Decidability and Undecidability

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

- The universal Turing machine U_{TM} can be used as a subroutine in other Turing machines.

> H = "On input $\langle M \rangle$, where M is a Turing machine:
> - Run M on ε.
> - If M accepts ε, then H accepts $\langle M \rangle$.
> - If M rejects ε, then H rejects $\langle M \rangle$.

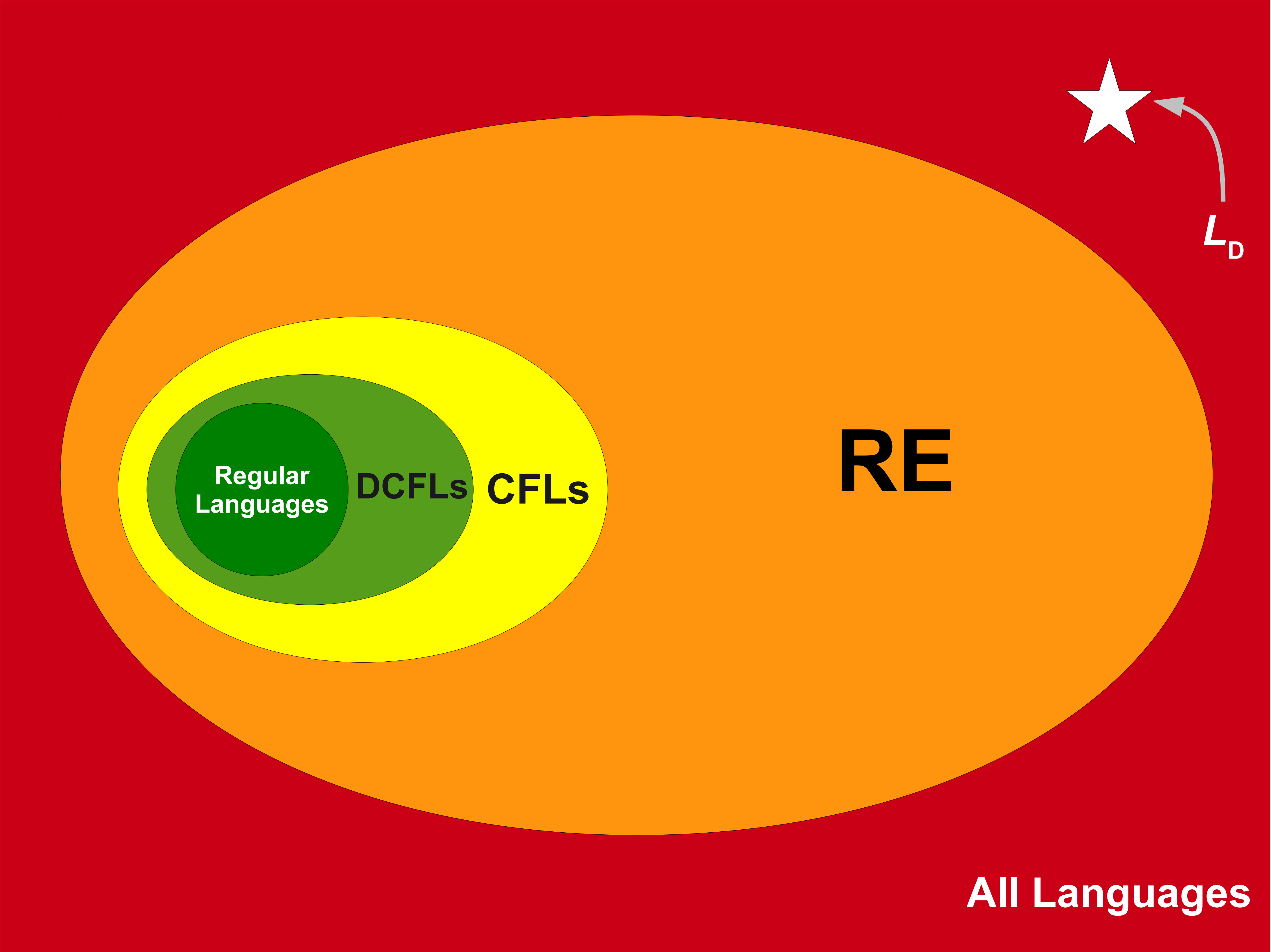
> H = "On input $\langle M \rangle$, where M is a Turing machine:
> - Nondeterministically guess a string w.
> - Run M on w.
> - If M accepts w, then H accepts $\langle M \rangle$.
> - If M rejects w, then H rejects $\langle M \rangle$.

Major Ideas from Last Time

- The **diagonalization language**, which we denote $\boldsymbol{L_\mathrm{D}}$, is defined as

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

- That is, L_D is the set of descriptions of Turing machines that do not accept themselves.
- ***Theorem:*** $L_\mathrm{D} \notin$ **RE**



Regular Languages
DCFLs
CFLs
RE
L_D
All Languages

Outline for Today

- **More non-RE Languages**
  - We now know $L_D \notin \mathbf{RE}$. Can we use this to find other non-**RE** languages?
- **Decidability and Class R**
  - How do we formalize the idea of an algorithm?
- **Undecidable Problems**
  - What problems admit no algorithmic solution?

Additional Unsolvable Problems

Finding Unsolvable Problems

- We can use the fact that $L_\mathrm{D} \notin \mathbf{RE}$ to show that other languages are also not **RE**.
- General proof approach: to show that some language L is not **RE**, we will do the following:
  - Assume for the sake of contradiction that $L \in \mathbf{RE}$, meaning that there is some TM M for it.
  - Show that we can build a TM that uses M as a subroutine in order to recognize L_D.
  - Reach a contradiction, since no TM recognizes L_D.
  - Conclude, therefore, that $L \notin \mathbf{RE}$.

The Complement of A_{TM}

- Recall: the language A_{TM} is the language of the universal Turing machine U_{TM}:

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

- The complement of A_{TM} (denoted $\overline{A}_{TM}$) is the language of all strings not contained in A_{TM}.
- Questions:
  - What language is this?
  - Is this language **RE**?

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

- The language A_{TM} is defined as

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

- Equivalently:

