Unsolvable Problems

Part Two

Outline for Today

- **Recap from Last Time**
  - Where are we right now?
- **Self-Referential Software**
  - Building self-referential software.
- **Resolving R $\overset{?}{=}$ RE**
  - Are decidability and recognizability the same?
- **Discovering Impossible Problems**
  - How do you show problems have no solutions?

Recap from Last Time

The Universal Turing Machine

- ***Theorem****:* There is a Turing machine $\mathbf{U_{TM}}$ called the ***universal Turing machine*** that, when run on $\langle M, w\rangle$, where M is a Turing machine and w is a string, simulates M running on w.
- Conceptually:

```
U_TM = "On input ⟨M, w⟩, where M is a TM and w ∈ Σ*:
            Set up the initial configuration of M running on w.
            while (true) {
                If M accepted w, then U_TM accepts ⟨M, w⟩.
                If M rejected w, then U_TM rejects ⟨M, w⟩.
                Otherwise, simulate one more step of M on w.
            }"
```

The Language of U_{TM}

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

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

- Since $A_{TM} = \mathscr{L}(U_{TM})$, we know that A_{TM} is an **RE** language.

Diagonalization Revisited

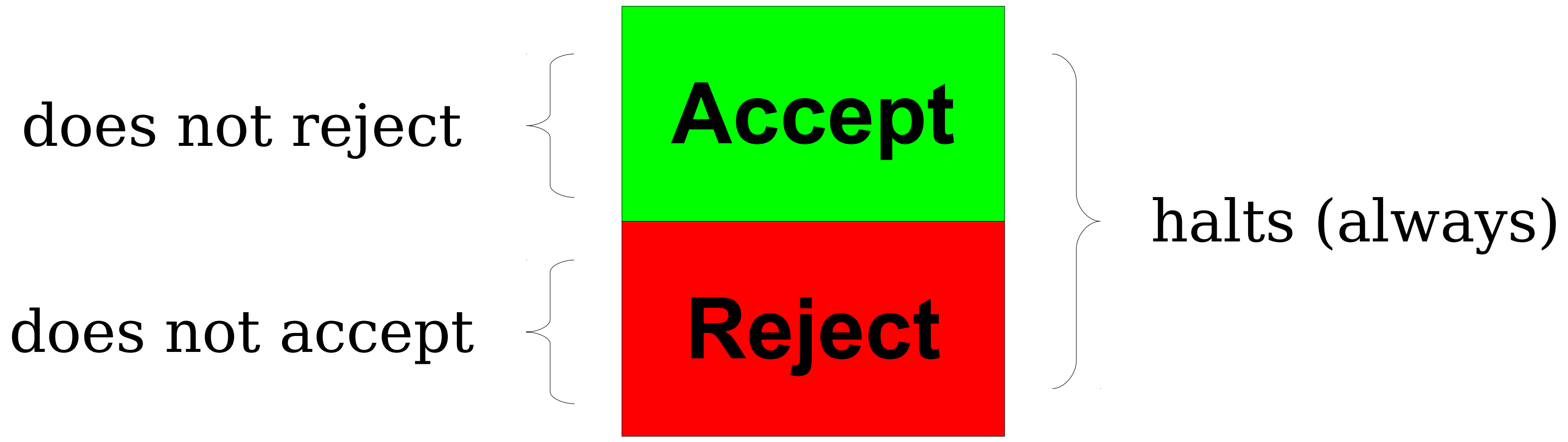
- The ***diagonalization language***, which we denote $\boldsymbol{L}_{\mathbf{D}}$, is defined as

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

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

Deciders

- Some Turing machines always halt; they never go into an infinite loop.
- If M is a TM and M halts on every possible input, then we say that M is a ***decider***.
- 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*** if there is a decider M such that $\mathscr{L}(M) = L$.
- The class $\mathbf{R}$ is the set of all decidable languages.

