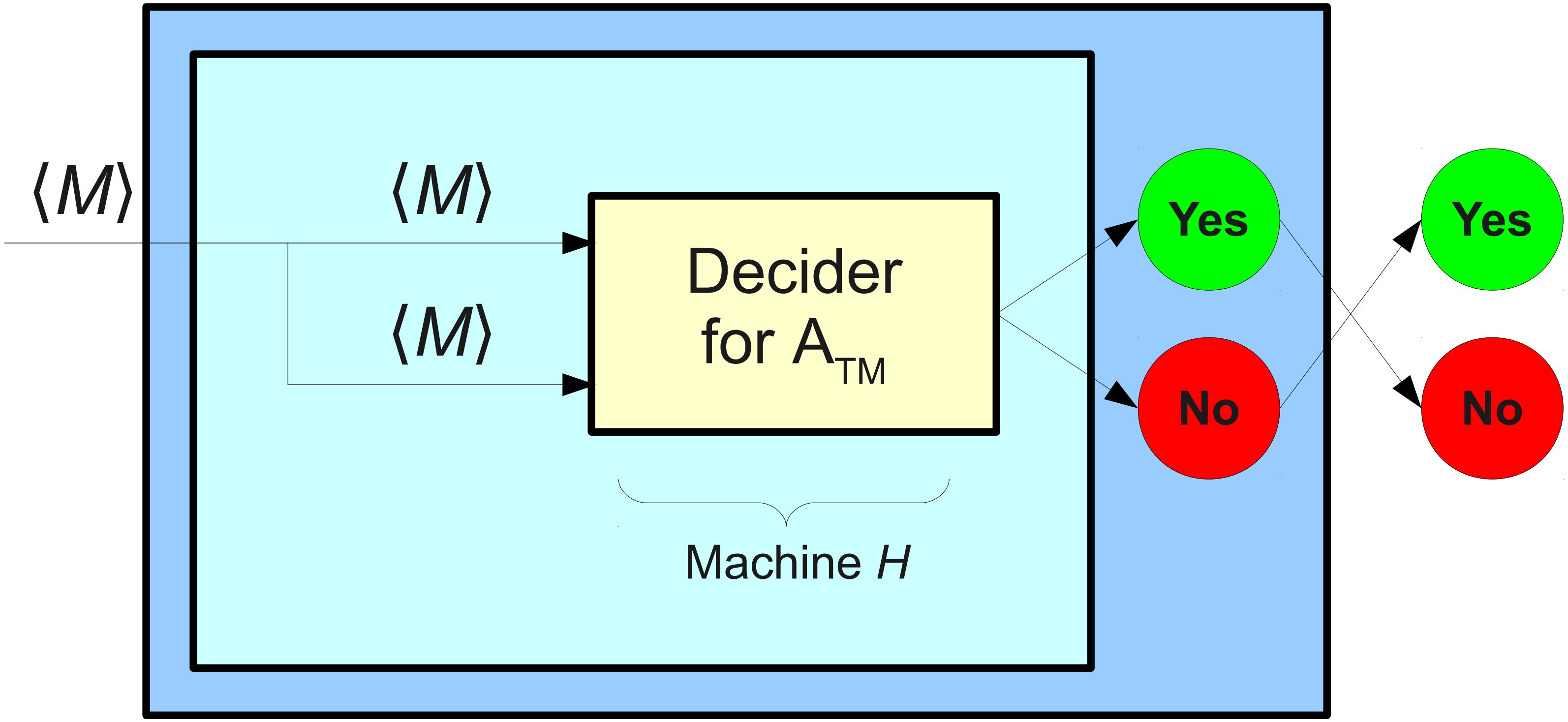
- This TM accepts φ iff φ is provable.
- Using A_{TM}, we could automatically determine whether *any* formula was provable by deciding if the above TM accepts it.

Theorem: A_{TM} is undecidable.

Corollary: $\mathbf{R} \neq \mathbf{RE}$.

Assume, for the sake of contradiction, that A_{TM} is decidable.

Let H be a decider for it.