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

- Thus $\overline{A}_{TM}$ is

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

- T

epts *w*}

- E

M

ts *w*}

- Thus A_{TM} is

{*x* | *x* ≠ ⟨*M*, *w*⟩ for any TM *M* and string *w*, or *M* is a TM that does not accept *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;
}
```

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

- The language A_{TM} is defined as

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

- Thus $\overline{A}_{TM}$ is the language

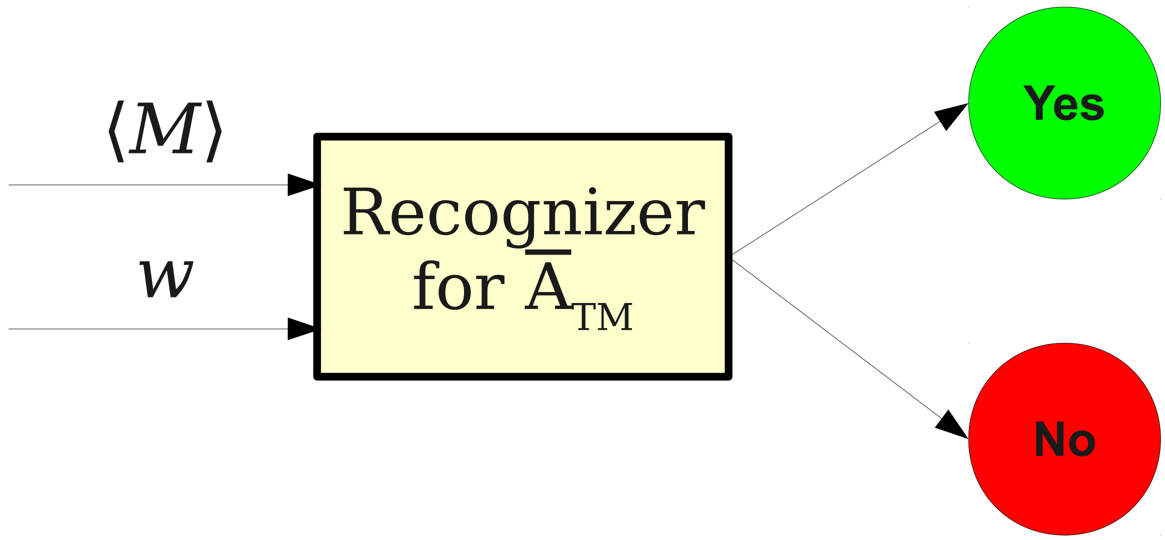
$$\{\langle M, w\rangle \mid M \text{ is a TM that doesn't accept } w\}$$

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

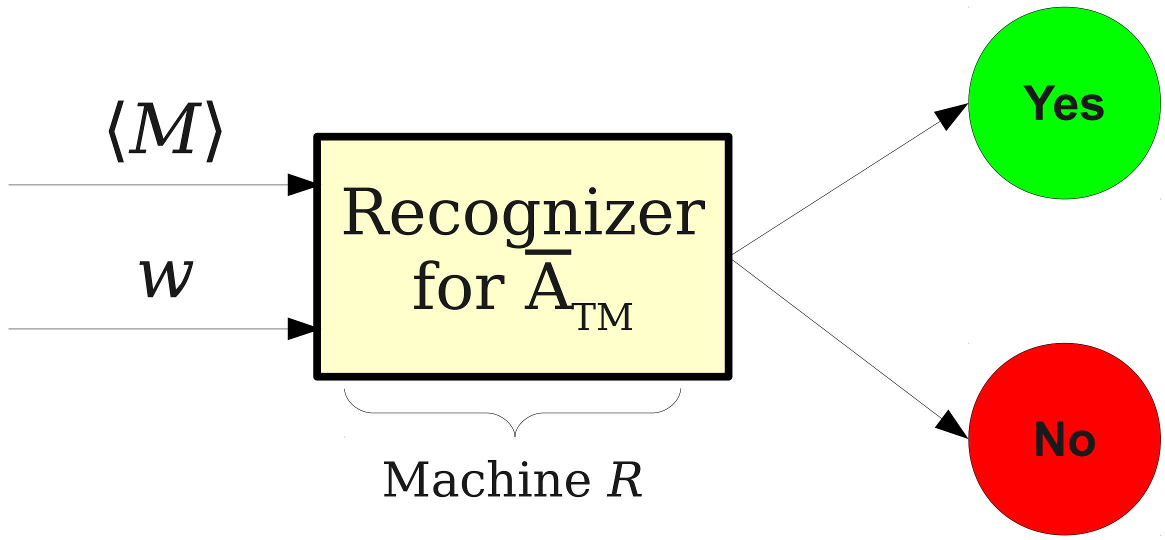
- Although the language $\mathrm{A}_{\mathrm{TM}} \in \mathbf{RE}$ (since it's the language of U_{TM}), its complement $\overline{\mathrm{A}}_{\mathrm{TM}} \notin \mathbf{RE}$.
- We will prove this as follows:
  - Assume, for contradiction, that $\overline{\mathrm{A}}_{\mathrm{TM}} \in \mathbf{RE}$.
  - This means there is a TM R for $\overline{\mathrm{A}}_{\mathrm{TM}}$.
  - Using R as a subroutine, we will build a TM H that will recognize L_{D}.
  - This is impossible, since $L_{\mathrm{D}} \notin \mathbf{RE}$.
  - Conclude, therefore, that $\overline{\mathrm{A}}_{\mathrm{TM}} \notin \mathbf{RE}$.

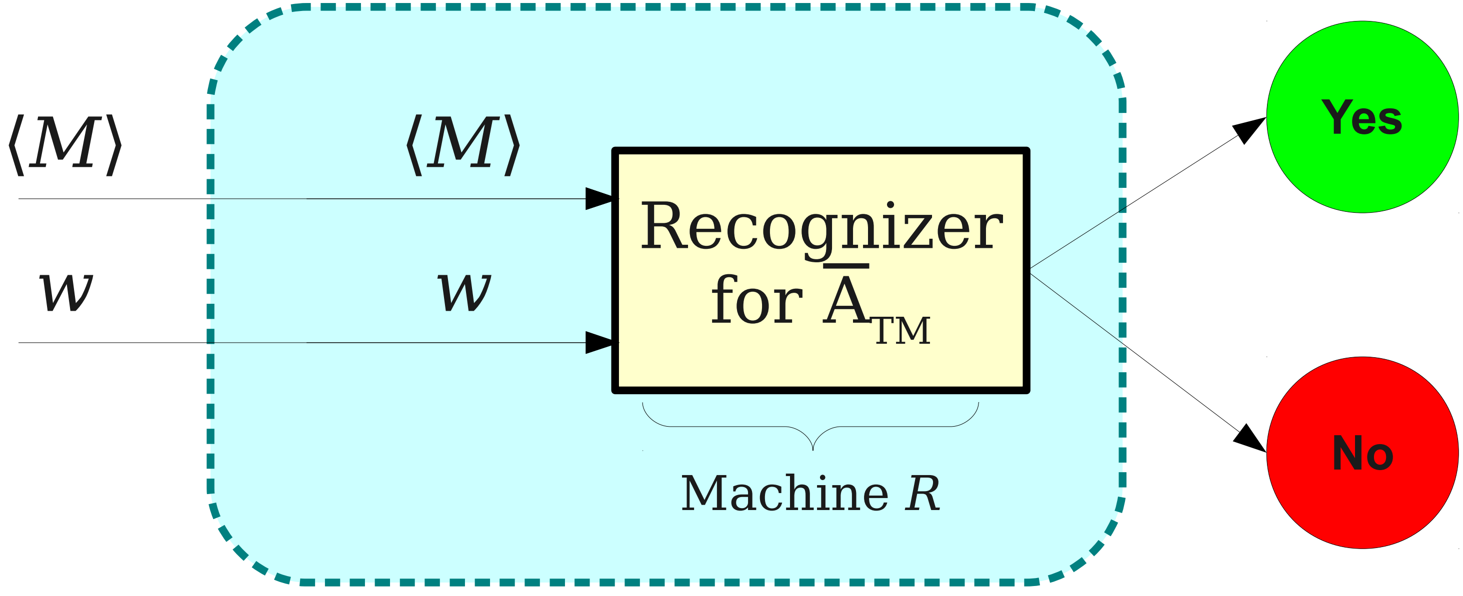
Comparing L_{D} and $\overline{\text{A}}_{\text{TM}}$

- The languages L_{D} and $\overline{\text{A}}_{\text{TM}}$ are closely related:
  - L_{D}: Does M not accept $\langle M \rangle$?
  - $\overline{\text{A}}_{\text{TM}}$: Does M not accept string w?
- Given this connection, we will show how to turn a hypothetical recognizer for $\overline{\text{A}}_{\text{TM}}$ into a hypothetical recognizer for L_{D}.

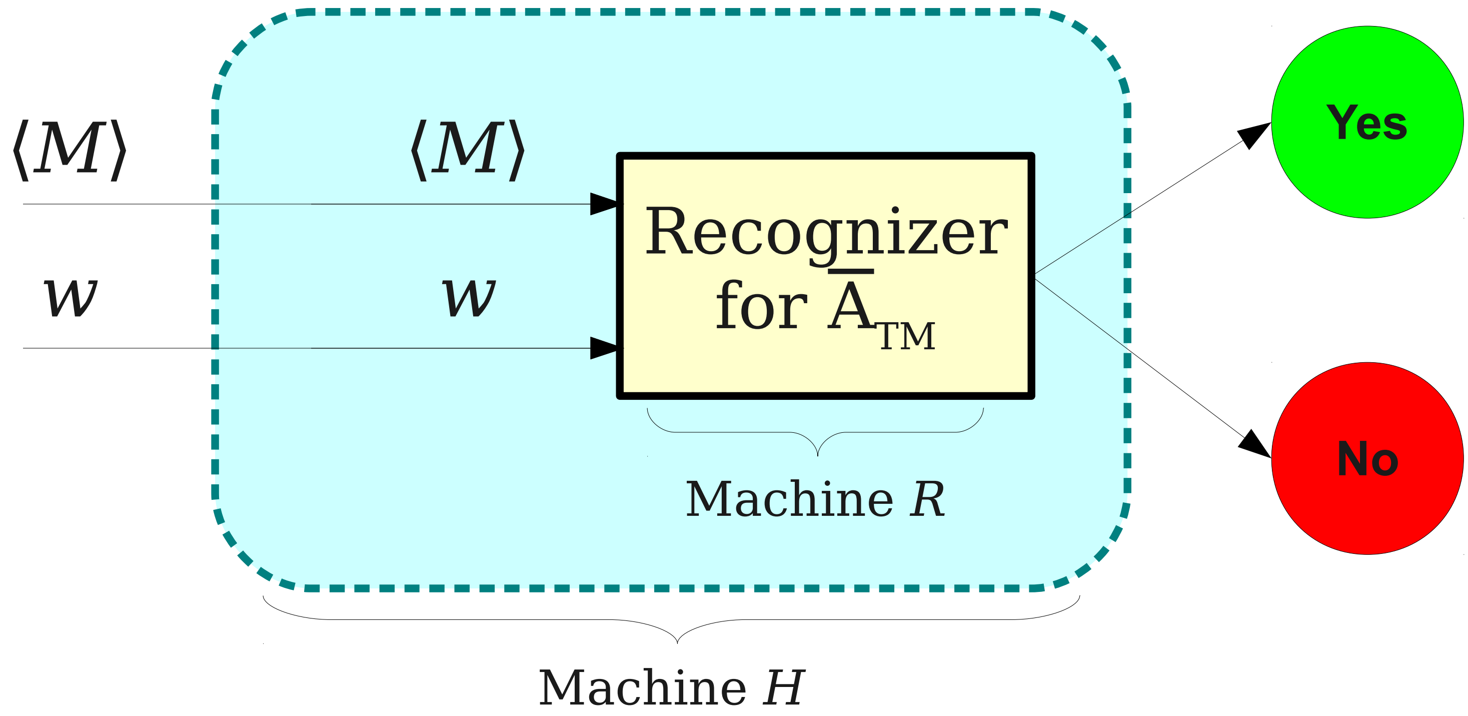


$\langle M \rangle$
w
Recognizer for $\overline{A}_{TM}$
Yes
No

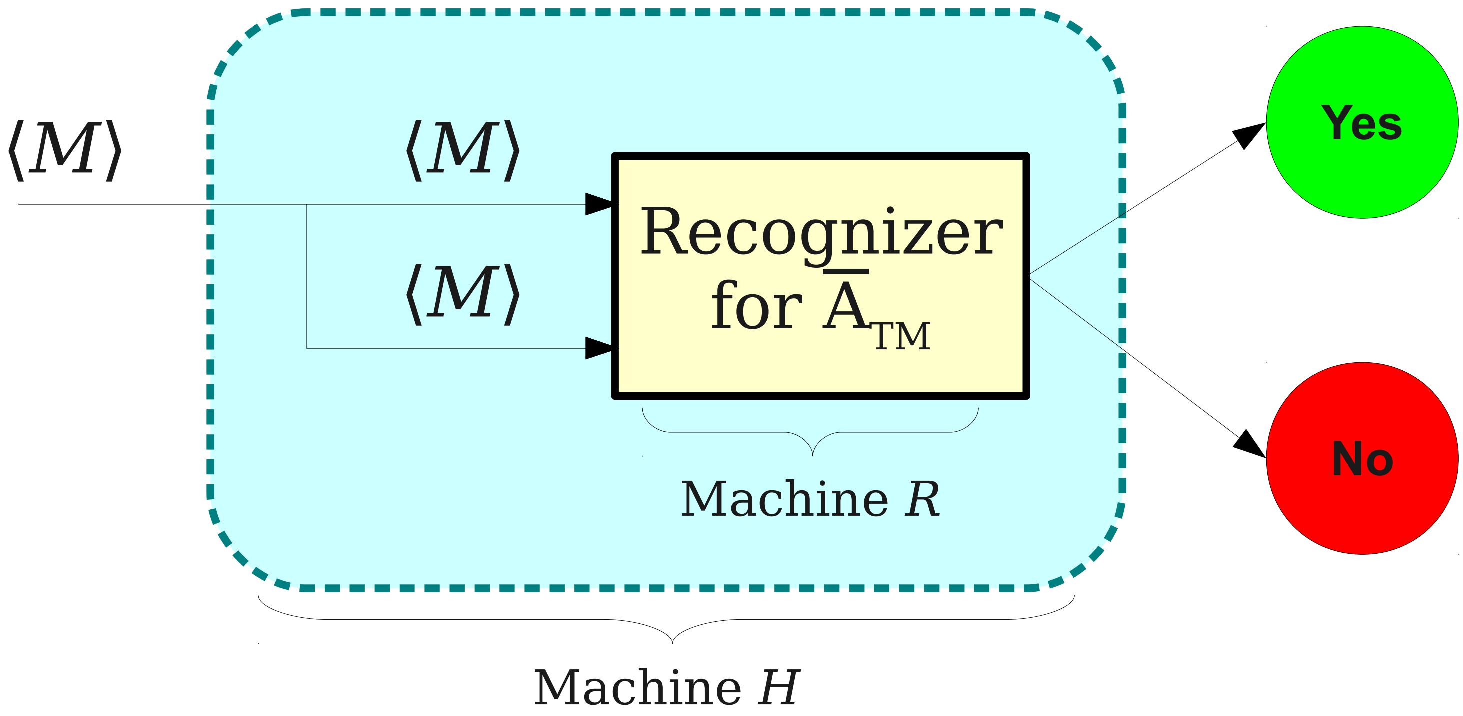

⟨M⟩
w
Recognizer for $\overline{\mathrm{A}}_{\mathrm{TM}}$
Machine R
Yes
No



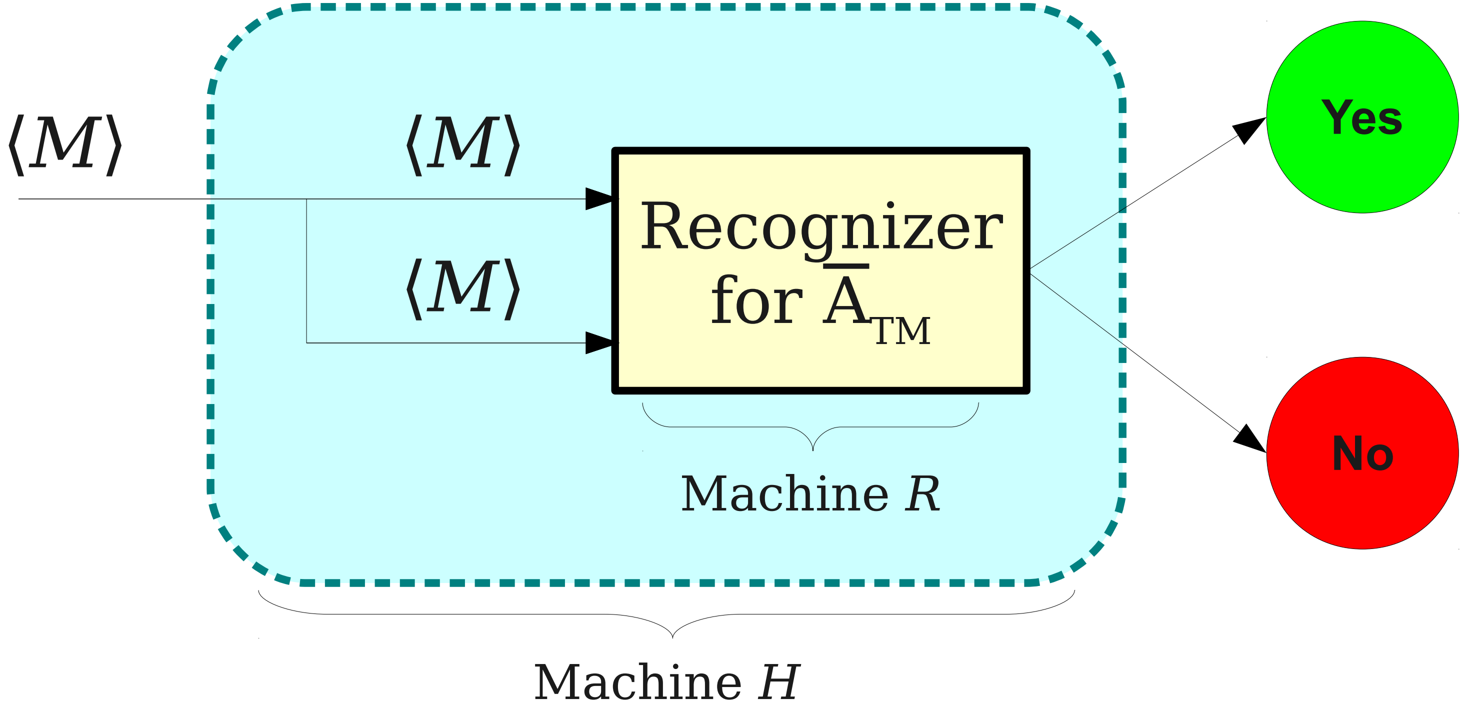
$\langle M \rangle$
w
$\langle M \rangle$
w
Recognizer for $\overline{\mathrm{A}}_{\mathrm{TM}}$
Machine R
Yes
No



$\langle M \rangle$
w
$\langle M \rangle$
w
Recognizer for $\overline{\mathrm{A}}_{\mathrm{TM}}$
Machine R
Machine H
Yes
No

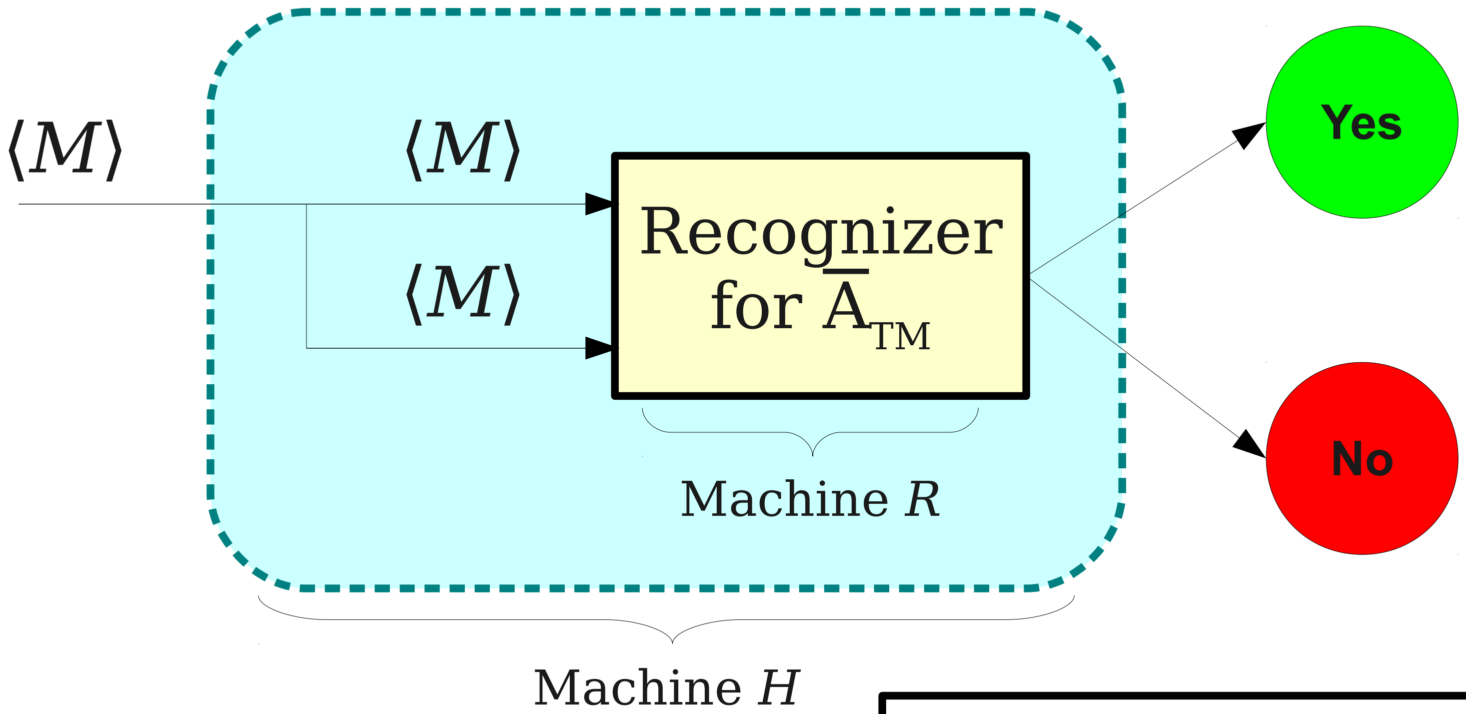


⟨M⟩
⟨M⟩
⟨M⟩
Recognizer for $\overline{A}_{TM}$
Machine R
Machine H
Yes
No



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

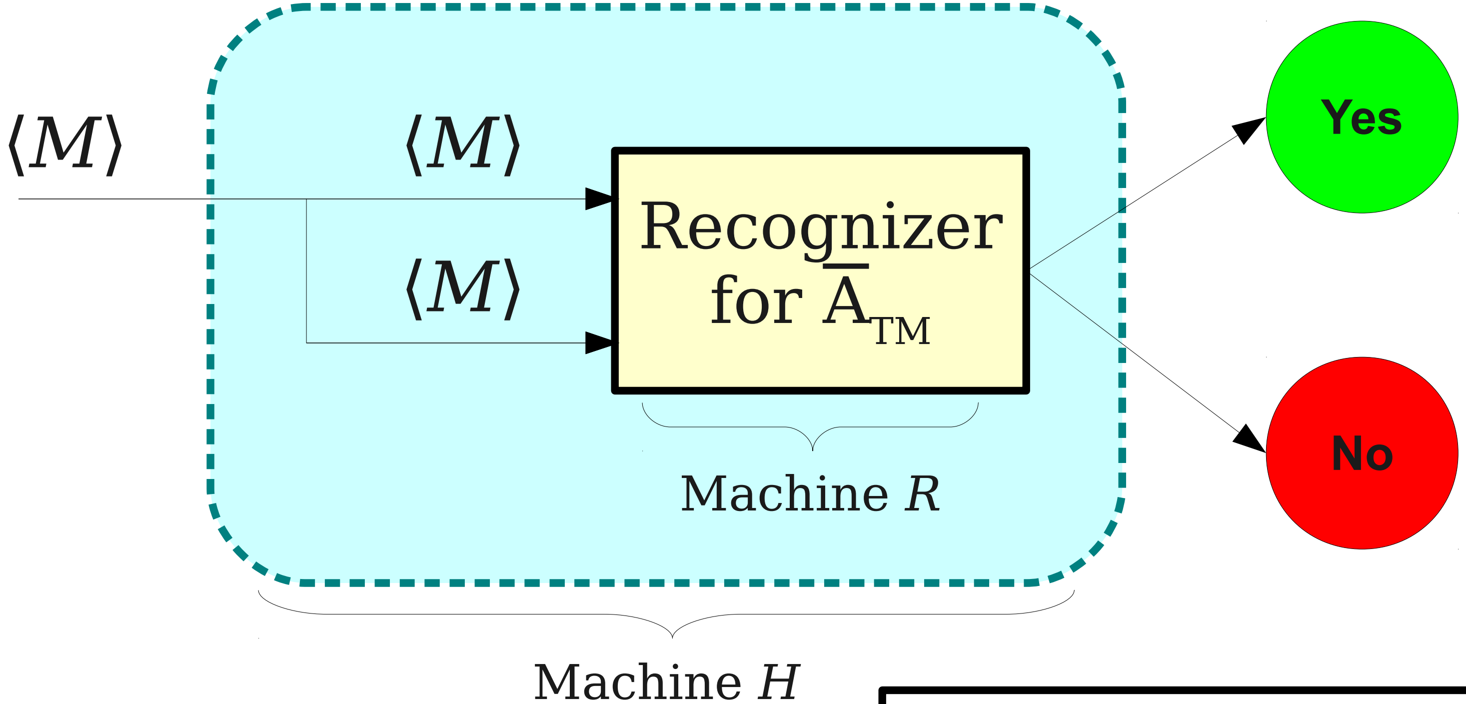
- Construct the string $\langle M, \langle M \rangle \rangle$.
- Run R on $\langle M, \langle M \rangle \rangle$.
- If R accepts $\langle M, \langle M \rangle \rangle$, then H accepts $\langle M \rangle$.
- If R rejects $\langle M, \langle M \rangle \rangle$, then H rejects $\langle M \rangle$."



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 $\langle M \rangle$.”

What happens if...

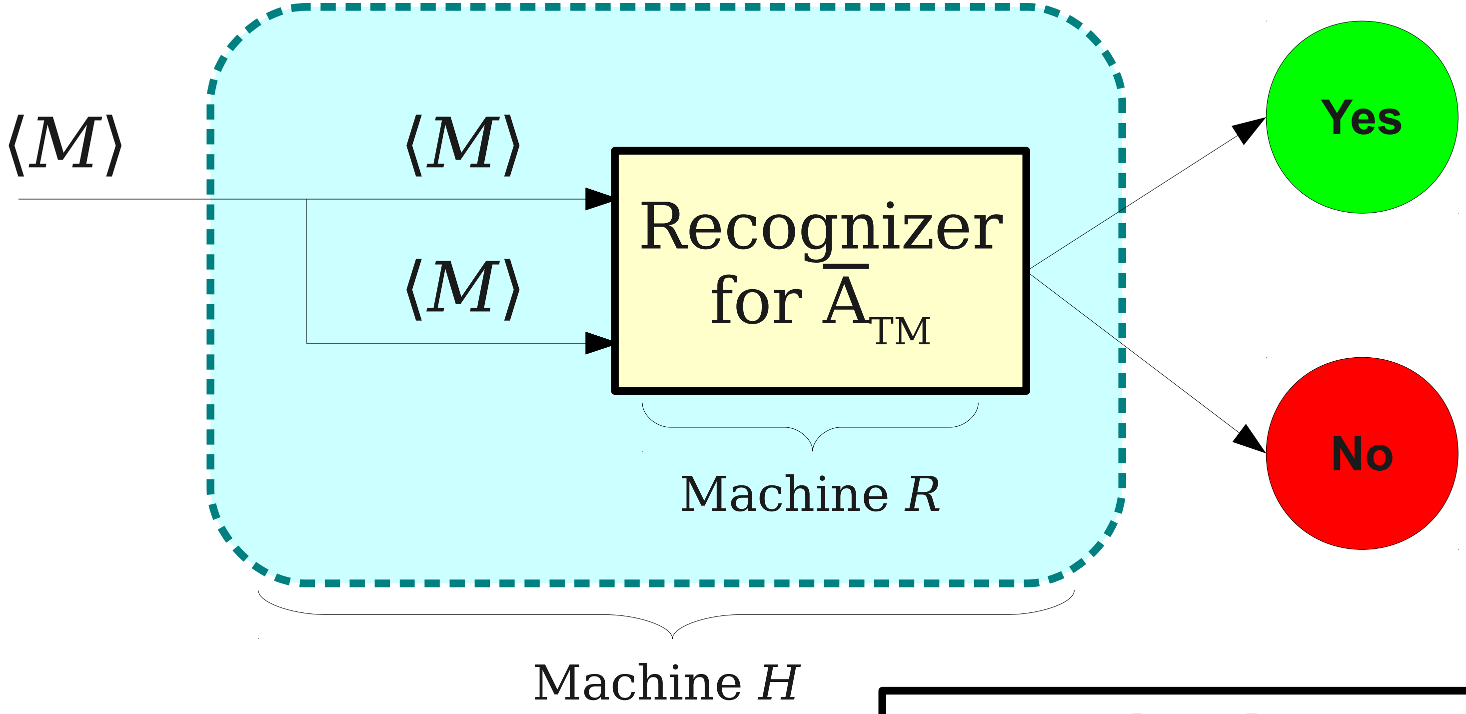


H = "On input $\langle M \rangle$:

- 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 $\langle M \rangle$."

What happens if...

M does not accept $\langle M \rangle$?



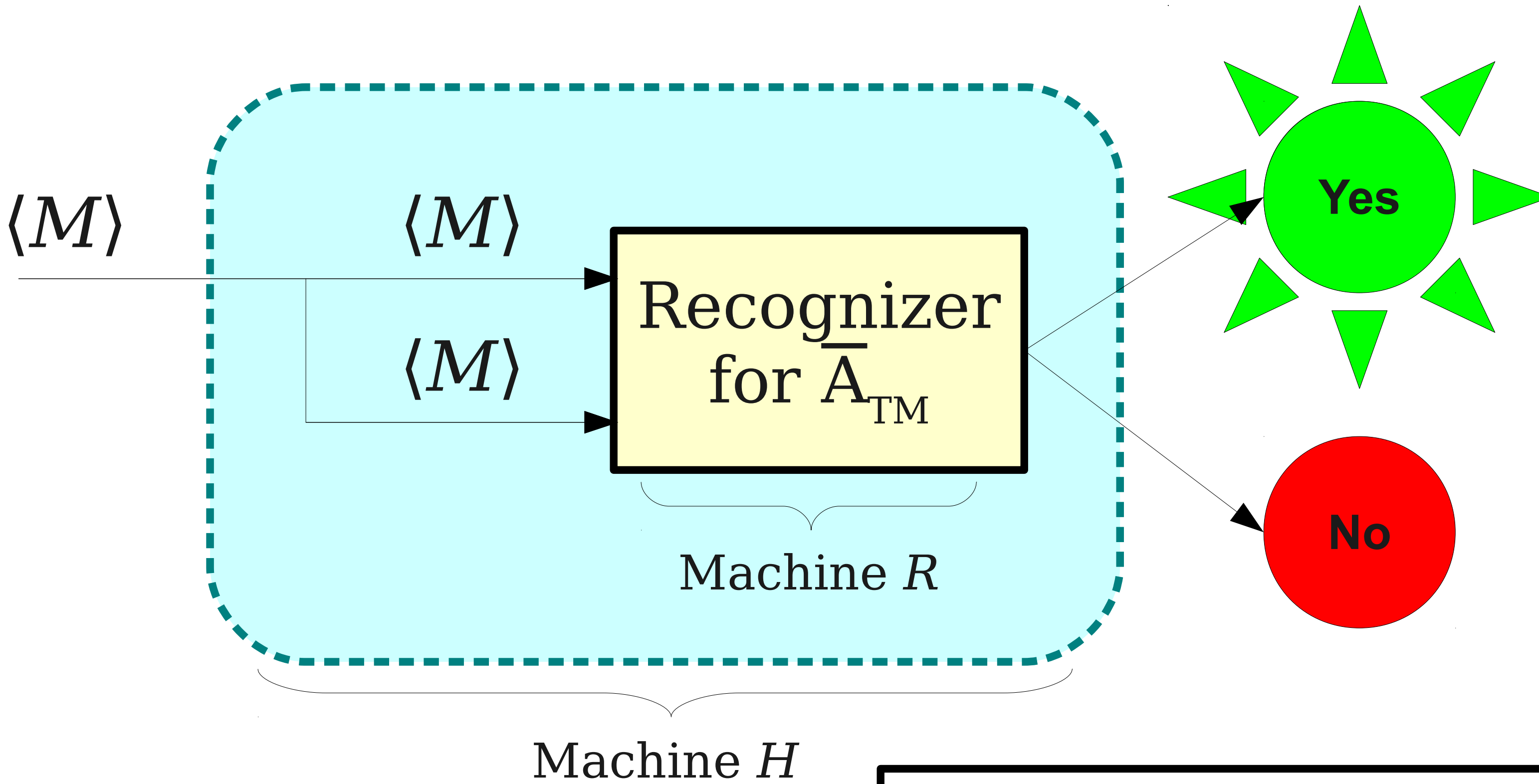
What happens if...

M does not accept $\langle M \rangle$?

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 $\langle M \rangle$."

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



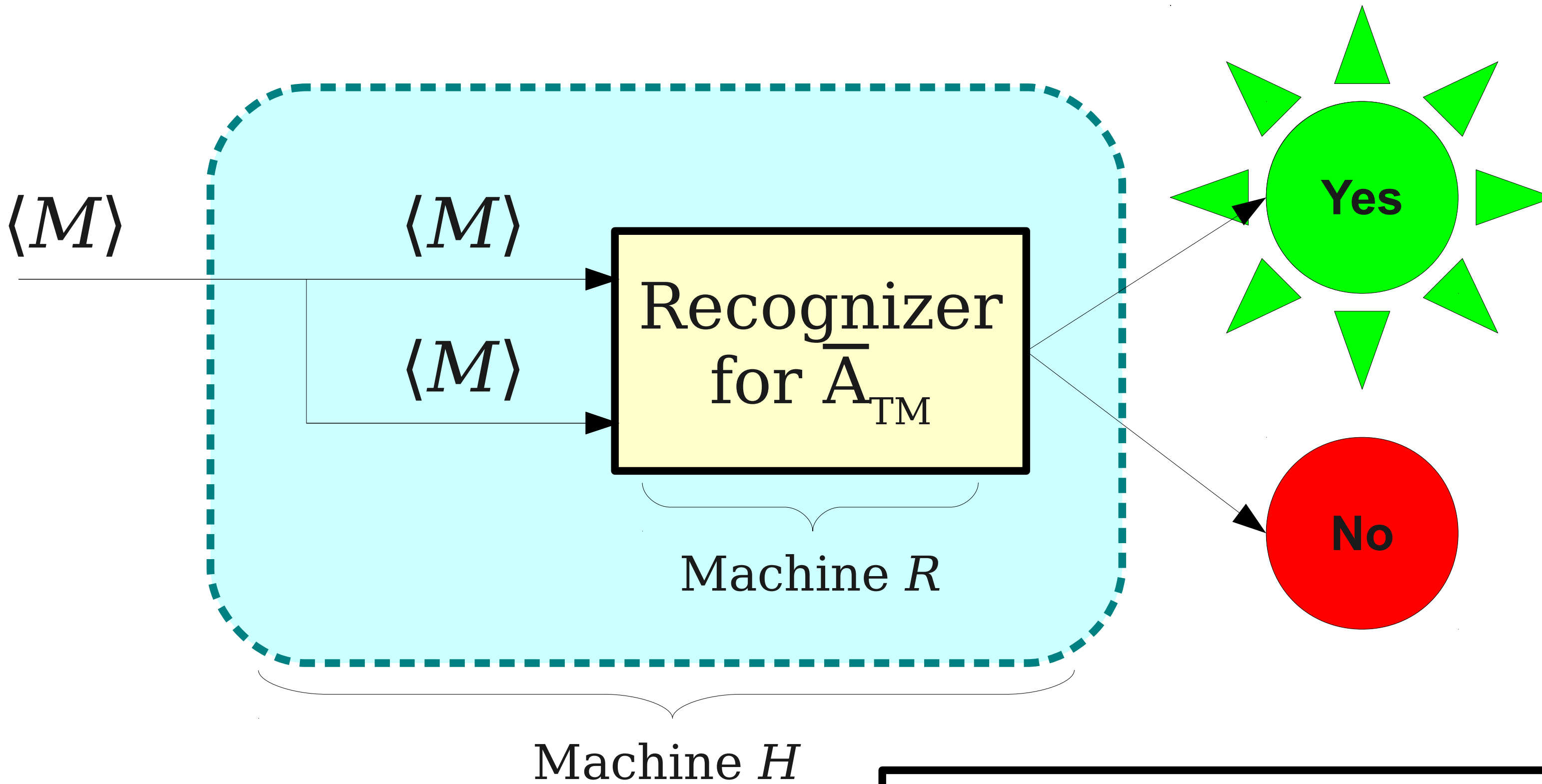
H = "On input $\langle M \rangle$:

- 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 $\langle M \rangle$."

What happens if...

M does not accept $\langle M \rangle$?

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



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

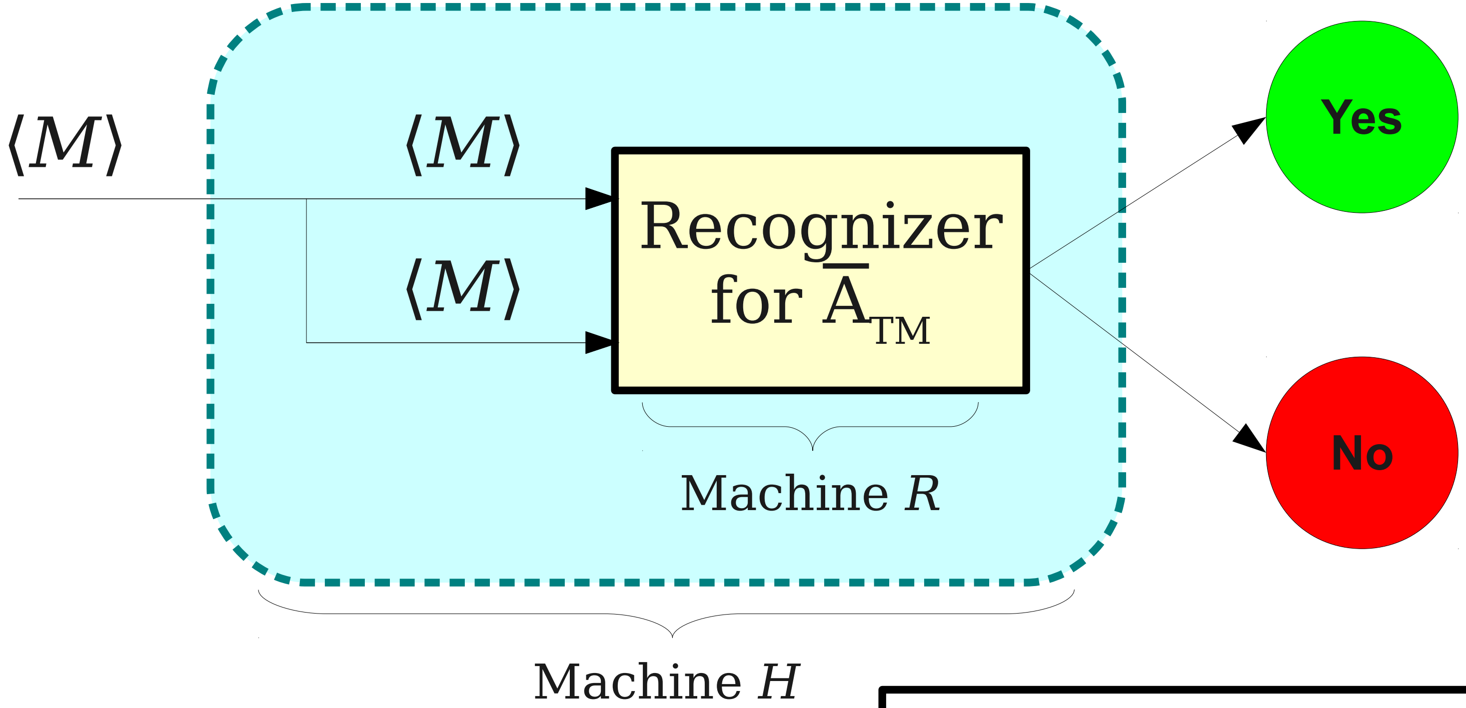
- Construct the string $\langle M, \langle M \rangle \rangle$.
- Run R on $\langle M, \langle M \rangle \rangle$.
- If R accepts $\langle M, \langle M \rangle \rangle$, then H accepts $\langle M \rangle$.
- If R rejects $\langle M, \langle M \rangle \rangle$, then H rejects $\langle M \rangle$."

What happens if...

M does not accept $\langle M \rangle$?

Accept

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



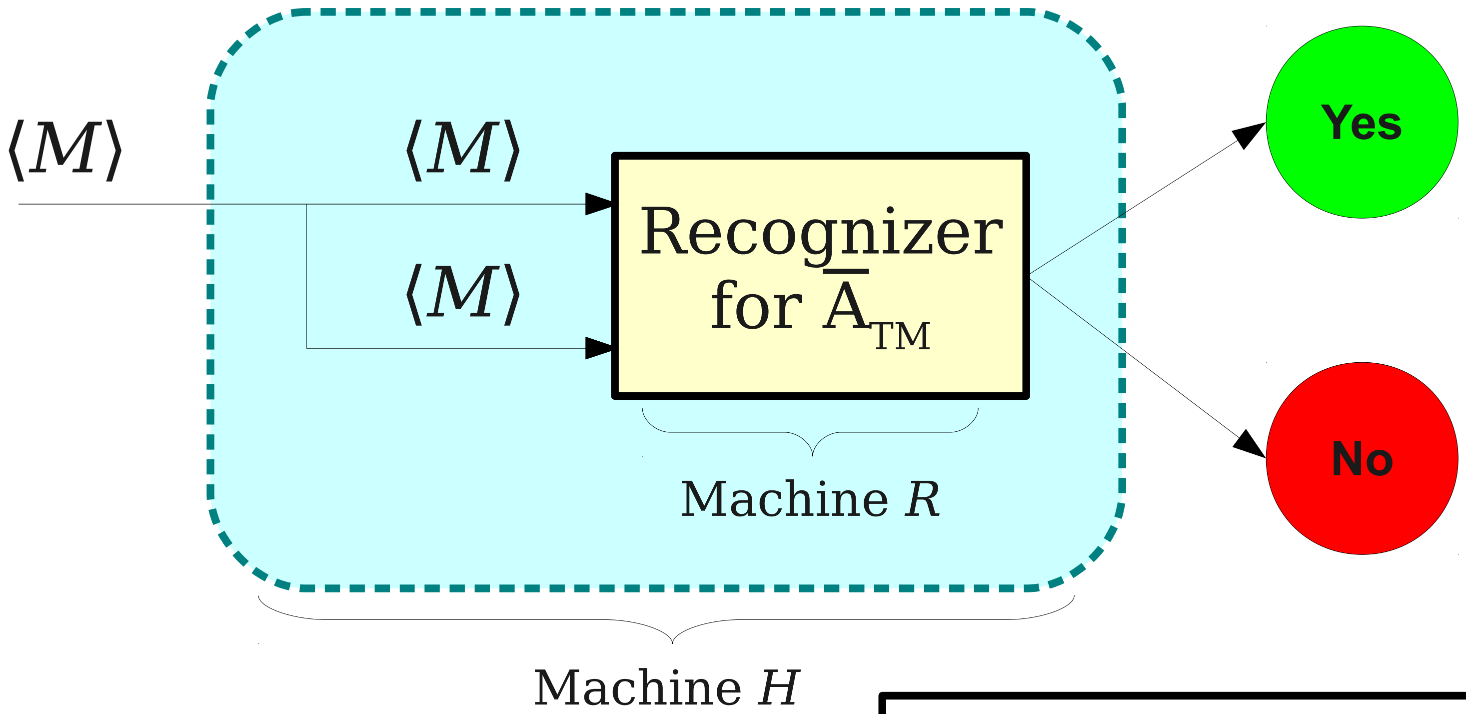
H = "On input $\langle M \rangle$:

- 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 $\langle M \rangle$."

What happens if...

M does not accept $\langle M \rangle$?

Accept



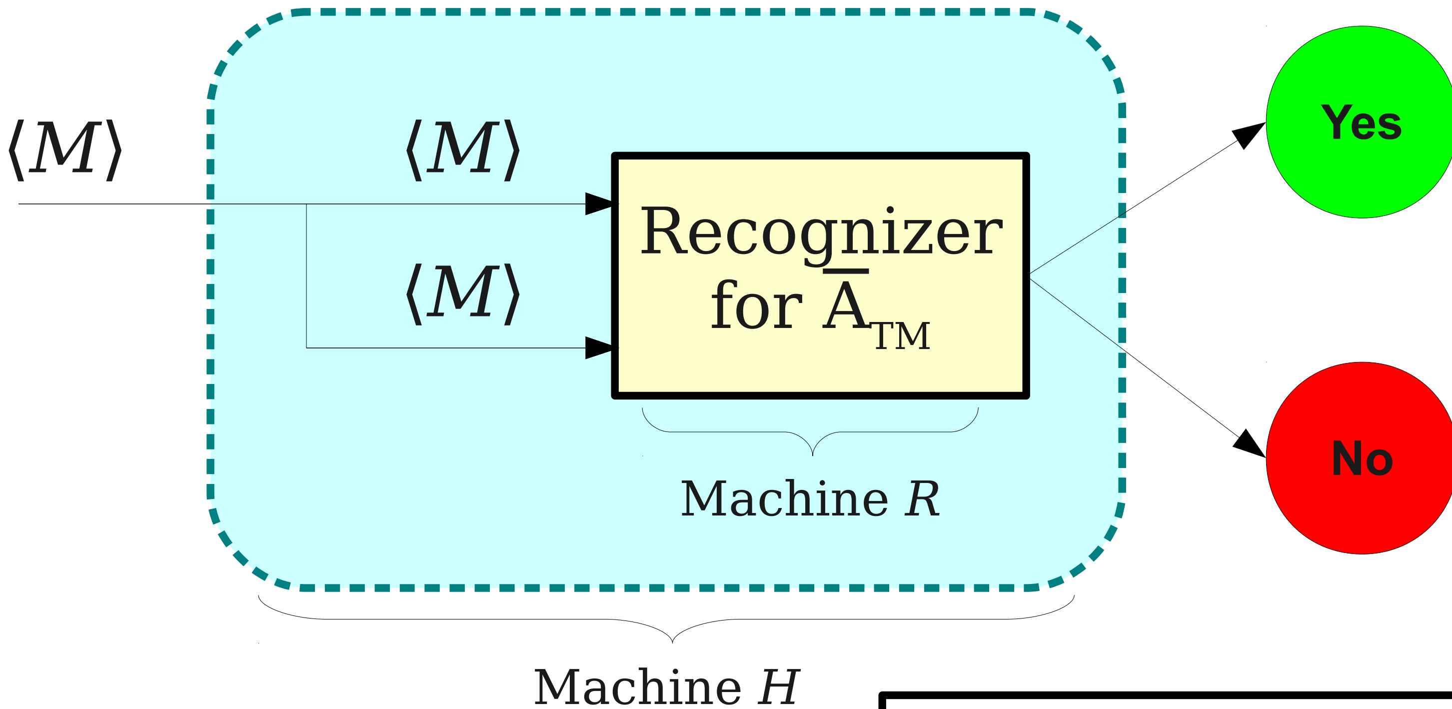
H = "On input ⟨M⟩:

- Construct the string ⟨M, ⟨M⟩⟩.
- Run R on ⟨M, ⟨M⟩⟩.
- If R accepts ⟨M, ⟨M⟩⟩, then H accepts ⟨M⟩.
- If R rejects ⟨M, ⟨M⟩⟩, then H rejects ⟨M⟩."

What happens if...

M does not accept ⟨M⟩?
Accept

M accepts ⟨M⟩?



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

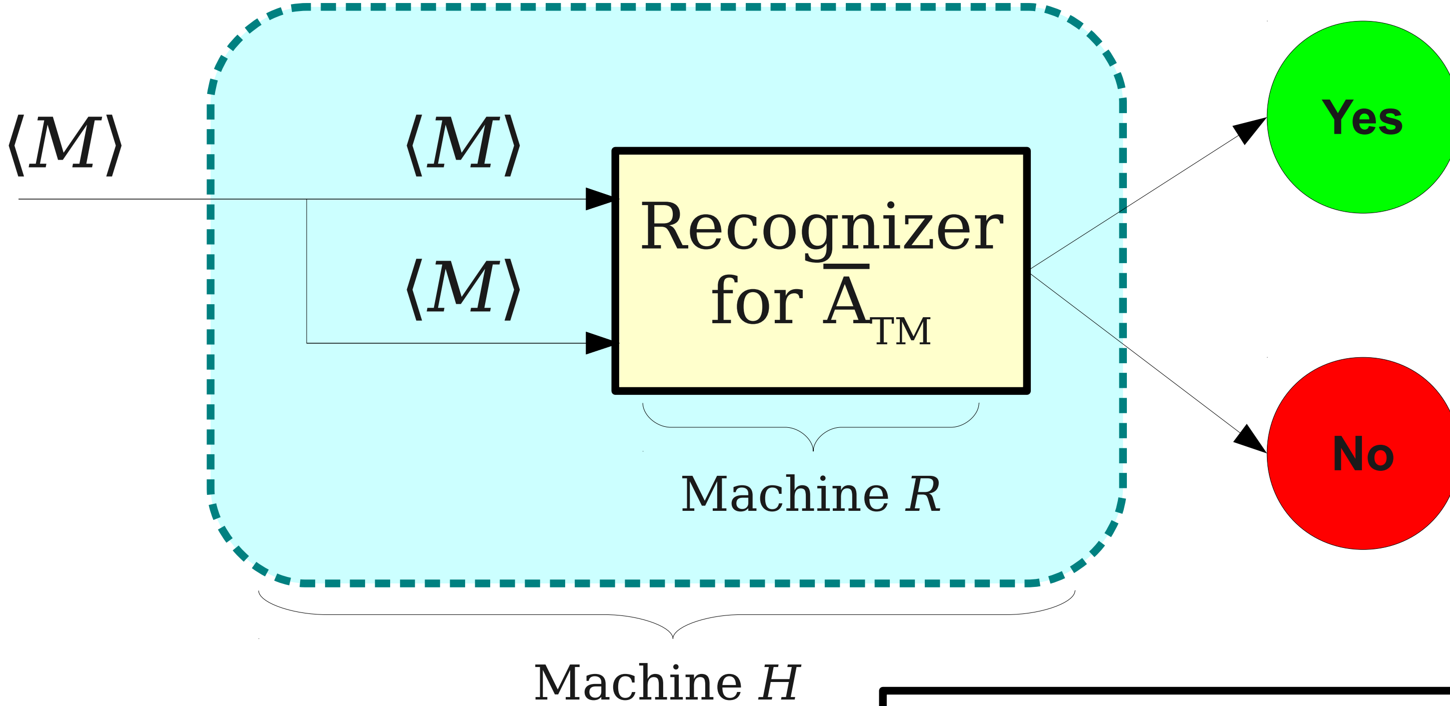
- Construct the string $\langle M, \langle M \rangle \rangle$.
- Run R on $\langle M, \langle M \rangle \rangle$.
- If R accepts $\langle M, \langle M \rangle \rangle$, then H accepts $\langle M \rangle$.
- If R rejects $\langle M, \langle M \rangle \rangle$, then H rejects $\langle M \rangle$."

What happens if...

M does not accept $\langle M \rangle$?
Accept

M accepts $\langle M \rangle$?

Machine R does not accept $\langle M, \langle M \rangle \rangle$



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

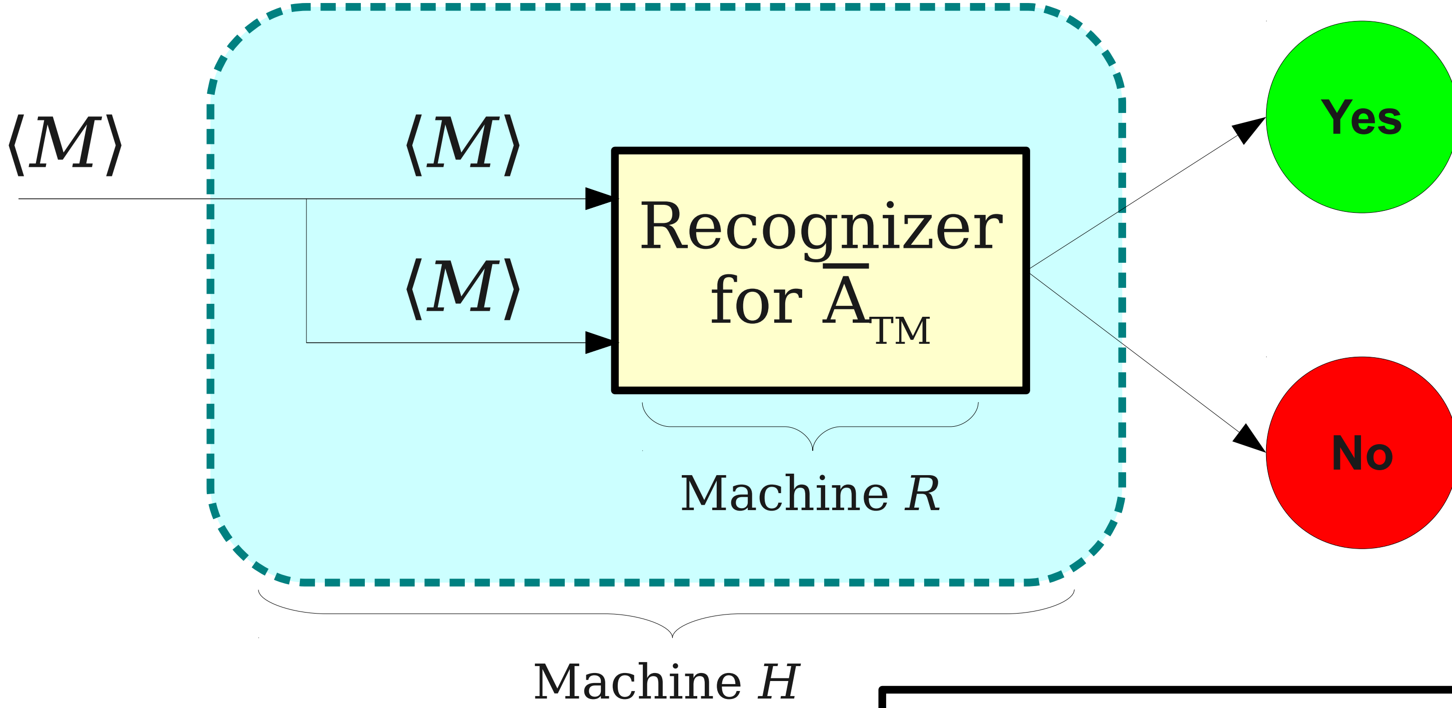
- Construct the string $\langle M, \langle M \rangle \rangle$.
- Run R on $\langle M, \langle M \rangle \rangle$.
- If R accepts $\langle M, \langle M \rangle \rangle$, then H accepts $\langle M \rangle$.
- If R rejects $\langle M, \langle M \rangle \rangle$, then H rejects $\langle M \rangle$."

What happens if...

M does not accept $\langle M \rangle$?
Accept

M accepts $\langle M \rangle$?
Reject or **Loop**

Machine R does not accept $\langle M, \langle M \rangle \rangle$



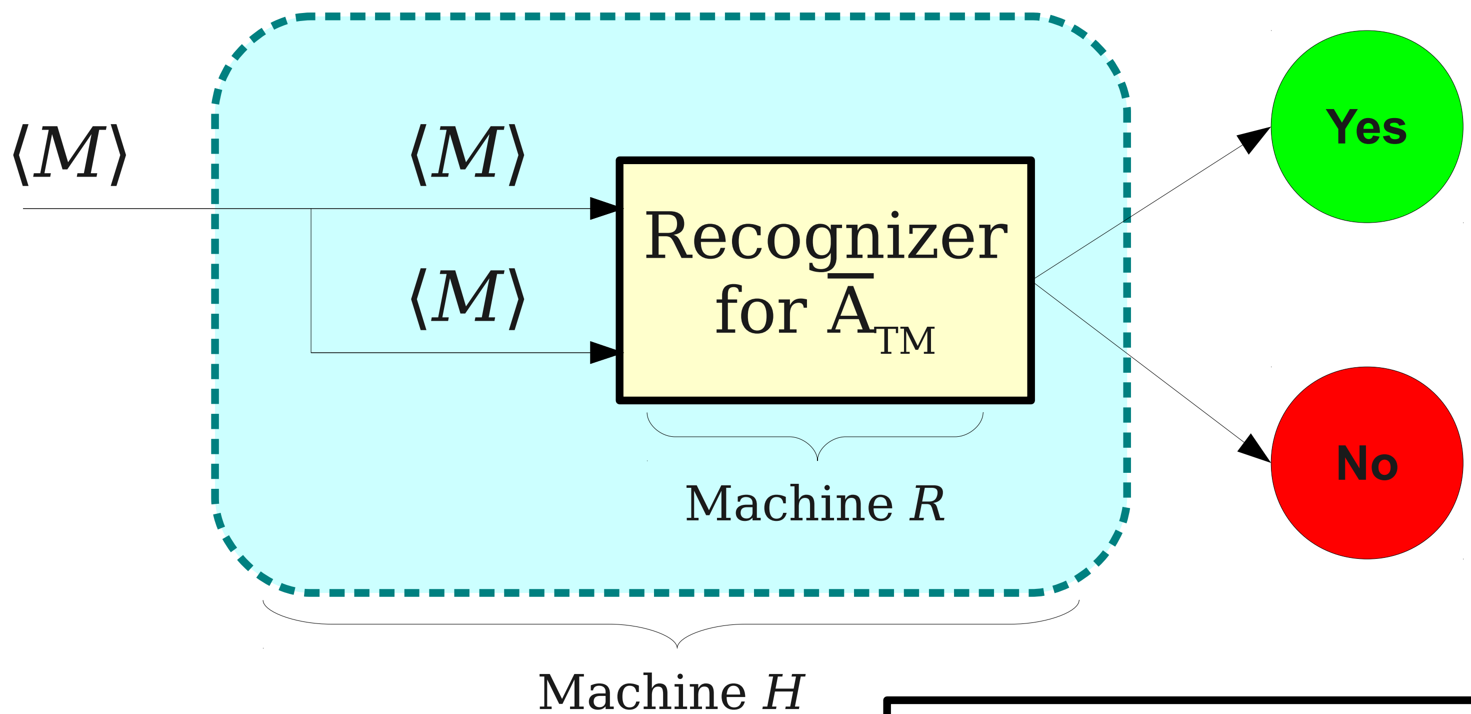
H = "On input $\langle M \rangle$:

- 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 $\langle M \rangle$."

What happens if...

M does not accept $\langle M \rangle$?
Accept

M accepts $\langle M \rangle$?
Reject or **Loop**



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 $\langle M \rangle$."

What happens if...

M does not accept $\langle M \rangle$?
Accept

M accepts $\langle M \rangle$?
Reject or **Loop**

H is a TM for L_D!

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

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

Proof:

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

Proof: By contradiction; assume that $\overline{\mathrm{A}}_{\mathrm{TM}} \in \mathbf{RE}$.

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