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

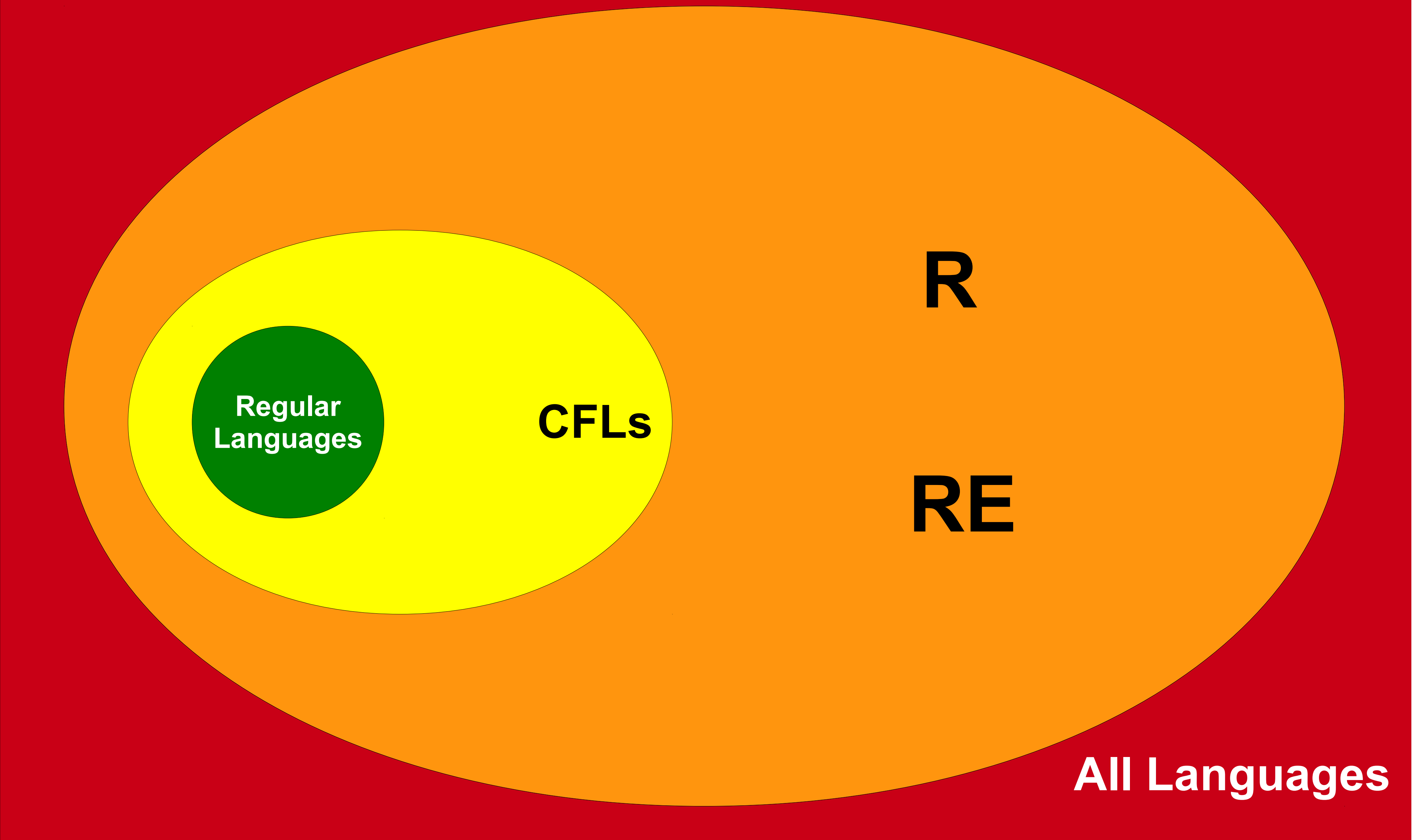
$\mathbf{R} \overset{?}{=} \mathbf{RE}$

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

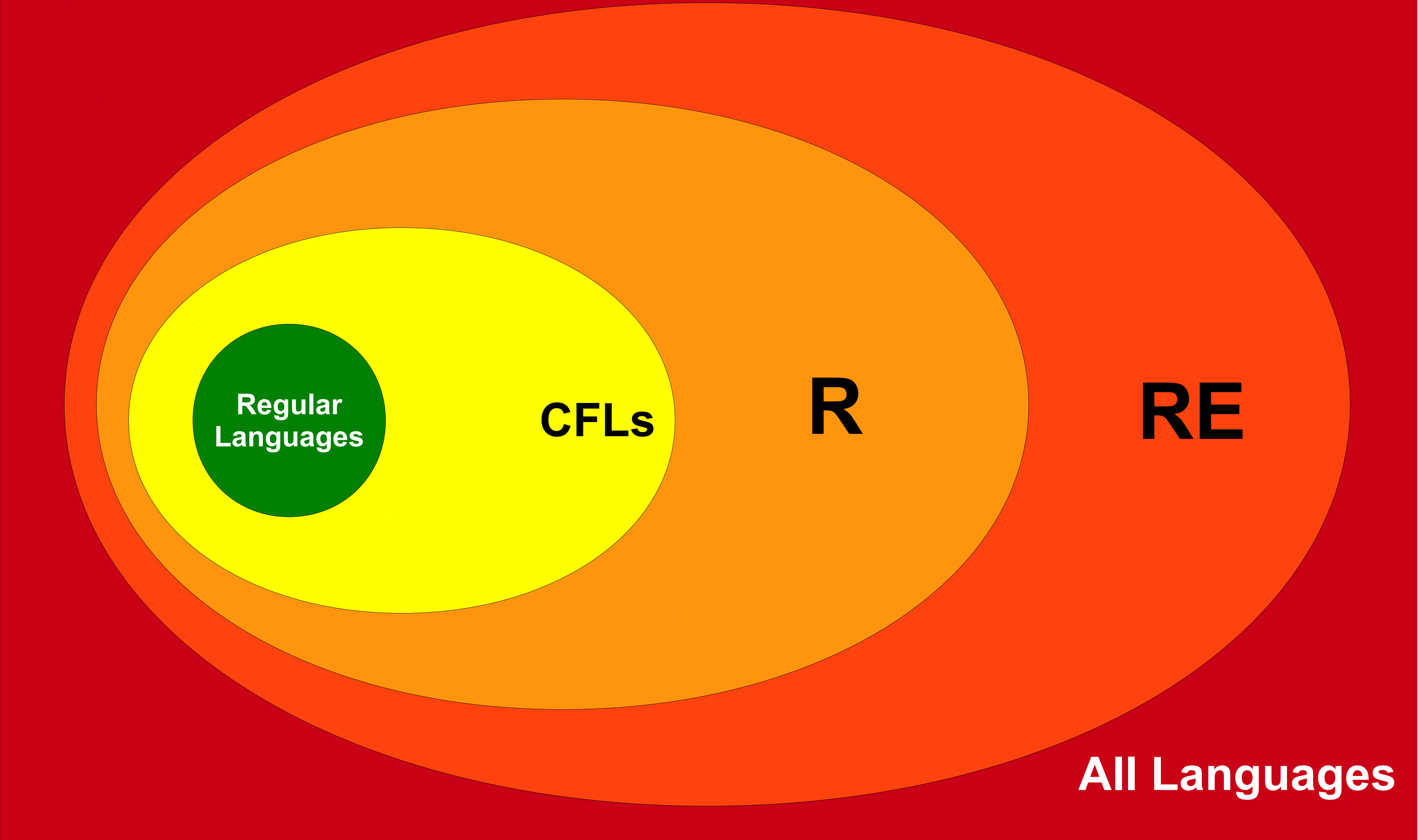
Is $\mathbf{R} = \mathbf{RE}$?

- 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?



Fundamental Question for Today:

What makes impossible problems impossible?

Self-Referential Software

Quines

Quines

- A ***Quine*** is a program that, when run, prints its own source code.
- Quines aren't allowed to just read the file containing their source code and print it out; that's cheating.
- How would you write such a program?

Writing a Quine

Self-Referential Programs

- ***Claim:*** It's possible to build computer programs that can ask questions about their own source code.
- General technique:
  - Write the code you'd like for processing the source code and doing something with the result.
  - Using the Quine construction technique from before, transform the program into a new program that gets its own source code, then does the processing you want.

The Recursion Theorem

- There is a deep result in computability theory called ***Kleene's second recursion theorem*** that, informally, states the following:

It is possible to construct TMs that perform arbitrary computations on their own descriptions.

- Intuitively, this generalizes our Quine constructions to work with arbitrary TMs.
- Want the formal statement of the theorem? Take CS154!

Resolving $\mathbf{R} \overset{?}{=} \mathbf{RE}$

R and **RE** Languages

- A language is in **R** if and only if there is an algorithm for deciding whether a string is in the language.
- A language is in **RE** if and only if there is a recognizer for the language (that is, a TM that can confirm yes answers, but might not actually solve the problem.)
- Does **R** = **RE**?