"On input $\langle M \rangle$:
Construct $\langle M, \langle M \rangle\rangle$.
Run H on $\langle M, \langle M \rangle\rangle$.
If H accepts $\langle M, \langle M \rangle\rangle$, reject.
If H rejects $\langle M, \langle M \rangle\rangle$, accept."

If $\langle M \rangle \in \mathscr{L}(M)$, reject.
If $\langle M \rangle \notin \mathscr{L}(M)$, accept.

This is a decider for L_D!

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

Theorem: $A_{TM} \notin \mathbf{R}$.

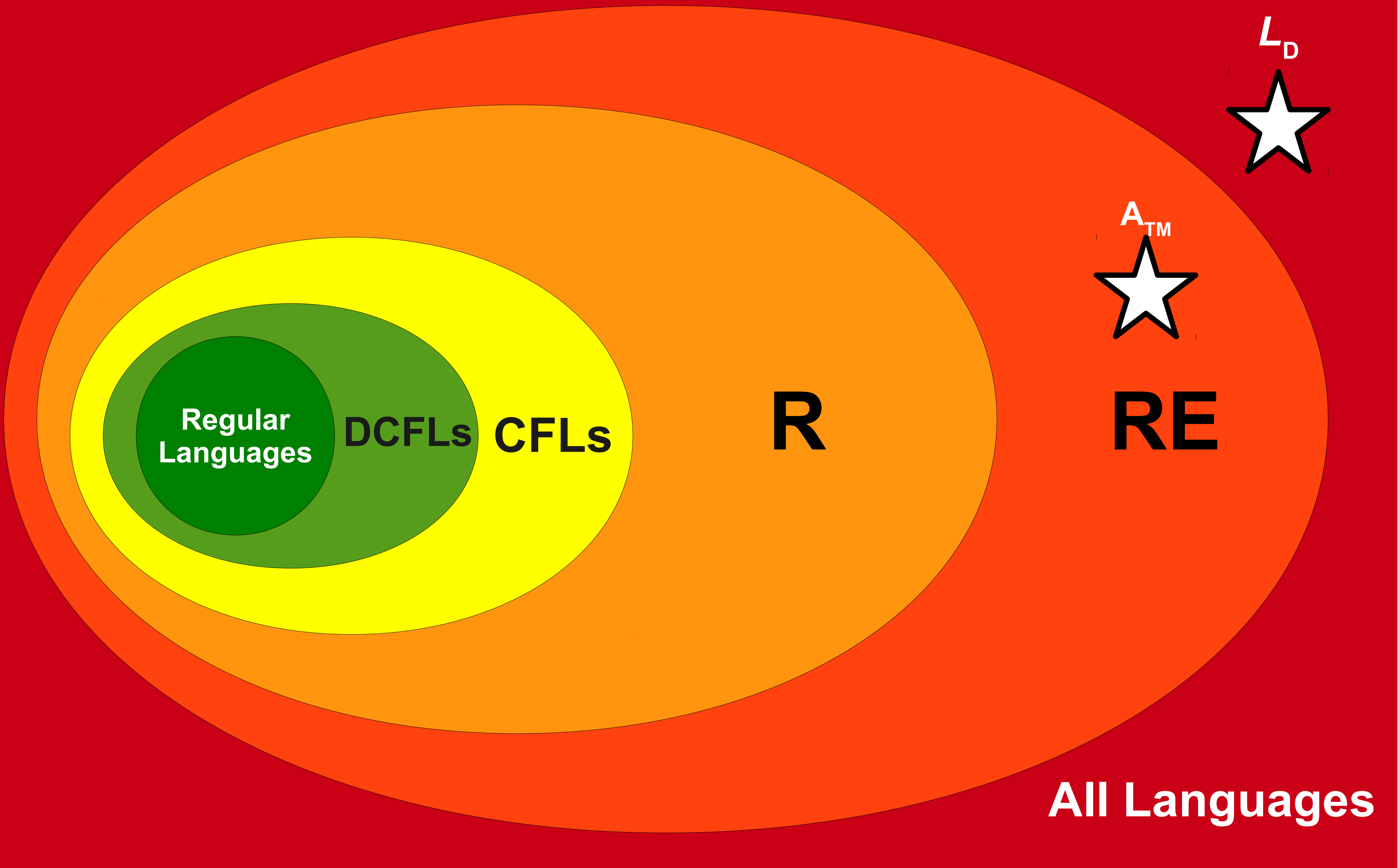
Proof: By contradiction; assume that $A_{TM} \in \mathbf{R}$ and let H be a decider for it. Then consider this machine D:

D = "On input $\langle M \rangle$:
Construct $\langle M, \langle M \rangle \rangle$.
Run H on $\langle M, \langle M \rangle \rangle$.
If H accepts $\langle M, \langle M \rangle \rangle$, reject.
If H rejects $\langle M, \langle M \rangle \rangle$, accept."

We claim that $\mathscr{L}(D) = L_D$. To see this, note that D accepts $\langle M \rangle$ iff H rejects $\langle M, \langle M \rangle \rangle$. Since H is a decider for A_{TM}, H rejects $\langle M, \langle M \rangle \rangle$ iff $\langle M, \langle M \rangle \rangle \notin A_{TM}$. Note that $\langle M, \langle M \rangle \rangle \notin A_{TM}$ iff $\langle M \rangle \notin \mathscr{L}(M)$, since $\langle M, \langle M \rangle \rangle$ is an encoding of a TM/string pair. Consequently, we have that D accepts $\langle M \rangle$ iff $\langle M \rangle \notin \mathscr{L}(M)$. Therefore, $\mathscr{L}(D) = L_D$.

Since $\mathscr{L}(D) = L_D$, we know that $L_D \in \mathbf{RE}$. But this is impossible, since we know that $L_D \notin \mathbf{RE}$. We have reached a contradiction, so our assumption must have been wrong. Thus $A_{TM} \notin \mathbf{R}$. ■

The Limits of Computability



What Just Happened?

- Initially, we proved that $L_D \notin \mathbf{RE}$.
- Using this fact, we proved that $A_{TM} \notin \mathbf{R}$ by using the following reasoning:
  - If $A_{TM} \in \mathbf{R}$, then $L_D \in \mathbf{RE}$.
  - $L_D \notin \mathbf{RE}$.
  - Therefore, $A_{TM} \notin \mathbf{R}$.

Finding Unsolvable Problems

- Unlike regular languages or context-free languages, there is no pumping lemma for **R** or **RE** languages.
  - The model of computation is just too powerful.
- Instead, we will find unsolvable problems using reasoning like before:
  - Assume that some language A is "solvable."
  - Using the "solver" for A, build a "solver" for B.
  - Using advance knowledge that B is "unsolvable," derive a contradiction.
  - Conclude, therefore, that A is "unsolvable."

A Different Perspective on A_{TM}

Assume H is a decider for A_{TM}.

D = "On input $\langle M \rangle$:
Construct $\langle M, \langle M \rangle \rangle$.
Run H on $\langle M, \langle M \rangle \rangle$.
If H accepts $\langle M, \langle M \rangle \rangle$, reject.
If H rejects $\langle M, \langle M \rangle \rangle$, accept."

What happens if we run D on $\langle D \rangle$?

D accepts $\langle D \rangle$

iff

H rejects $\langle D, \langle D \rangle \rangle$

iff

D does not accept $\langle D \rangle$

Another Undecidable 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:

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

- 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 $\mathscr{L}(H) = HALT$, so $HALT \in \mathbf{RE}$.

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

(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 $\mathit{HALT} \notin \mathbf{R}$.

Deciding A_{TM} using *HALT*

- Suppose you are given a TM M and a string w.
- You are promised that M halts on w.
- Can you decide whether M accepts w?
- **Yes:** Just run it and see what happens.
- Now, suppose you have a decider for *HALT*.
- Can you decide whether M accepts w?

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

Run the decider for *HALT* on $\langle M, w \rangle$.

If the decider rejects $\langle M, w \rangle$, reject.

Otherwise: (the decider accepts $\langle M, w \rangle$)

Run M on w.

If M accepts w, accept.

If M rejects w, reject."

D accepts $\langle M, w \rangle$

iff

The decider for HALT accepts $\langle M, w \rangle$ and M accepts w

iff

M halts on w and M accepts w

iff

M accepts w

iff

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

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

Run the decider for *HALT* on $\langle M, w \rangle$.