Proof: By contradiction; assume that $\overline{\mathrm{A}}_{\mathrm{TM}} \in \mathbf{RE}$. Then there must be a recognizer for $\overline{\mathrm{A}}_{\mathrm{TM}}$; call it R.

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

Proof: By contradiction; assume that $\overline{\mathrm{A}}_{\mathrm{TM}} \in \mathbf{RE}$. Then there must be a recognizer for $\overline{\mathrm{A}}_{\mathrm{TM}}$; call it R.

Consider the TM H defined below:

H = "On input $\langle M \rangle$, where M is a TM:
Construct the string $\langle M, \langle M \rangle \rangle$.
Run R on $\langle M, \langle M \rangle \rangle$.
If R accepts $\langle M, \langle M \rangle \rangle$, H accepts $\langle M \rangle$.
If R rejects $\langle M, \langle M \rangle \rangle$, H rejects $\langle M \rangle$."

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

Proof: By contradiction; assume that $\overline{\mathrm{A}}_{\mathrm{TM}} \in \mathbf{RE}$. Then there must be a recognizer for $\overline{\mathrm{A}}_{\mathrm{TM}}$; call it R.

Consider the TM H defined below:

H = "On input $\langle M \rangle$, where M is a TM:
Construct the string $\langle M, \langle M \rangle \rangle$.
Run R on $\langle M, \langle M \rangle \rangle$.
If R accepts $\langle M, \langle M \rangle \rangle$, H accepts $\langle M \rangle$.
If R rejects $\langle M, \langle M \rangle \rangle$, H rejects $\langle M \rangle$."

We claim that $\mathscr{L}(H) = L_{\mathrm{D}}$.

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

Proof: By contradiction; assume that $\overline{\mathrm{A}}_{\mathrm{TM}} \in \mathbf{RE}$. Then there must be a recognizer for $\overline{\mathrm{A}}_{\mathrm{TM}}$; call it R.

Consider the TM H defined below:

H = "On input $\langle M \rangle$, where M is a TM:
Construct the string $\langle M, \langle M \rangle \rangle$.
Run R on $\langle M, \langle M \rangle \rangle$.
If R accepts $\langle M, \langle M \rangle \rangle$, H accepts $\langle M \rangle$.
If R rejects $\langle M, \langle M \rangle \rangle$, H rejects $\langle M \rangle$."

We claim that $\mathscr{L}(H) = L_{\mathrm{D}}$. We will prove this by showing that $\langle M \rangle \in L_{\mathrm{D}}$ iff H accepts $\langle M \rangle$.

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

Proof: By contradiction; assume that $\overline{\mathrm{A}}_{\mathrm{TM}} \in \mathbf{RE}$. Then there must be a recognizer for $\overline{\mathrm{A}}_{\mathrm{TM}}$; call it R.

Consider the TM H defined below:

H = "On input $\langle M \rangle$, where M is a TM:
Construct the string $\langle M, \langle M \rangle\rangle$.
Run R on $\langle M, \langle M \rangle\rangle$.
If R accepts $\langle M, \langle M \rangle\rangle$, H accepts $\langle M \rangle$.
If R rejects $\langle M, \langle M \rangle\rangle$, H rejects $\langle M \rangle$."

We claim that $\mathscr{L}(H) = L_{\mathrm{D}}$. We will prove this by showing that $\langle M \rangle \in L_{\mathrm{D}}$ iff H accepts $\langle M \rangle$.

By construction we have that H accepts $\langle M \rangle$ iff R accepts $\langle M, \langle M \rangle\rangle$.

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

Proof: By contradiction; assume that $\overline{\mathrm{A}}_{\mathrm{TM}} \in \mathbf{RE}$. Then there must be a recognizer for $\overline{\mathrm{A}}_{\mathrm{TM}}$; call it R.

Consider the TM H defined below:

H = "On input $\langle M \rangle$, where M is a TM:
Construct the string $\langle M, \langle M \rangle \rangle$.
Run R on $\langle M, \langle M \rangle \rangle$.
If R accepts $\langle M, \langle M \rangle \rangle$, H accepts $\langle M \rangle$.
If R rejects $\langle M, \langle M \rangle \rangle$, H rejects $\langle M \rangle$."

We claim that $\mathscr{L}(H) = L_{\mathrm{D}}$. We will prove this by showing that $\langle M \rangle \in L_{\mathrm{D}}$ iff H accepts $\langle M \rangle$.

By construction we have that H accepts $\langle M \rangle$ iff R accepts $\langle M, \langle M \rangle \rangle$. Since R is a recognizer for $\overline{\mathrm{A}}_{\mathrm{TM}}$, R accepts $\langle M, \langle M \rangle \rangle$ iff M does not accept $\langle M \rangle$.

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

Proof: By contradiction; assume that $\overline{\mathrm{A}}_{\mathrm{TM}} \in \mathbf{RE}$. Then there must be a recognizer for $\overline{\mathrm{A}}_{\mathrm{TM}}$; call it R.

Consider the TM H defined below:

H = "On input $\langle M \rangle$, where M is a TM:
Construct the string $\langle M, \langle M \rangle \rangle$.
Run R on $\langle M, \langle M \rangle \rangle$.
If R accepts $\langle M, \langle M \rangle \rangle$, H accepts $\langle M \rangle$.
If R rejects $\langle M, \langle M \rangle \rangle$, H rejects $\langle M \rangle$."

We claim that $\mathscr{L}(H) = L_{\mathrm{D}}$. We will prove this by showing that $\langle M \rangle \in L_{\mathrm{D}}$ iff H accepts $\langle M \rangle$.

By construction we have that H accepts $\langle M \rangle$ iff R accepts $\langle M, \langle M \rangle \rangle$. Since R is a recognizer for $\overline{\mathrm{A}}_{\mathrm{TM}}$, R accepts $\langle M, \langle M \rangle \rangle$ iff M does not accept $\langle M \rangle$. Finally, note that M does not accept $\langle M \rangle$ iff $\langle M \rangle \in L_{\mathrm{D}}$.

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

Proof: By contradiction; assume that $\overline{\mathrm{A}}_{\mathrm{TM}} \in \mathbf{RE}$. Then there must be a recognizer for $\overline{\mathrm{A}}_{\mathrm{TM}}$; call it R.

Consider the TM H defined below:

H = "On input $\langle M \rangle$, where M is a TM:
Construct the string $\langle M, \langle M \rangle \rangle$.
Run R on $\langle M, \langle M \rangle \rangle$.
If R accepts $\langle M, \langle M \rangle \rangle$, H accepts $\langle M \rangle$.
If R rejects $\langle M, \langle M \rangle \rangle$, H rejects $\langle M \rangle$."

We claim that $\mathscr{L}(H) = L_{\mathrm{D}}$. We will prove this by showing that $\langle M \rangle \in L_{\mathrm{D}}$ iff H accepts $\langle M \rangle$.

By construction we have that H accepts $\langle M \rangle$ iff R accepts $\langle M, \langle M \rangle \rangle$. Since R is a recognizer for $\overline{\mathrm{A}}_{\mathrm{TM}}$, R accepts $\langle M, \langle M \rangle \rangle$ iff M does not accept $\langle M \rangle$. Finally, note that M does not accept $\langle M \rangle$ iff $\langle M \rangle \in L_{\mathrm{D}}$. Therefore, we have H accepts $\langle M \rangle$ iff $\langle M \rangle \in L_{\mathrm{D}}$, so $\mathscr{L}(H) = L_{\mathrm{D}}$.

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

Proof: By contradiction; assume that $\overline{\mathrm{A}}_{\mathrm{TM}} \in \mathbf{RE}$. Then there must be a recognizer for $\overline{\mathrm{A}}_{\mathrm{TM}}$; call it R.