Returning to A_{TM}

- We know that the language A_{TM} is an **RE** language.
- If **R** = **RE**, then A_{TM} would have to belong to **R** as well, so it would be decidable.
- ***Claim:*** This will lead to Very Bad Things.

A Recipe for Disaster

- Consider the following facts about TMs:
  - TMs can run other TMs as subroutines.
  - TMs can make decisions about what to do next based on the results of these subroutines.
- What happens when we throw these two facts into the mix?
  - TMs can accept other TMs as input.
  - TMs can get their own descriptions.

A Recipe for Disaster

- Let's suppose that A_{TM} is decidable.
- This means that TMs can *decide* whether an arbitrary TM will accept an arbitrary string.
- What does the following TM do?

```
M = "On input w:
        · Have M get its own description, ⟨M⟩.
        · Decide whether M will accept w.
        · If M will accept w, choose to reject w.
        · If M will not accept w, choose to accept w."
```

Knowing the Future

- This TM is analogous to a classical philosophical/logical paradox:

  If you know what you are fated to do, can you avoid your fate?

- If A_{TM} is decidable, we can construct a TM that determines what it's going to do in the future (whether it will accept its input), then actively chooses to do the opposite.
- This leads to an impossible situation with only one resolution: **A_{TM} must not be decidable!**

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

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

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

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

Proof: By contradiction; assume that $A_{TM} \in \mathbf{R}$. This means that TMs can decide whether any TM/string pair belongs to A_{TM}.

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

Proof: By contradiction; assume that $A_{TM} \in \mathbf{R}$. This means that TMs can decide whether any TM/string pair belongs to A_{TM}. Equivalently, this means that Turing machines can decide whether an arbitrary TM will accept an arbitrary string.

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

Proof: By contradiction; assume that $A_{TM} \in \mathbf{R}$. This means that TMs can decide whether any TM/string pair belongs to A_{TM}. Equivalently, this means that Turing machines can decide whether an arbitrary TM will accept an arbitrary string. Given this, we could then construct the following TM:

```
M = "On input w:
        Have M obtain its own description, ⟨M⟩.
        Decide whether M will accept w.
        If M will accept w, then M chooses to reject w.
        If M will not accept w, then M chooses to accept w."
```

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

Proof: By contradiction; assume that $A_{TM} \in \mathbf{R}$. This means that TMs can decide whether any TM/string pair belongs to A_{TM}. Equivalently, this means that Turing machines can decide whether an arbitrary TM will accept an arbitrary string. Given this, we could then construct the following TM:

```
M = "On input w:
        Have M obtain its own description, ⟨M⟩.
        Decide whether M will accept w.
        If M will accept w, then M chooses to reject w.
        If M will not accept w, then M chooses to accept w."
```

Consider what happens if we run M on a string w.

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

Proof: By contradiction; assume that $A_{TM} \in \mathbf{R}$. This means that TMs can decide whether any TM/string pair belongs to A_{TM}. Equivalently, this means that Turing machines can decide whether an arbitrary TM will accept an arbitrary string. Given this, we could then construct the following TM:

M = "On input *w*:
  Have *M* obtain its own description, ⟨*M*⟩.
  Decide whether *M* will accept *w*.
  If *M* will accept *w*, then *M* chooses to reject *w*.
  If *M* will not accept *w*, then *M* chooses to accept *w*."

Consider what happens if we run *M* on a string *w*. If *M* accepts *w*, then by its own construction *M* was supposed to not accept *w*.

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

Proof: By contradiction; assume that $A_{TM} \in \mathbf{R}$. This means that TMs can decide whether any TM/string pair belongs to A_{TM}. Equivalently, this means that Turing machines can decide whether an arbitrary TM will accept an arbitrary string. Given this, we could then construct the following TM:

```
M = “On input w:
        Have M obtain its own description, ⟨M⟩.
        Decide whether M will accept w.
        If M will accept w, then M chooses to reject w.
        If M will not accept w, then M chooses to accept w.”
```

Consider what happens if we run M on a string w. If M accepts w, then by its own construction M was supposed to not accept w. Otherwise, if M does not accept w, then by construction we know that M rejects, and that happens when M was supposed to accept w.