If the decider rejects $\langle M, w \rangle$, reject.

Otherwise: (the decider accepts $\langle M, w \rangle$)

Run M on w.

If M accepts w, accept.

If M rejects w, reject."

$\mathscr{L}(D) = \mathrm{A}_{\mathrm{TM}}$

D is a decider.

So $\mathrm{A}_{\mathrm{TM}} \in \mathbf{R}$.

Run D on any input $\langle M, w \rangle$.

If the decider for *HALT* rejects, $\langle M, w \rangle$, D rejects.

Otherwise, we know M halts on w.

Then we run M on w.

We know M eventually halts on w.

If M accepts w, D accepts; if M rejects w, D rejects.

Thus D always halts.

Theorem: HALT $\notin$ **R**.

Proof: By contradiction; assume that *HALT* $\in$ **R** and let H be a decider for it.

Consider the following machine D:

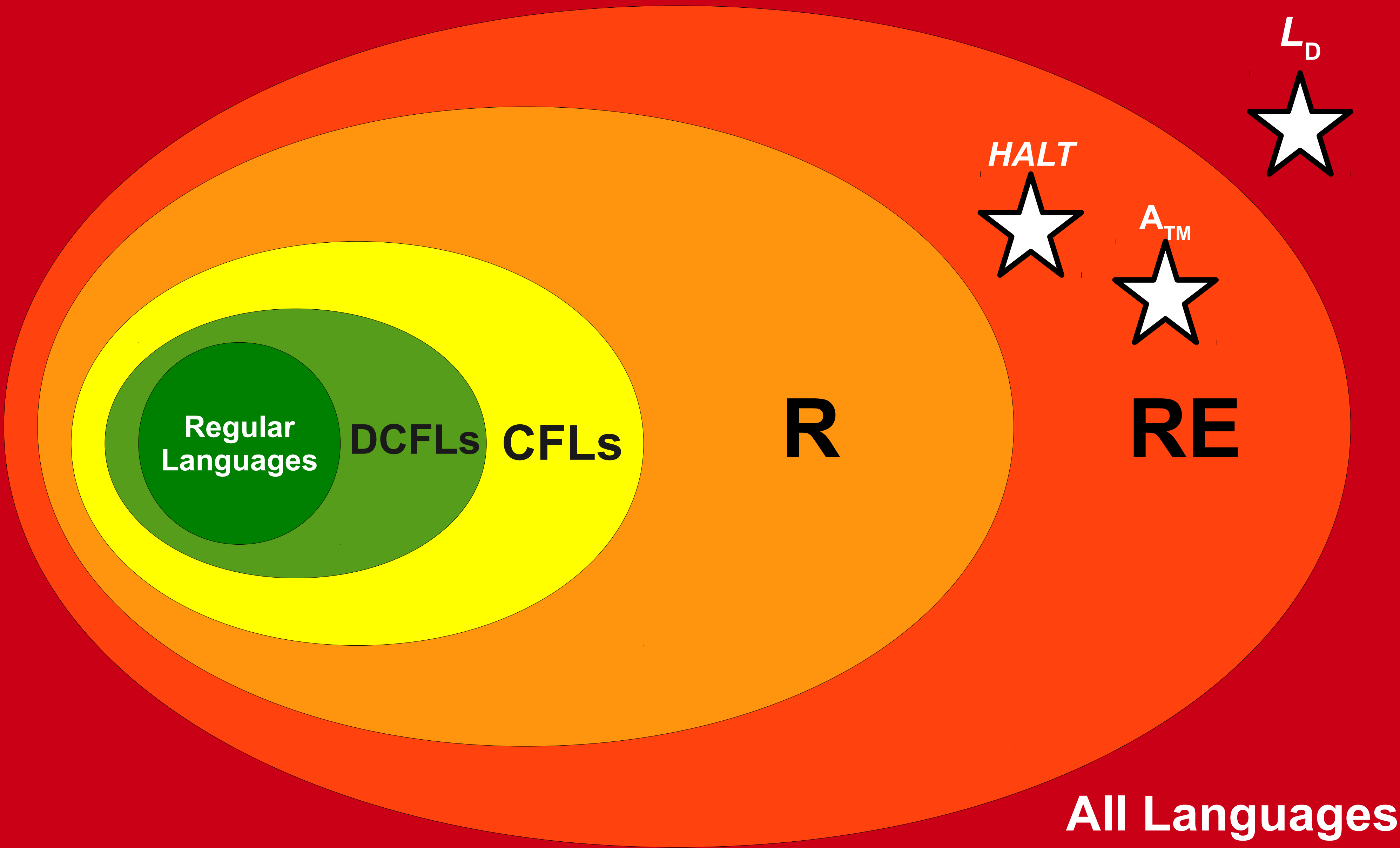
D = "On input $\langle M, w\rangle$:
    Run H on $\langle M, w\rangle$.
    If H rejects $\langle M, w\rangle$, reject.
    If H accepts $\langle M, w\rangle$:
        Run M on w.
        If M accepts w, accept.
        If M rejects w, reject."