Consider the TM H defined below:

H = "On input $\langle M \rangle$, where M is a TM:
Construct the string $\langle M, \langle M \rangle \rangle$.
Run R on $\langle M, \langle M \rangle \rangle$.
If R accepts $\langle M, \langle M \rangle \rangle$, H accepts $\langle M \rangle$.
If R rejects $\langle M, \langle M \rangle \rangle$, H rejects $\langle M \rangle$."

We claim that $\mathscr{L}(H) = L_{\mathrm{D}}$. We will prove this by showing that $\langle M \rangle \in L_{\mathrm{D}}$ iff H accepts $\langle M \rangle$.

By construction we have that H accepts $\langle M \rangle$ iff R accepts $\langle M, \langle M \rangle \rangle$. Since R is a recognizer for $\overline{\mathrm{A}}_{\mathrm{TM}}$, R accepts $\langle M, \langle M \rangle \rangle$ iff M does not accept $\langle M \rangle$. Finally, note that M does not accept $\langle M \rangle$ iff $\langle M \rangle \in L_{\mathrm{D}}$. Therefore, we have H accepts $\langle M \rangle$ iff $\langle M \rangle \in L_{\mathrm{D}}$, so $\mathscr{L}(H) = L_{\mathrm{D}}$. But this is impossible, since $L_{\mathrm{D}} \notin \mathbf{RE}$.

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

Proof: By contradiction; assume that $\overline{\mathrm{A}}_{\mathrm{TM}} \in \mathbf{RE}$. Then there must be a recognizer for $\overline{\mathrm{A}}_{\mathrm{TM}}$; call it R.

Consider the TM H defined below:

H = "On input $\langle M \rangle$, where M is a TM:
Construct the string $\langle M, \langle M \rangle \rangle$.
Run R on $\langle M, \langle M \rangle \rangle$.
If R accepts $\langle M, \langle M \rangle \rangle$, H accepts $\langle M \rangle$.
If R rejects $\langle M, \langle M \rangle \rangle$, H rejects $\langle M \rangle$."

We claim that $\mathscr{L}(H) = L_{\mathrm{D}}$. We will prove this by showing that $\langle M \rangle \in L_{\mathrm{D}}$ iff H accepts $\langle M \rangle$.

By construction we have that H accepts $\langle M \rangle$ iff R accepts $\langle M, \langle M \rangle \rangle$. Since R is a recognizer for $\overline{\mathrm{A}}_{\mathrm{TM}}$, R accepts $\langle M, \langle M \rangle \rangle$ iff M does not accept $\langle M \rangle$. Finally, note that M does not accept $\langle M \rangle$ iff $\langle M \rangle \in L_{\mathrm{D}}$. Therefore, we have H accepts $\langle M \rangle$ iff $\langle M \rangle \in L_{\mathrm{D}}$, so $\mathscr{L}(H) = L_{\mathrm{D}}$. But this is impossible, since $L_{\mathrm{D}} \notin \mathbf{RE}$.

We have reached a contradiction, so our assumption must have been incorrect.

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

Proof: By contradiction; assume that $\overline{\mathrm{A}}_{\mathrm{TM}} \in \mathbf{RE}$. Then there must be a recognizer for $\overline{\mathrm{A}}_{\mathrm{TM}}$; call it R.

Consider the TM H defined below:

H = "On input $\langle M \rangle$, where M is a TM:
Construct the string $\langle M, \langle M \rangle\rangle$.
Run R on $\langle M, \langle M \rangle\rangle$.
If R accepts $\langle M, \langle M \rangle\rangle$, H accepts $\langle M \rangle$.
If R rejects $\langle M, \langle M \rangle\rangle$, H rejects $\langle M \rangle$."

We claim that $\mathscr{L}(H) = L_{\mathrm{D}}$. We will prove this by showing that $\langle M \rangle \in L_{\mathrm{D}}$ iff H accepts $\langle M \rangle$.

By construction we have that H accepts $\langle M \rangle$ iff R accepts $\langle M, \langle M \rangle\rangle$. Since R is a recognizer for $\overline{\mathrm{A}}_{\mathrm{TM}}$, R accepts $\langle M, \langle M \rangle\rangle$ iff M does not accept $\langle M \rangle$. Finally, note that M does not accept $\langle M \rangle$ iff $\langle M \rangle \in L_{\mathrm{D}}$. Therefore, we have H accepts $\langle M \rangle$ iff $\langle M \rangle \in L_{\mathrm{D}}$, so $\mathscr{L}(H) = L_{\mathrm{D}}$. But this is impossible, since $L_{\mathrm{D}} \notin \mathbf{RE}$.

We have reached a contradiction, so our assumption must have been incorrect. Thus $\overline{\mathrm{A}}_{\mathrm{TM}} \notin \mathbf{RE}$, as required.

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

Proof: By contradiction; assume that $\overline{\mathrm{A}}_{\mathrm{TM}} \in \mathbf{RE}$. Then there must be a recognizer for $\overline{\mathrm{A}}_{\mathrm{TM}}$; call it R.

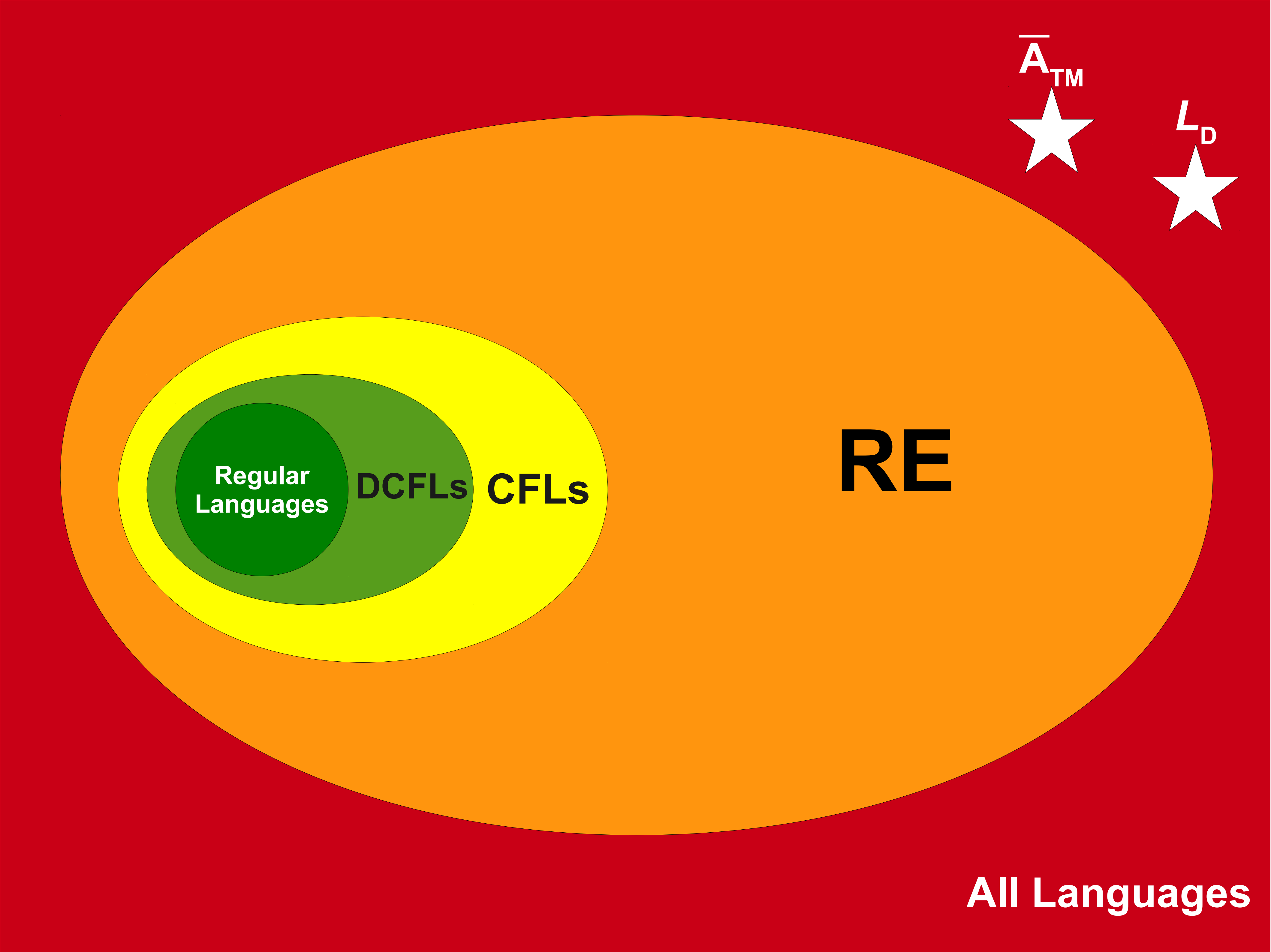
Consider the TM H defined below:

H = "On input $\langle M \rangle$, where M is a TM:
Construct the string $\langle M, \langle M \rangle\rangle$.
Run R on $\langle M, \langle M \rangle\rangle$.
If R accepts $\langle M, \langle M \rangle\rangle$, H accepts $\langle M \rangle$.
If R rejects $\langle M, \langle M \rangle\rangle$, H rejects $\langle M \rangle$."

We claim that $\mathscr{L}(H) = L_{\mathrm{D}}$. We will prove this by showing that $\langle M \rangle \in L_{\mathrm{D}}$ iff H accepts $\langle M \rangle$.

By construction we have that H accepts $\langle M \rangle$ iff R accepts $\langle M, \langle M \rangle\rangle$. Since R is a recognizer for $\overline{\mathrm{A}}_{\mathrm{TM}}$, R accepts $\langle M, \langle M \rangle\rangle$ iff M does not accept $\langle M \rangle$. Finally, note that M does not accept $\langle M \rangle$ iff $\langle M \rangle \in L_{\mathrm{D}}$. Therefore, we have H accepts $\langle M \rangle$ iff $\langle M \rangle \in L_{\mathrm{D}}$, so $\mathscr{L}(H) = L_{\mathrm{D}}$. But this is impossible, since $L_{\mathrm{D}} \notin \mathbf{RE}$.

We have reached a contradiction, so our assumption must have been incorrect. Thus $\overline{\mathrm{A}}_{\mathrm{TM}} \notin \mathbf{RE}$, as required. ■



$\overline{A}_{TM}$
L_D
RE
Regular Languages
DCFLs
CFLs
All Languages

Why All This Matters

- We *finally* have found concrete examples of unsolvable problems!
- We are starting to see a line of reasoning we can use to find unsolvable problems:
  - Start with a known unsolvable problem.
  - Try to show that the unsolvability of that problem entails the unsolvability of other problems.
- We will see this used extensively in the upcoming weeks.

Revisiting **RE**

Recall: Language of a TM

- The language of a Turing machine M, denoted $\mathscr{L}(M)$, is the set of all strings that M accepts:

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

- For any $w \in \mathscr{L}(M)$, M accepts w.
- For any $w \notin \mathscr{L}(M)$, M does not accept w.
  - It might loop forever, or it might explicitly reject.
- A language is called **recognizable** if it is the language of some TM.
- Notation: **RE** is the set of all recognizable languages.

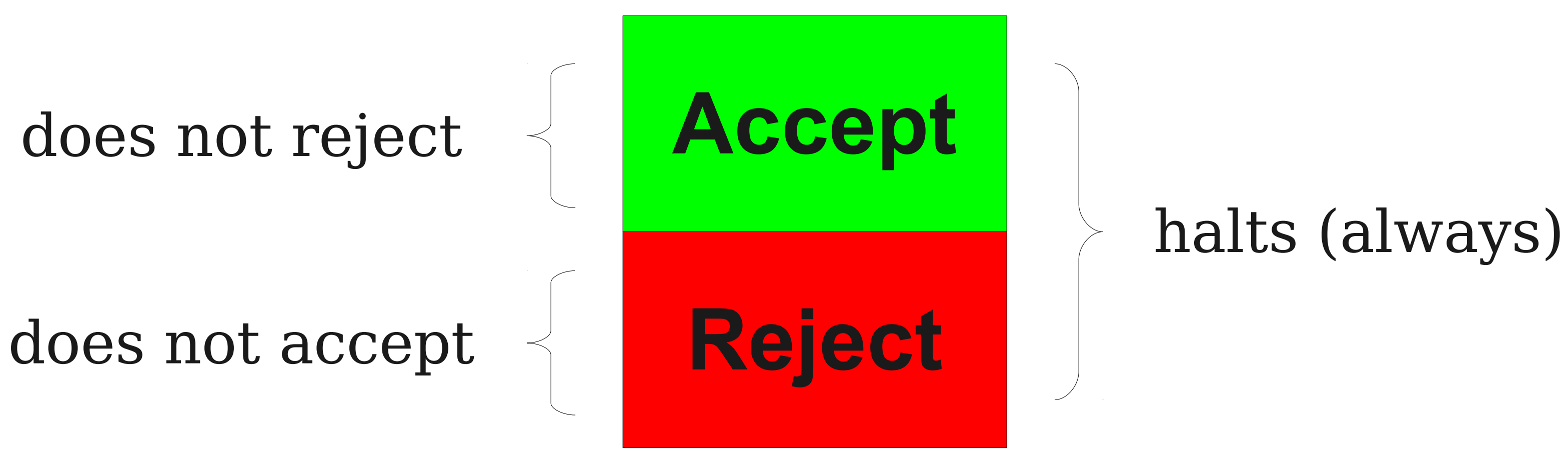
$$L \in \mathbf{RE} \quad \text{iff} \quad L \text{ is recognizable}$$

Why "Recognizable?"