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

Proof: By contradiction; assume that $A_{TM} \in \mathbf{R}$. This means that TMs can decide whether any TM/string pair belongs to A_{TM}. Equivalently, this means that Turing machines can decide whether an arbitrary TM will accept an arbitrary string. Given this, we could then construct the following TM:

```
M = "On input w:
        Have M obtain its own description, ⟨M⟩.
        Decide whether M will accept w.
        If M will accept w, then M chooses to reject w.
        If M will not accept w, then M chooses to accept w."
```

Consider what happens if we run M on a string w. If M accepts w, then by its own construction M was supposed to not accept w. Otherwise, if M does not accept w, then by construction we know that M rejects, and that happens when M was supposed to accept w.

In both cases we reach a contradiction, so our assumption must have been wrong.

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

Proof: By contradiction; assume that $A_{TM} \in \mathbf{R}$. This means that TMs can decide whether any TM/string pair belongs to A_{TM}. Equivalently, this means that Turing machines can decide whether an arbitrary TM will accept an arbitrary string. Given this, we could then construct the following TM:

M = "On input w:
    Have M obtain its own description, $\langle M \rangle$.
    Decide whether M will accept w.
    If M will accept w, then M chooses to reject w.
    If M will not accept w, then M chooses to accept w."

Consider what happens if we run M on a string w. If M accepts w, then by its own construction M was supposed to not accept w. Otherwise, if M does not accept w, then by construction we know that M rejects, and that happens when M was supposed to accept w.

In both cases we reach a contradiction, so our assumption must have been wrong. Therefore, $A_{TM} \notin \mathbf{R}$.

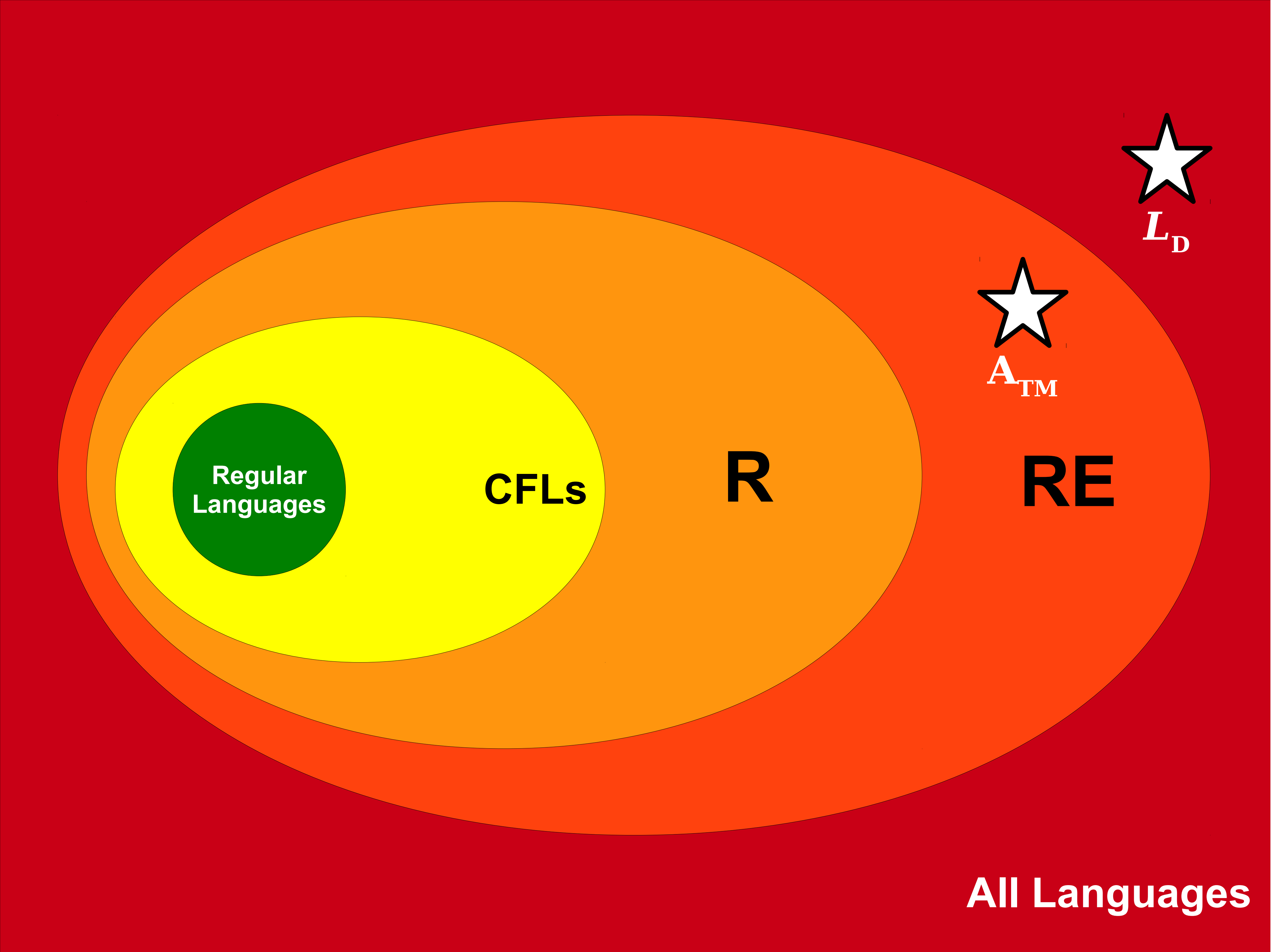
Theorem: $A_{TM} \notin \mathbf{R}$.