We claim that D is a decider for A_{TM}. First, we prove that D halts on all inputs. To see this, consider what happens if we run D on any TM/string pair $\langle M, w\rangle$. D first runs H on $\langle M, w\rangle$. If H rejects, D rejects and halts. Otherwise, since H is a decider, H accepts $\langle M, w\rangle$, so M halts on w. D then runs M on w. Since we know M halts on w, M either accepts or rejects. If M accepts, D accepts; if M rejects, D rejects. Thus D halts on all inputs.

To see that $\mathscr{L}(D) = A_{TM}$, note that D accepts $\langle M, w\rangle$ iff H accepts $\langle M, w\rangle$ and M accepts w. Since H accepts $\langle M, w\rangle$ iff M halts on w, we have that D accepts $\langle M, w\rangle$ iff M halts on w and M accepts w. Since M halts on w iff either M accepts w or M rejects w, the statement "M halts on w and M accepts w" is equivalent to "M accepts w." Thus D accepts $\langle M, w\rangle$ iff M accepts w. Since M accepts w iff $\langle M, w\rangle \in A_{TM}$, this means that D accepts $\langle M, w\rangle$ iff $\langle M, w\rangle \in A_{TM}$. Thus $\mathscr{L}(D) = A_{TM}$.

Since $\mathscr{L}(D) = A_{TM}$ and D is a decider, this means $A_{TM} \in$ **R**. But this is impossible, since we know $A_{TM} \notin$ **R**. We have reached a contradiction, so our assumption must have been wrong. Thus *HALT* $\notin$ **R**. ■

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.**

Two More Unsolvable Problems

More Unsolvable Problems

- Recall from last time:

  If $L \in \mathbf{RE}$ and $\overline{L} \in \mathbf{RE}$, then $L \in \mathbf{R}$.