- Given TM M with language $\mathscr{L}(M)$, running M on a string w will not necessarily tell you whether $w \in \mathscr{L}(M)$.
- If the machine is running, you can't tell whether
  - It is eventually going to halt, but just needs more time, or
  - It is never going to halt.
- However, if you know for a fact that $w \in \mathscr{L}(M)$, then the machine can confirm this (it eventually accepts).
- The machine can't *decide* whether or not $w \in \mathscr{L}(M)$, but it can *recognize* strings that are in the language.
- We sometimes call a TM for a language L a **recognizer** for L.

Deciders

- 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.



Decidable Languages

- 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**.

Examples of **R** Languages

- All regular languages are in **R**.
  - If L is regular, we can run the DFA for L on a string w and then either accept or reject w based on what state it ends in.
- $\{ \mathtt{0}^n\mathtt{1}^n \mid n \in \mathbb{N} \}$ is in **R**.
  - The TM we built last Wednesday is a decider.
- Multiplication is in **R**.
  - Can check if $m \times n = p$ by repeatedly subtracting out copies of n. If the equation balances, accept; if not, reject.

CFLs and **R**

- Using an NTM, we sketched a proof that all CFLs are in **RE**.
  - Nondeterministically guess a derivation, then deterministically check that derivation.
- Harder result: all CFLs are in **R**.
  - Read Sipser, Ch. 4.1 for details.
  - Or come talk to me after lecture!

Why **R** Matters

- If a language is in **R**, there is an algorithm that can decide membership in that language.
  - Run the decider and see what it says.
- If there is an algorithm that can decide membership in a language, that language is in **R**.
  - By the Church-Turing thesis, any effective model of computation is equivalent in power to a Turing machine.
  - Thus if there is *any* algorithm for deciding membership in the language, there must be a decider for it.
  - Thus the language is in **R**.
- **A language is in R iff there is an algorithm for deciding membership in that language.**

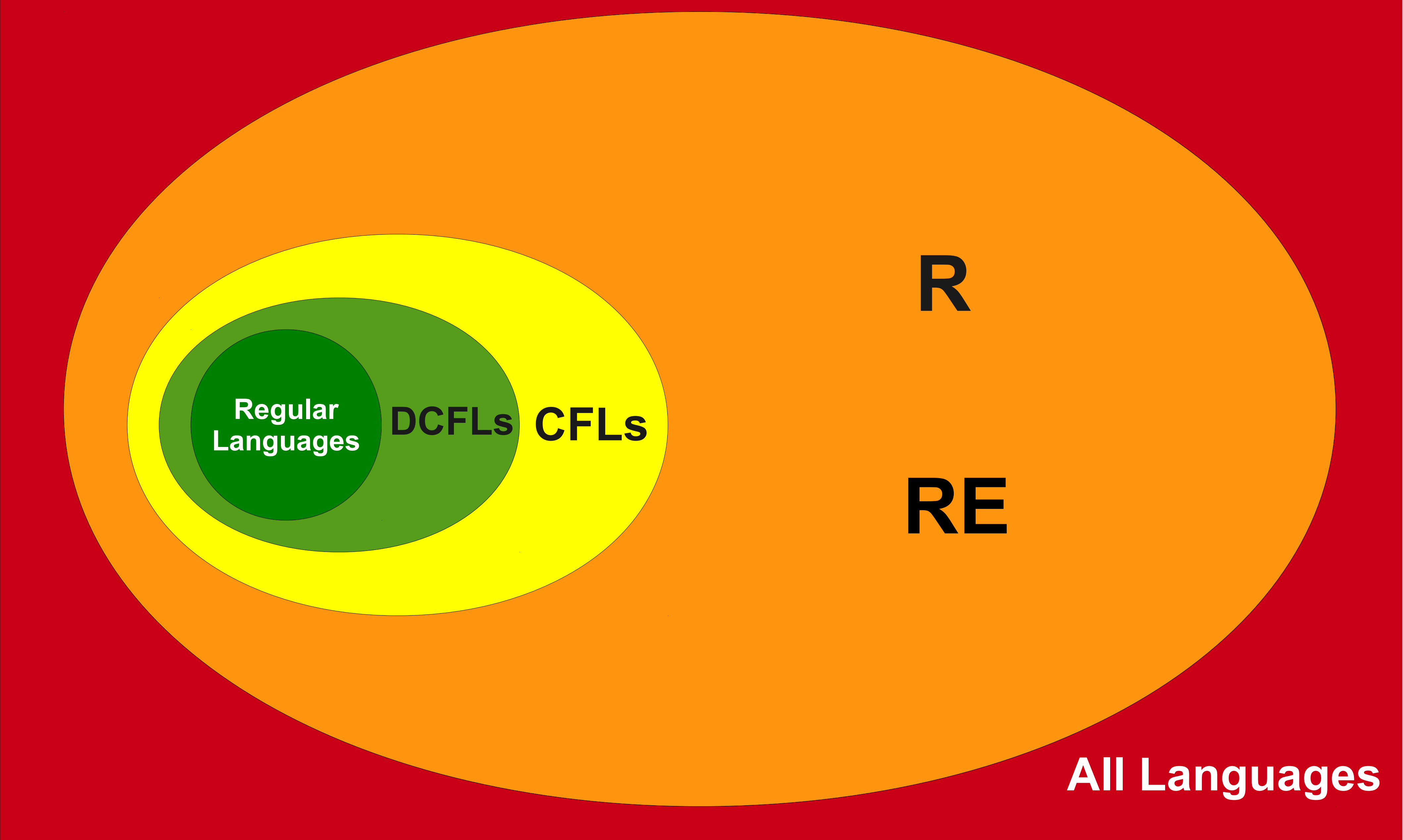
$\mathbf{R} \stackrel{?}{=} \mathbf{RE}$

- Every decider is a Turing machine, but not every Turing machine is a decider.
- Thus $\mathbf{R} \subseteq \mathbf{RE}$.
- Hugely important theoretical question:

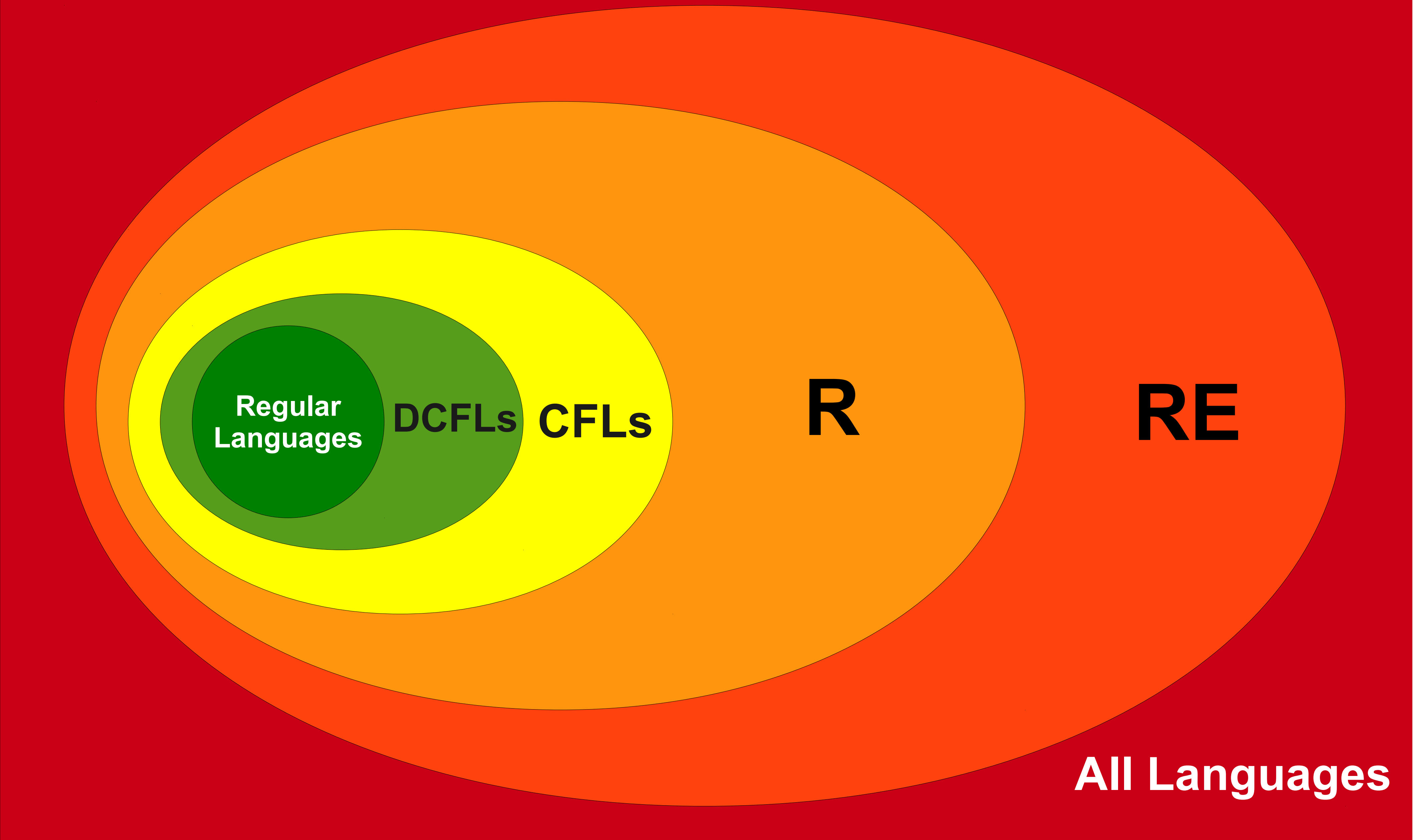
$$\text{Is } \mathbf{R} = \mathbf{RE}?$$

- That is, if we can *verify* that a string is in a language, can we *decide* whether that string is in the language?

Which Picture is Correct?



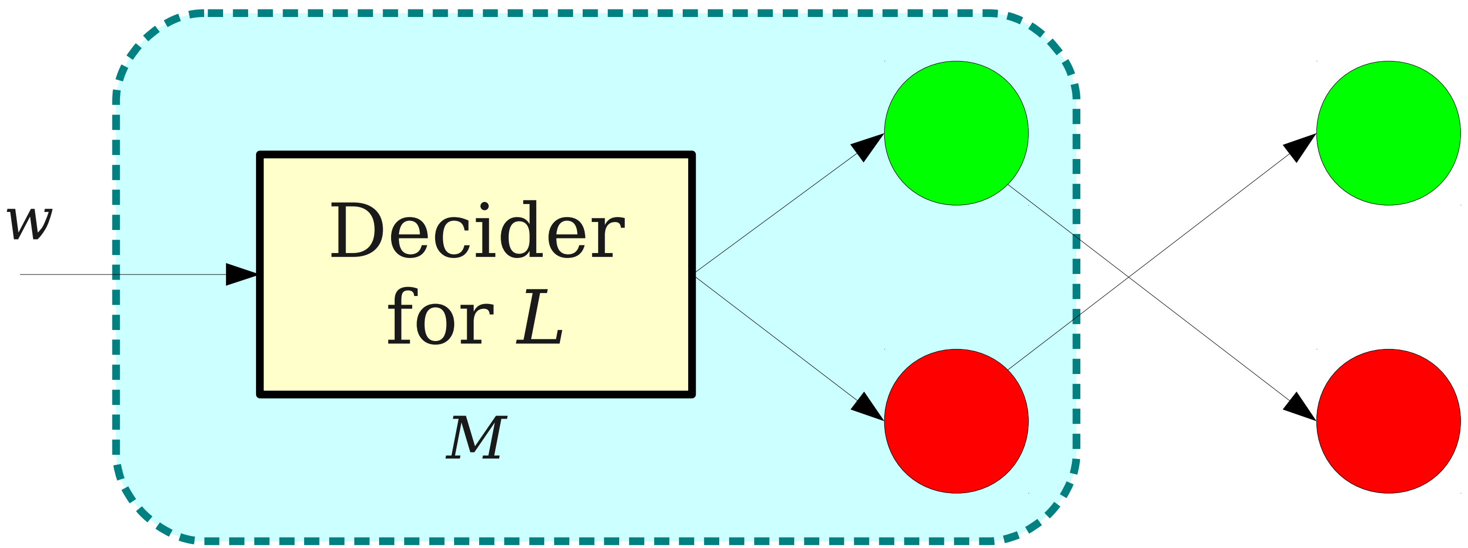
Which Picture is Correct?



An Important Observation

$\mathbf{R}$ is Closed Under Complementation

If $L \in \mathbf{R}$, then $\overline{L} \in \mathbf{R}$ as well.



M' = "On input w:
Run M on w.
If M accepts w, reject.
If M rejects w, accept."

Will this work if M is a **recognizer**, rather than a **decider**?

Theorem: **R** is closed under complementation.

Theorem: $\mathbf{R}$ is closed under complementation.
Proof: Consider any $L \in \mathbf{R}$.

Theorem: **R** is closed under complementation.

Proof: Consider any $L \in \mathbf{R}$. We will prove that $\overline{L} \in \mathbf{R}$ by constructing a decider M' such that $\mathscr{L}(M') = \overline{L}$.

Theorem: **R** is closed under complementation.

Proof: Consider any $L \in \mathbf{R}$. We will prove that $\overline{L} \in \mathbf{R}$ by constructing a decider M' such that $\mathscr{L}(M') = \overline{L}$.

This is the standard way to show that a language is in **R**. Note that we aren't just building any arbitrary TM; it has to be a decider.

Theorem: **R** is closed under complementation.

Proof: Consider any $L \in$ **R**. We will prove that $\overline{L} \in$ **R** by constructing a decider M' such that $\mathscr{L}(M') = \overline{L}$.

Let M be a decider for L.

Theorem: **R** is closed under complementation.

Proof: Consider any $L \in \mathbf{R}$. We will prove that $\overline{L} \in \mathbf{R}$ by constructing a decider M' such that $\mathscr{L}(M') = \overline{L}$.

Let M be a decider for L. Then construct the machine M' as follows:

M' = "On input $w \in \Sigma^*$:

Run M on w.

If M accepts w, reject.

If M rejects w, accept."

Theorem: **R** is closed under complementation.

Proof: Consider any $L \in \mathbf{R}$. We will prove that $\overline{L} \in \mathbf{R}$ by constructing a decider M' such that $\mathscr{L}(M') = \overline{L}$.

Let M be a decider for L. Then construct the machine M' as follows:

M' = "On input $w \in \Sigma^*$:
Run M on w.
If M accepts w, reject.
If M rejects w, accept."

We need to show that M' is a decider and that $\mathscr{L}(M') = \overline{L}$.

Theorem: **R** is closed under complementation.
Proof: Consider any $L \in \mathbf{R}$. We will prove that $\overline{L} \in \mathbf{R}$ by constructing a decider M' such that $\mathscr{L}(M') = \overline{L}$.

Let M be a decider for L. Then construct the machine M' as follows:

M' = "On input $w \in \Sigma^*$:
Run M on w.
If M accepts w, reject.
If M rejects w, accept."

We need to show that M' is a decider and that $\mathscr{L}(M') = \overline{L}$.

There are two proofs required here, and they're separate from one another. Just showing one or the other isn't sufficient.

Theorem: **R** is closed under complementation.

Proof: Consider any $L \in \mathbf{R}$. We will prove that $\overline{L} \in \mathbf{R}$ by constructing a decider M' such that $\mathscr{L}(M') = \overline{L}$.

Let M be a decider for L. Then construct the machine M' as follows:

M' = "On input $w \in \Sigma^*$:

Run M on w.

If M accepts w, reject.

If M rejects w, accept."

We need to show that M' is a decider and that $\mathscr{L}(M') = \overline{L}$.

To show that M' is a decider, we will prove that it always halts.

Theorem: $\mathbf{R}$ is closed under complementation.

Proof: Consider any $L \in \mathbf{R}$. We will prove that $\overline{L} \in \mathbf{R}$ by constructing a decider M' such that $\mathscr{L}(M') = \overline{L}$.

Let M be a decider for L. Then construct the machine M' as follows:

M' = "On input $w \in \Sigma^*$:
Run M on w.
If M accepts w, reject.
If M rejects w, accept."

We need to show that M' is a decider and that $\mathscr{L}(M') = \overline{L}$.

To show that M' is a decider, we will prove that it always halts. Consider what happens if we run M' on any input w.

Theorem: **R** is closed under complementation.

Proof: Consider any $L \in \mathbf{R}$. We will prove that $\overline{L} \in \mathbf{R}$ by constructing a decider M' such that $\mathscr{L}(M') = \overline{L}$.

Let M be a decider for L. Then construct the machine M' as follows:

M' = "On input $w \in \Sigma^*$:
Run M on w.
If M accepts w, reject.
If M rejects w, accept."

We need to show that M' is a decider and that $\mathscr{L}(M') = \overline{L}$.

To show that M' is a decider, we will prove that it always halts. Consider what happens if we run M' on any input w. First, M' runs M on w.

Theorem: **R** is closed under complementation.

Proof: Consider any $L \in \mathbf{R}$. We will prove that $\overline{L} \in \mathbf{R}$ by constructing a decider M' such that $\mathscr{L}(M') = \overline{L}$.

Let M be a decider for L. Then construct the machine M' as follows:

M' = "On input $w \in \Sigma^*$:
Run M on w.
If M accepts w, reject.
If M rejects w, accept."

We need to show that M' is a decider and that $\mathscr{L}(M') = \overline{L}$.

To show that M' is a decider, we will prove that it always halts. Consider what happens if we run M' on any input w. First, M' runs M on w. Since M is a decider, M either accepts w or rejects w.

Theorem: **R** is closed under complementation.

Proof: Consider any $L \in \mathbf{R}$. We will prove that $\overline{L} \in \mathbf{R}$ by constructing a decider M' such that $\mathscr{L}(M') = \overline{L}$.

Let M be a decider for L. Then construct the machine M' as follows:

M' = "On input $w \in \Sigma^*$:

Run M on w.

If M accepts w, reject.

If M rejects w, accept."

We need to show that M' is a decider and that $\mathscr{L}(M') = \overline{L}$.

To show that M' is a decider, we will prove that it always halts. Consider what happens if we run M' on any input w. First, M' runs M on w. Since M is a decider, M either accepts w or rejects w. If M accepts w, M' rejects w.

Theorem: **R** is closed under complementation.
Proof: Consider any $L \in \mathbf{R}$. We will prove that $\overline{L} \in \mathbf{R}$ by constructing a decider M' such that $\mathscr{L}(M') = \overline{L}$.

Let M be a decider for L. Then construct the machine M' as follows:

M' = "On input $w \in \Sigma^*$:
Run M on w.
If M accepts w, reject.
If M rejects w, accept."

We need to show that M' is a decider and that $\mathscr{L}(M') = \overline{L}$.

To show that M' is a decider, we will prove that it always halts. Consider what happens if we run M' on any input w. First, M' runs M on w. Since M is a decider, M either accepts w or rejects w. If M accepts w, M' rejects w. If M rejects w, M' accepts w.

Theorem: $\mathbf{R}$ is closed under complementation.
Proof: Consider any $L \in \mathbf{R}$. We will prove that $\overline{L} \in \mathbf{R}$ by constructing a decider M' such that $\mathscr{L}(M') = \overline{L}$.

Let M be a decider for L. Then construct the machine M' as follows:

M' = "On input $w \in \Sigma^*$:
Run M on w.
If M accepts w, reject.
If M rejects w, accept."

We need to show that M' is a decider and that $\mathscr{L}(M') = \overline{L}$.

To show that M' is a decider, we will prove that it always halts. Consider what happens if we run M' on any input w. First, M' runs M on w. Since M is a decider, M either accepts w or rejects w. If M accepts w, M' rejects w. If M rejects w, M' accepts w. Thus M' always accepts or rejects, so M' is a decider.

Theorem: **R** is closed under complementation.
Proof: Consider any $L \in \mathbf{R}$. We will prove that $\overline{L} \in \mathbf{R}$ by constructing a decider M' such that $\mathscr{L}(M') = \overline{L}$.

Let M be a decider for L. Then construct the machine M' as follows:

M' = "On input $w \in \Sigma^*$:
Run M on w.
If M accepts w, reject.
If M rejects w, accept."

We need to show that M' is a decider and that $\mathscr{L}(M') = \overline{L}$.

To show that M' is a decider, we will prove that it always halts. Consider what happens if we run M' on any input w. First, M' runs M on w. Since M is a decider, M either accepts w or rejects w. If M accepts w, M' rejects w. If M rejects w, M' accepts w. Thus M' always accepts or rejects, so M' is a decider.

To show that $\mathscr{L}(M') = \overline{L}$, we will prove that M' accepts w iff $w \in \overline{L}$.

Theorem: **R** is closed under complementation.

Proof: Consider any $L \in \mathbf{R}$. We will prove that $\overline{L} \in \mathbf{R}$ by constructing a decider M' such that $\mathscr{L}(M') = \overline{L}$.

Let M be a decider for L. Then construct the machine M' as follows:

M' = "On input $w \in \Sigma^*$:

Run M on w.

If M accepts w, reject.

If M rejects w, accept."

We need to show that M' is a decider and that $\mathscr{L}(M') = \overline{L}$.

To show that M' is a decider, we will prove that it always halts. Consider what happens if we run M' on any input w. First, M' runs M on w. Since M is a decider, M either accepts w or rejects w. If M accepts w, M' rejects w. If M rejects w, M' accepts w. Thus M' always accepts or rejects, so M' is a decider.

To show that $\mathscr{L}(M') = \overline{L}$, we will prove that M' accepts w iff $w \in \overline{L}$. Note that M' accepts w iff $w \in \Sigma^*$ and M rejects w.

Theorem: **R** is closed under complementation.

Proof: Consider any $L \in \mathbf{R}$. We will prove that $\overline{L} \in \mathbf{R}$ by constructing a decider M' such that $\mathscr{L}(M') = \overline{L}$.

Let M be a decider for L. Then construct the machine M' as follows:

M' = "On input $w \in \Sigma^*$:
Run M on w.
If M accepts w, reject.
If M rejects w, accept."

We need to show that M' is a decider and that $\mathscr{L}(M') = \overline{L}$.

To show that M' is a decider, we will prove that it always halts. Consider what happens if we run M' on any input w. First, M' runs M on w. Since M is a decider, M either accepts w or rejects w. If M accepts w, M' rejects w. If M rejects w, M' accepts w. Thus M' always accepts or rejects, so M' is a decider.

To show that $\mathscr{L}(M') = \overline{L}$, we will prove that M' accepts w iff $w \in \overline{L}$. Note that M' accepts w iff $w \in \Sigma^*$ and M rejects w. Since M is a decider, M rejects w iff M does not accept w.

Theorem: **R** is closed under complementation.
Proof: Consider any $L \in \mathbf{R}$. We will prove that $\overline{L} \in \mathbf{R}$ by constructing a decider M' such that $\mathscr{L}(M') = \overline{L}$.

Let M be a decider for L. Then construct the machine M' as follows:

M' = "On input $w \in \Sigma^*$:
Run M on w.
If M accepts w, reject.
If M rejects w, accept."

We need to show that M' is a decider and that $\mathscr{L}(M') = \overline{L}$.

To show that M' is a decider, we will prove that it always halts. Consider what happens if we run M' on any input w. First, M' runs M on w. Since M is a decider, M either accepts w or rejects w. If M accepts w, M' rejects w. If M rejects w, M' accepts w. Thus M' always accepts or rejects, so M' is a decider.

To show that $\mathscr{L}(M') = \overline{L}$, we will prove that M' accepts w iff $w \in \overline{L}$. Note that M' accepts w iff $w \in \Sigma^*$ and M rejects w. Since M is a decider, M rejects w iff M does not accept w. M does not accept w iff $w \notin \mathscr{L}(M)$.

Theorem: $\mathbf{R}$ is closed under complementation.
Proof: Consider any $L \in \mathbf{R}$. We will prove that $\overline{L} \in \mathbf{R}$ by constructing a decider M' such that $\mathscr{L}(M') = \overline{L}$.

Let M be a decider for L. Then construct the machine M' as follows:

M' = "On input $w \in \Sigma^*$:
Run M on w.
If M accepts w, reject.
If M rejects w, accept."

We need to show that M' is a decider and that $\mathscr{L}(M') = \overline{L}$.

To show that M' is a decider, we will prove that it always halts. Consider what happens if we run M' on any input w. First, M' runs M on w. Since M is a decider, M either accepts w or rejects w. If M accepts w, M' rejects w. If M rejects w, M' accepts w. Thus M' always accepts or rejects, so M' is a decider.

To show that $\mathscr{L}(M') = \overline{L}$, we will prove that M' accepts w iff $w \in \overline{L}$. Note that M' accepts w iff $w \in \Sigma^*$ and M rejects w. Since M is a decider, M rejects w iff M does not accept w. M does not accept w iff $w \notin \mathscr{L}(M)$. Thus M' accepts w iff $w \in \Sigma^*$ and $w \notin \mathscr{L}(M)$, so M' accepts w iff $w \in \overline{L}$.

Theorem: **R** is closed under complementation.
Proof: Consider any $L \in \mathbf{R}$. We will prove that $\overline{L} \in \mathbf{R}$ by constructing a decider M' such that $\mathscr{L}(M') = \overline{L}$.

Let M be a decider for L. Then construct the machine M' as follows:

M' = "On input $w \in \Sigma^*$:
Run M on w.
If M accepts w, reject.
If M rejects w, accept."

We need to show that M' is a decider and that $\mathscr{L}(M') = \overline{L}$.

To show that M' is a decider, we will prove that it always halts. Consider what happens if we run M' on any input w. First, M' runs M on w. Since M is a decider, M either accepts w or rejects w. If M accepts w, M' rejects w. If M rejects w, M' accepts w. Thus M' always accepts or rejects, so M' is a decider.

To show that $\mathscr{L}(M') = \overline{L}$, we will prove that M' accepts w iff $w \in \overline{L}$. Note that M' accepts w iff $w \in \Sigma^*$ and M rejects w. Since M is a decider, M rejects w iff M does not accept w. M does not accept w iff $w \notin \mathscr{L}(M)$. Thus M' accepts w iff $w \in \Sigma^*$ and $w \notin \mathscr{L}(M)$, so M' accepts w iff $w \in \overline{L}$. Therefore, $\mathscr{L}(M') = \overline{L}$.

Theorem: **R** is closed under complementation.

Proof: Consider any $L \in \mathbf{R}$. We will prove that $\overline{L} \in \mathbf{R}$ by constructing a decider M' such that $\mathscr{L}(M') = \overline{L}$.

Let M be a decider for L. Then construct the machine M' as follows:

M' = "On input $w \in \Sigma^*$:
Run M on w.
If M accepts w, reject.
If M rejects w, accept."

We need to show that M' is a decider and that $\mathscr{L}(M') = \overline{L}$.

To show that M' is a decider, we will prove that it always halts. Consider what happens if we run M' on any input w. First, M' runs M on w. Since M is a decider, M either accepts w or rejects w. If M accepts w, M' rejects w. If M rejects w, M' accepts w. Thus M' always accepts or rejects, so M' is a decider.

To show that $\mathscr{L}(M') = \overline{L}$, we will prove that M' accepts w iff $w \in \overline{L}$. Note that M' accepts w iff $w \in \Sigma^*$ and M rejects w. Since M is a decider, M rejects w iff M does not accept w. M does not accept w iff $w \notin \mathscr{L}(M)$. Thus M' accepts w iff $w \in \Sigma^*$ and $w \notin \mathscr{L}(M)$, so M' accepts w iff $w \in \overline{L}$. Therefore, $\mathscr{L}(M') = \overline{L}$.

Since M' is a decider with $\mathscr{L}(M') = \overline{L}$, we have $\overline{L} \in \mathbf{R}$, as required.

Theorem: $\mathbf{R}$ is closed under complementation.

Proof: Consider any $L \in \mathbf{R}$. We will prove that $\overline{L} \in \mathbf{R}$ by constructing a decider M' such that $\mathscr{L}(M') = \overline{L}$.

Let M be a decider for L. Then construct the machine M' as follows:

M' = "On input $w \in \Sigma^*$:
Run M on w.
If M accepts w, reject.
If M rejects w, accept."

We need to show that M' is a decider and that $\mathscr{L}(M') = \overline{L}$.

To show that M' is a decider, we will prove that it always halts. Consider what happens if we run M' on any input w. First, M' runs M on w. Since M is a decider, M either accepts w or rejects w. If M accepts w, M' rejects w. If M rejects w, M' accepts w. Thus M' always accepts or rejects, so M' is a decider.

To show that $\mathscr{L}(M') = \overline{L}$, we will prove that M' accepts w iff $w \in \overline{L}$. Note that M' accepts w iff $w \in \Sigma^*$ and M rejects w. Since M is a decider, M rejects w iff M does not accept w. M does not accept w iff $w \notin \mathscr{L}(M)$. Thus M' accepts w iff $w \in \Sigma^*$ and $w \notin \mathscr{L}(M)$, so M' accepts w iff $w \in \overline{L}$. Therefore, $\mathscr{L}(M') = \overline{L}$.

Since M' is a decider with $\mathscr{L}(M') = \overline{L}$, we have $\overline{L} \in \mathbf{R}$, as required. ■

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

- We can now resolve the question of $\mathbf{R} \stackrel{?}{=} \mathbf{RE}$.
- If $\mathbf{R} = \mathbf{RE}$, we need to show that if there is a recognizer for *any* **RE** language L, there has to be a decider for L.
- If $\mathbf{R} \neq \mathbf{RE}$, we just need to find a single language in **RE** that is not in **R**.

A_{TM}

- Recall: the language A_{TM} is the language of the universal Turing machine U_{TM}.
- Consequently, $A_{TM} \in \mathbf{RE}$.
- Is $A_{TM} \in \mathbf{R}$?

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

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

Proof: By contradiction; assume $A_{TM} \in \mathbf{R}$.

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

Proof: By contradiction; assume $A_{TM} \in \mathbf{R}$. Since $\mathbf{R}$ is closed under complementation, this means that $\overline{A}_{TM} \in \mathbf{R}$.

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

Proof: By contradiction; assume $A_{TM} \in \mathbf{R}$. Since $\mathbf{R}$ is closed under complementation, this means that $\overline{A_{TM}} \in \mathbf{R}$. Since $\mathbf{R} \subseteq \mathbf{RE}$, this means that $\overline{A_{TM}} \in \mathbf{RE}$.

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