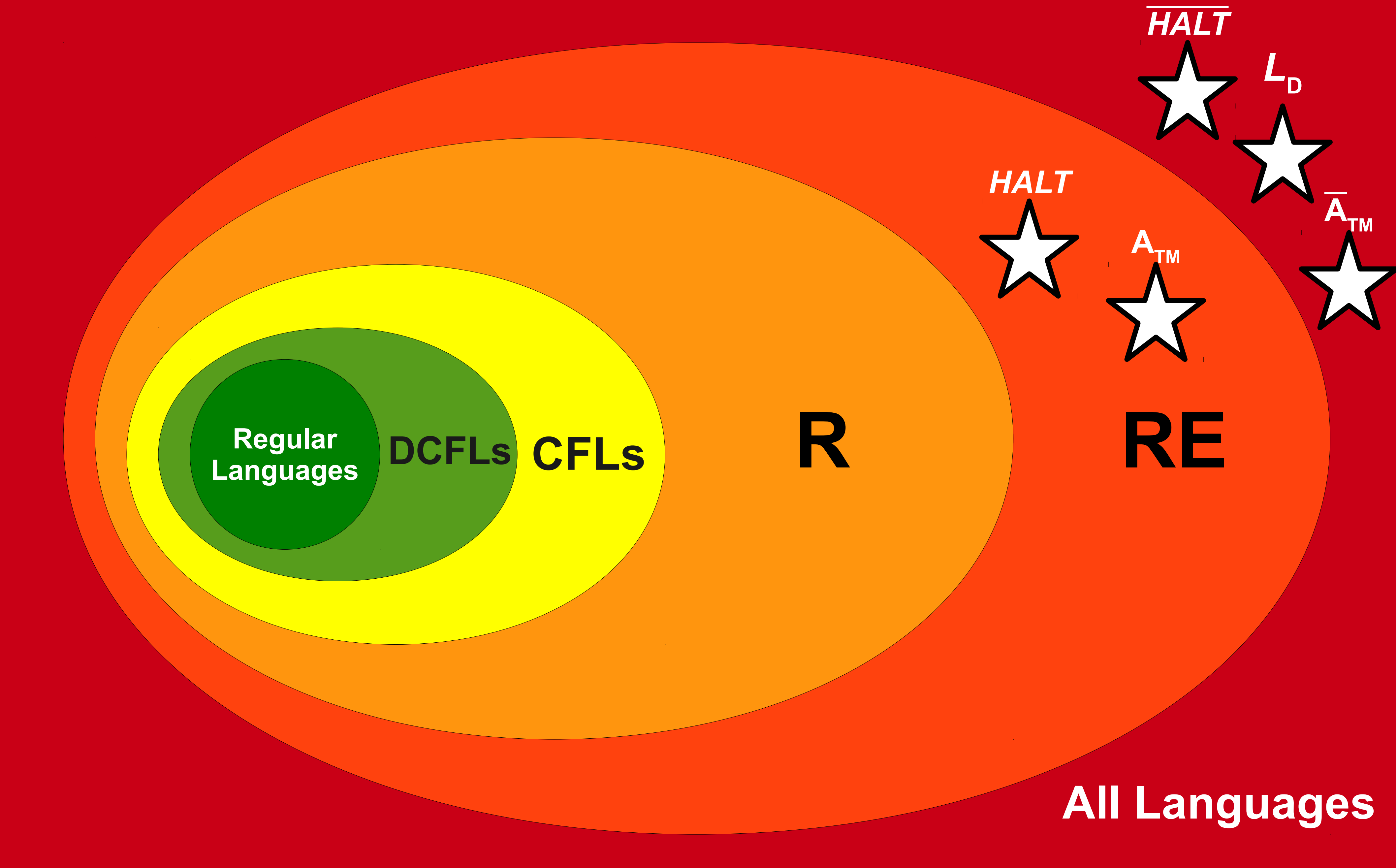
- Taking the contrapositive:

  If L $\notin \mathbf{R}$, then $L \notin \mathbf{RE}$ or $\overline{L} \notin \mathbf{RE}$.

- As a corollary:

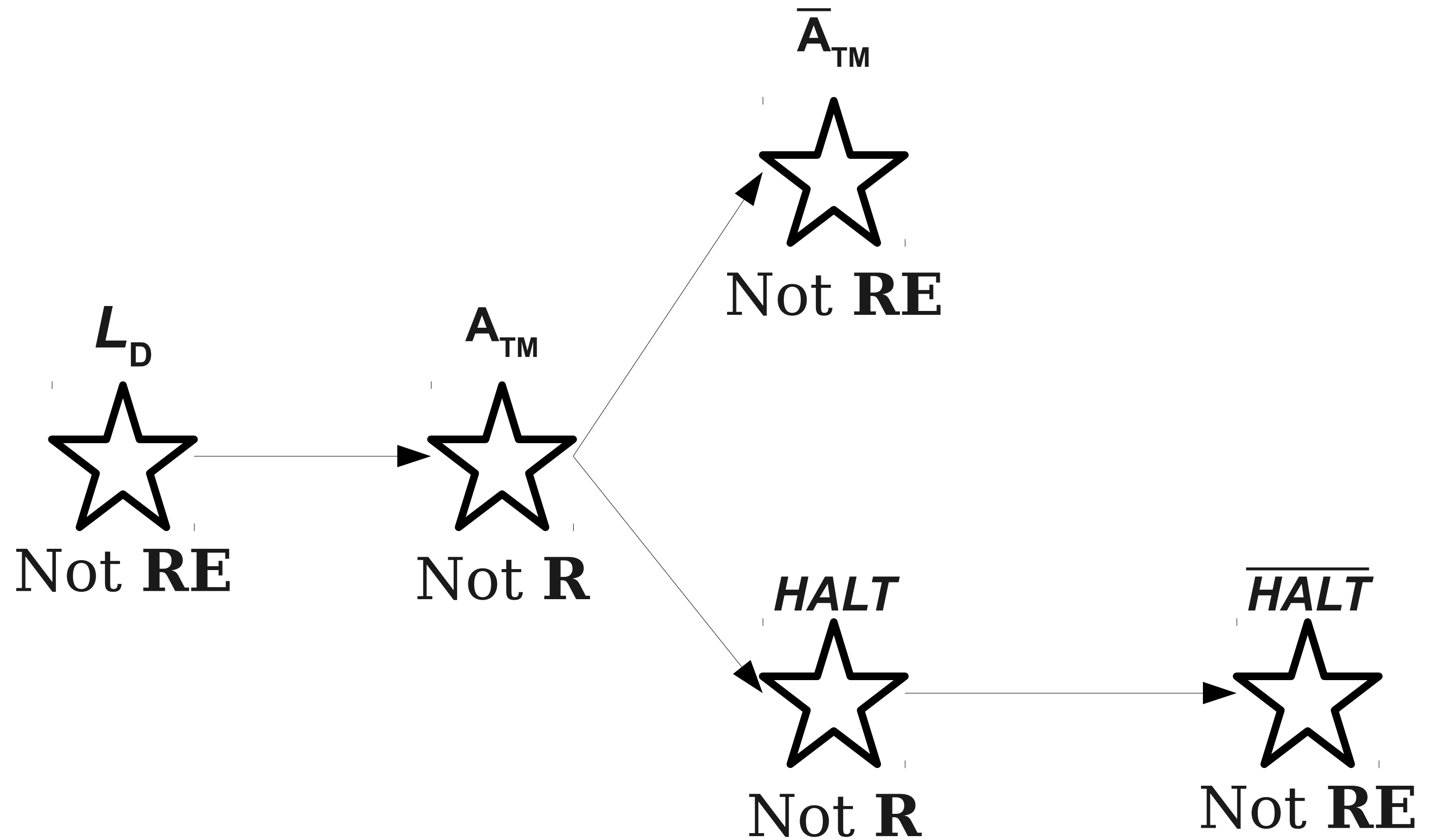
  If $L \notin \mathbf{R}$ and $L \in \mathbf{RE}$, then $\overline{L} \notin \mathbf{RE}$.

The Limits of Computability



Major Ideas from Today

Finding Unsolvable Problems



Finding Unsolvable Problems

- We directly proved that $L_D \notin \mathbf{RE}$ by using a proof by diagonalization.
- We proved $A_{TM} \notin \mathbf{R}$ (and thus $\mathbf{R} \neq \mathbf{RE}$) by showing that if $A_{TM} \in \mathbf{R}$, then $L_D \in \mathbf{RE}$ (which we know is not true).
- We proved $HALT \notin \mathbf{R}$ by showing that if $HALT \in \mathbf{R}$, then $A_{TM} \in \mathbf{R}$ (which we know is not true).
- We proved $\overline{A}_{TM} \notin \mathbf{RE}$ and $\overline{HALT} \notin \mathbf{RE}$ by showing that if they were in $\mathbf{RE}$, then $A_{TM} \in \mathbf{R}$ and $HALT \in \mathbf{R}$ (which we know is not true).

Finding Unsolvable Problems

- Proving languages are not in **RE** or not in **R** is *fundamentally different* than proving languages are not regular or not context free.
- We will need to develop a more powerful array of tools to prove problems are undecidable or unrecognizable.

Next Time

- **Reductions**
  - Solving one problem using a solver for another.
- **Mapping Reductions**
  - Relating the difficulty of problems to one another using reductions.
- **More Unsolvable Problems**
  - What other problems cannot be solved by computers?