Proof: By contradiction; assume $A_{TM} \in \mathbf{R}$. Since $\mathbf{R}$ is closed under complementation, this means that $\overline{A}_{TM} \in \mathbf{R}$. Since $\mathbf{R} \subseteq \mathbf{RE}$, this means that $\overline{A}_{TM} \in \mathbf{RE}$. But this is impossible, since we know $\overline{A}_{TM} \notin \mathbf{RE}$.

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

Proof: By contradiction; assume $A_{TM} \in \mathbf{R}$. Since $\mathbf{R}$ is closed under complementation, this means that $\overline{A}_{TM} \in \mathbf{R}$. Since $\mathbf{R} \subseteq \mathbf{RE}$, this means that $\overline{A}_{TM} \in \mathbf{RE}$. But this is impossible, since we know $\overline{A}_{TM} \notin \mathbf{RE}$.

We have reached a contradiction, so our assumption must have been incorrect.

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

Proof: By contradiction; assume $A_{TM} \in \mathbf{R}$. Since $\mathbf{R}$ is closed under complementation, this means that $\overline{A}_{TM} \in \mathbf{R}$. Since $\mathbf{R} \subseteq \mathbf{RE}$, this means that $\overline{A}_{TM} \in \mathbf{RE}$. But this is impossible, since we know $\overline{A}_{TM} \notin \mathbf{RE}$.

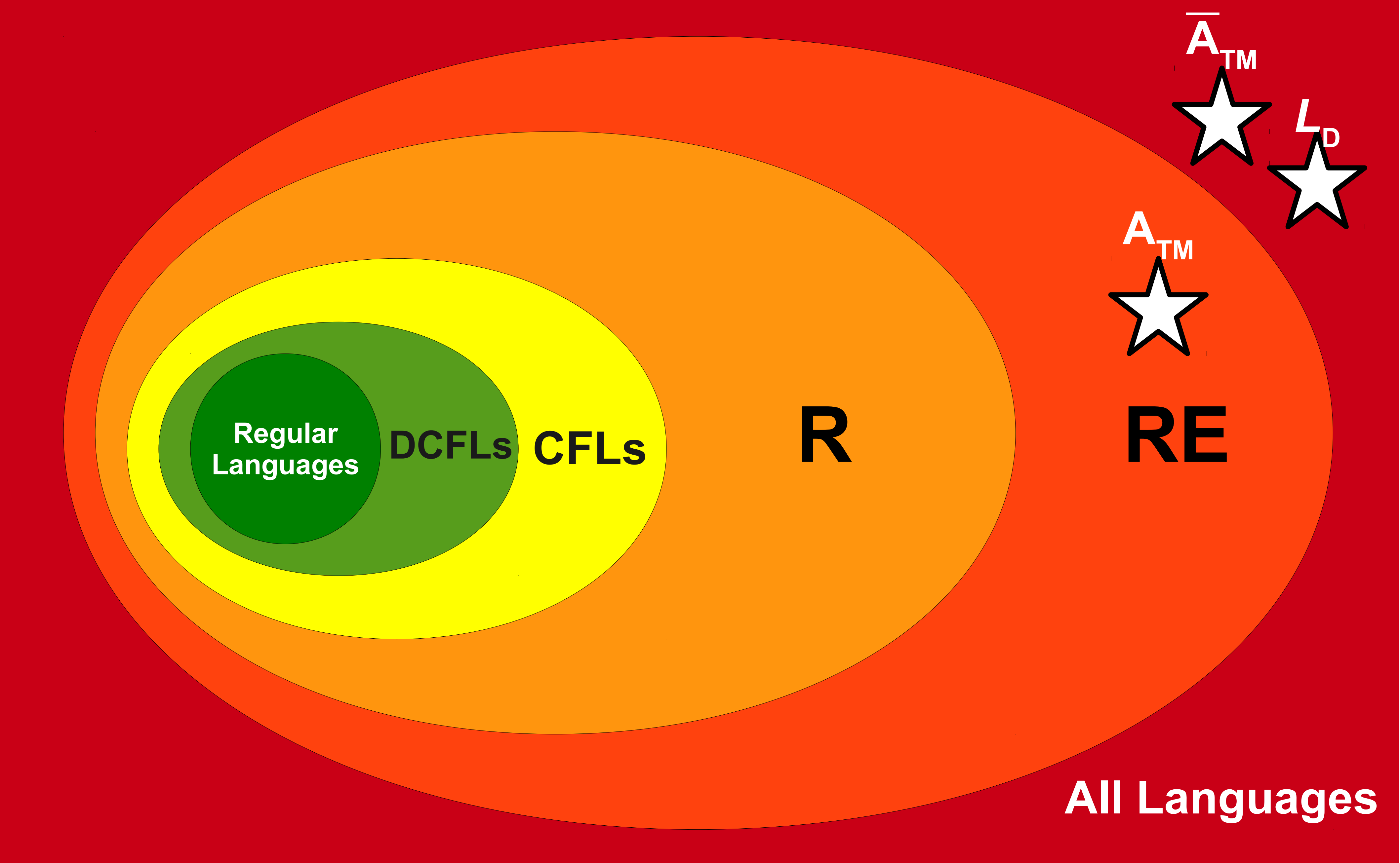
We have reached a contradiction, so our assumption must have been incorrect. Thus $A_{TM} \notin \mathbf{R}$, as required.

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

Proof: By contradiction; assume $A_{TM} \in \mathbf{R}$. Since $\mathbf{R}$ is closed under complementation, this means that $\overline{A}_{TM} \in \mathbf{R}$. Since $\mathbf{R} \subseteq \mathbf{RE}$, this means that $\overline{A}_{TM} \in \mathbf{RE}$. But this is impossible, since we know $\overline{A}_{TM} \notin \mathbf{RE}$.

We have reached a contradiction, so our assumption must have been incorrect. Thus $A_{TM} \notin \mathbf{R}$, as required. ■

The Limits of Computability



What this Means

- The undecidability of A_{TM} means that we cannot “cheat” with Turing machines.
- We cannot necessarily build a TM to do an exhaustive search over a space (i.e. a recognizer), then decide whether it accepts without running it.
- **Intuition:** In most cases, you cannot *decide* what a TM will do without running it to see what happens.
- In some cases, you can *recognize* when a TM has performed some task.
- In some cases, you can't do either. For example, you cannot always recognize that a TM will not accept a string.

What this Means

- **Major result:** **R ≠ RE**.
- There are some problems where we can only give a “yes” answer when the answer is “yes” and cannot necessarily give a yes-or-no answer.
- Solving a problem is *fundamentally harder* than recognizing a correct answer.

Another Undecidable Problem

L_{D} Revisited

- The diagonalization language L_{D} is the language

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

- As we saw before, $L_{\mathrm{D}} \notin \mathbf{RE}$.
- But what about $\overline{L}_{\mathrm{D}}$?

$\overline{L}_{\text{D}}$

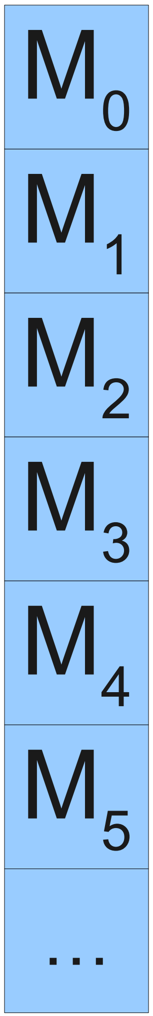
- The language L_{D} is the language

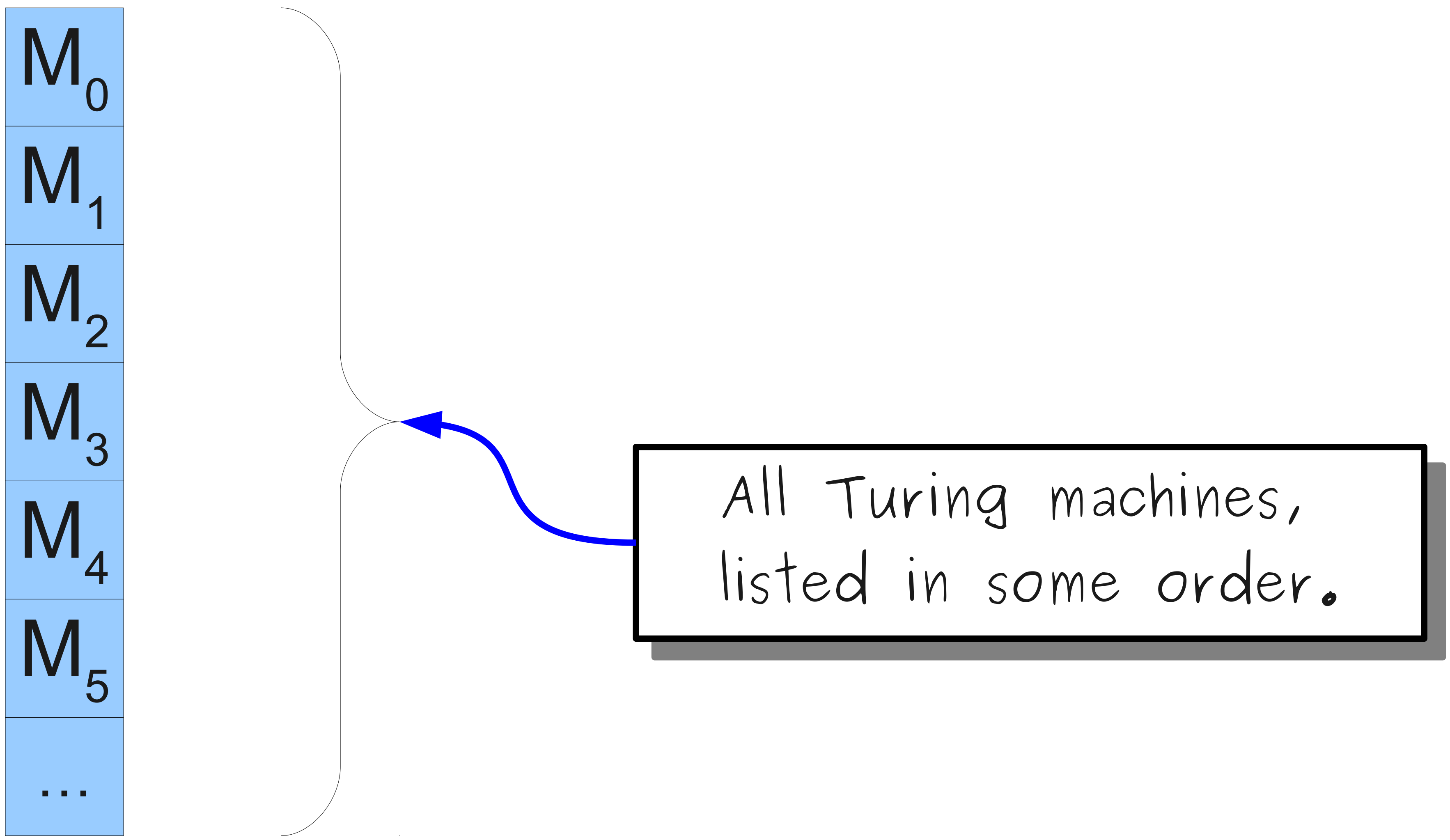
  $\boldsymbol{L_{\text{D}} = \{\langle M\rangle \mid M \textbf{ is a TM and } \langle M\rangle \notin \mathscr{L}(M)\}}$

- Therefore, $\overline{L}_{\text{D}}$ is the language

  $\boldsymbol{\overline{L}_{\text{D}} = \{\langle M\rangle \mid M \textbf{ is a TM and } \langle M\rangle \in \mathscr{L}(M)\}}$

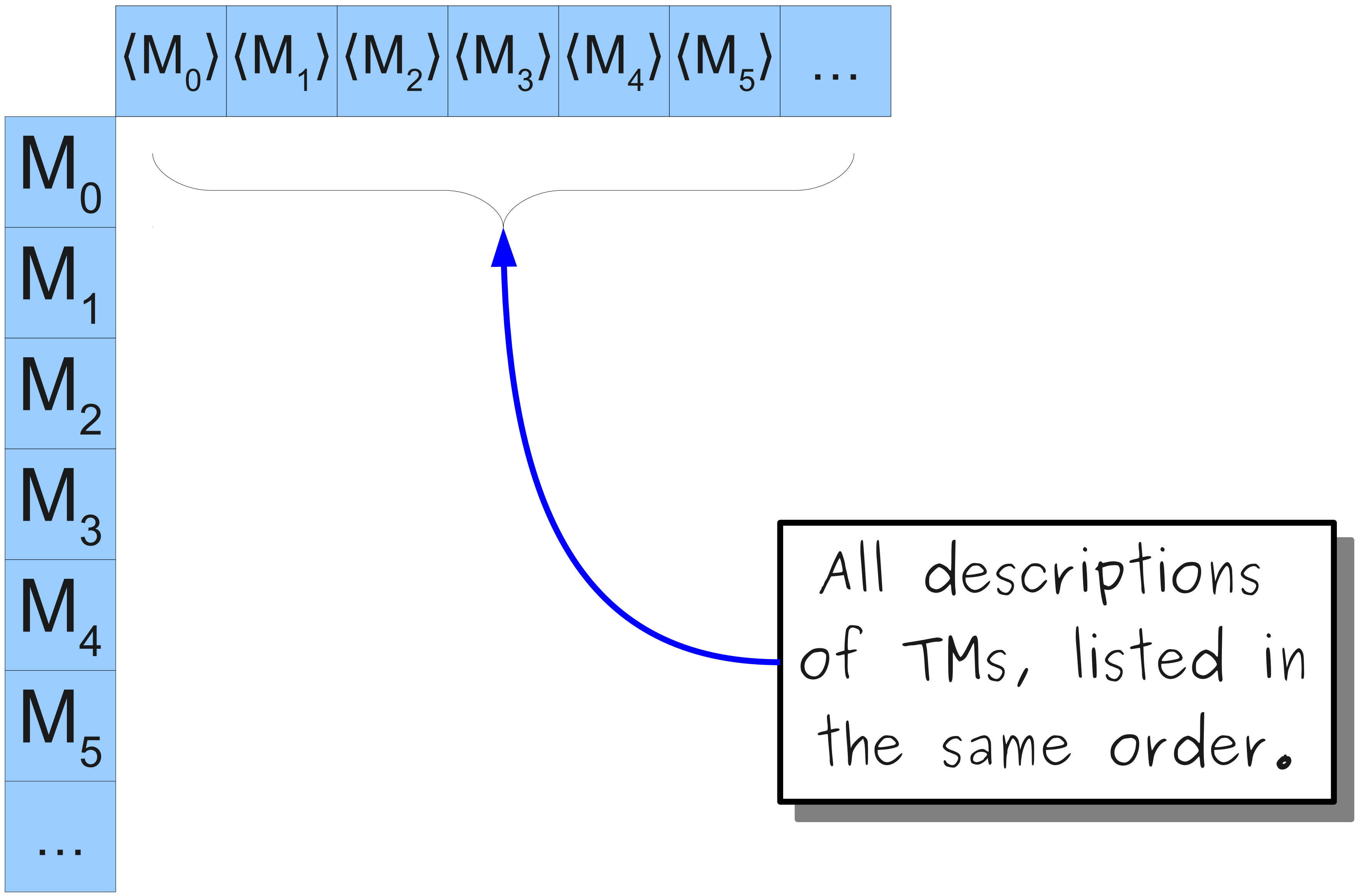
- Two questions:
  - What is this language?
  - Is this language **RE**?


M0
M1
M2
M3
M4
M5
...



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


⟨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							
M_2							
M_3							
M_4							
M_5							
…							

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

	$\langle M_0 \rangle$	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\langle M_5 \rangle$	…
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							
M_4							
M_5							
…							

	$\langle M_0 \rangle$	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\langle M_5 \rangle$	…
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							
M_5							
…							

	$\langle M_0 \rangle$	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\langle M_5 \rangle$	…
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							
…							

	$\langle M_0 \rangle$	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\langle M_5 \rangle$	…
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	…
…	…	…	…	…	…	…	…

	$\langle M_0 \rangle$	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\langle M_5 \rangle$	…
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	…

	$\langle M_0 \rangle$	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\langle M_5 \rangle$	…
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	…

"The language of all TMs that 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	…
…	…	…	…	…	…	…	…

Acc	Acc	Acc	No	Acc	No	…

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

	$\langle M_0 \rangle$	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\langle M_5 \rangle$	…
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	…

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

	$\langle M_0 \rangle$	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\langle M_5 \rangle$	…
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	…
…	…	…	…	…	…	…	…

Acc	Acc	Acc	No	Acc	No	…

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

This language is $\overline{L}_D$.

$\overline{L}_{\mathrm{D}} \in \mathbf{RE}$

- Here's an TM for $\overline{L}_{\mathrm{D}}$:

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

  Run M on $\langle M \rangle$.

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

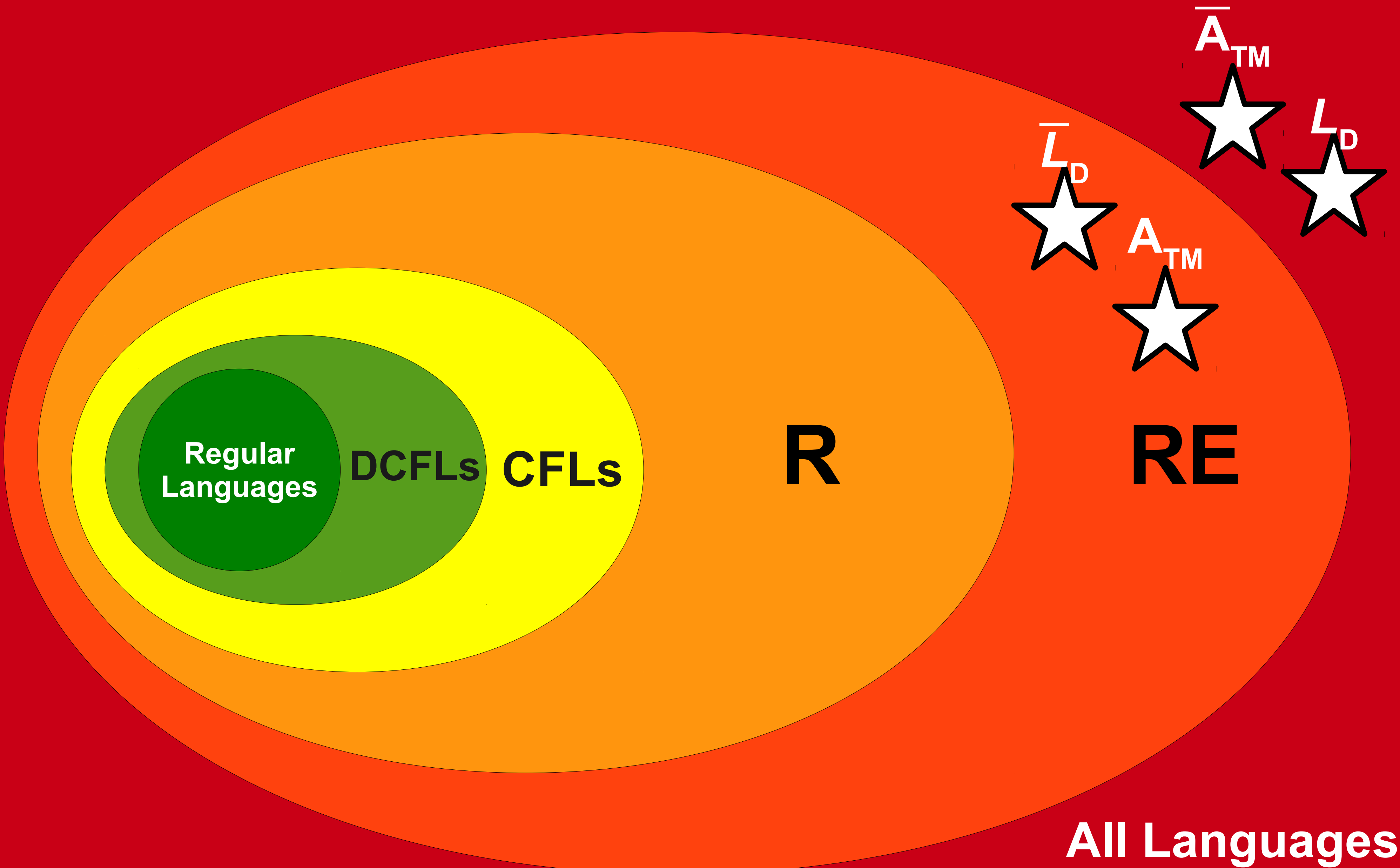
  If M rejects $\langle M \rangle$, reject."

- Then R accepts $\langle M \rangle$ iff $\langle M \rangle \in \mathscr{L}(M)$ iff $\langle \mathrm{M} \rangle \in \overline{L}_{\mathrm{D}}$, so $\mathscr{L}(R) = \overline{L}_{\mathrm{D}}$.

Is $\overline{L}_{\mathrm{D}}$ Decidable?

- We know that $\overline{L}_{\mathrm{D}} \in \mathbf{RE}$. Is $\overline{L}_{\mathrm{D}} \in \mathbf{R}$?
- **No** – by a similar argument from before.
  - If $\overline{L}_{\mathrm{D}} \in \mathbf{R}$, then $\overline{\overline{L}}_{\mathrm{D}} = L_{\mathrm{D}} \in \mathbf{R}$.
  - Since $\mathbf{R} \subset \mathbf{RE}$, this means that $L_{\mathrm{D}} \in \mathbf{RE}$.
  - This contradicts that $L_{\mathrm{D}} \notin \mathbf{RE}$.
  - So our assumption is wrong and $\overline{L}_{\mathrm{D}} \notin \mathbf{R}$.

The Limits of Computability



Finding Unsolvable Problems