Proof: By contradiction; assume that $A_{TM} \in \mathbf{R}$. This means that TMs can decide whether any TM/string pair belongs to A_{TM}. Equivalently, this means that Turing machines can decide whether an arbitrary TM will accept an arbitrary string. Given this, we could then construct the following TM:

M = "On input w:
 Have M obtain its own description, $\langle M \rangle$.
 Decide whether M will accept w.
 If M will accept w, then M chooses to reject w.
 If M will not accept w, then M chooses to accept w."

Consider what happens if we run M on a string w. If M accepts w, then by its own construction M was supposed to not accept w. Otherwise, if M does not accept w, then by construction we know that M rejects, and that happens when M was supposed to accept w.

In both cases we reach a contradiction, so our assumption must have been wrong. Therefore, $A_{TM} \notin \mathbf{R}$. ■


L_D
A_{TM}
Regular Languages
CFLs
R
RE
All Languages

Time-Out for Announcements!

Midterm Logistics

- Second midterm is this Thursday, November 13, from 7PM – 10PM.
- Rooms divvied up by last name:
  - **`Abr – Sad`**: Go to **Cemex Auditorium**.
  - **`Sal – Zie`**: Go to **Cubberly Auditorium**.
- Exam is closed-computer, closed-book, open one page of notes (double-sided, 8.5” × 11”).
- Cumulative exam, focus is PS4 – PS6.

Solution Sets Available

- Problem Set Six solutions will be available outside at the end of class.
- Extra Practice Problems 3 solutions will be available outside as well.
- Missed them? Check the filing cabinet.
- SCPD – we'll directly email these solutions to you so that you receive them before the exam.

Your Questions!

“Is there a reason that the midterms are out of 24? I know that grades would probably end up being pretty much the same even if there were more points because of the way the class is weighted, but it seems to put a lot of pressure on getting every point.”

“Keith, did you section-lead when you were an undergraduate at Stanford?”

“I hated math classes when I was a child, and that fact almost stopped me from reaching my first stab at proof-based math in CS103, which I love. What math subjects should kids take, and, more importantly, how can we help kids fall in love with math?”

More Undecidable Problems

The Halting Problem

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

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

- As a formal language:

  ***HALT* = { ⟨*M*, *w*⟩ | *M* is a TM that halts on *w*. }**

- Question: is *HALT* decidable?

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

- Let's suppose that *HALT* is decidable.
- This means that TMs can *decide* whether an arbitrary TM will halt on an arbitrary string.
- What does the following TM do?

```
M = “On input w:
    · Have M get its own description, ⟨M⟩.
    · Decide whether M will halt on w.
    · If M will halt on w, choose to loop infinitely.
    · If M will loop on w, choose to accept w.”
```

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

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

Proof: By contradiction; assume that $HALT \in \mathbf{R}$.

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

Proof: By contradiction; assume that $HALT \in \mathbf{R}$. This means that TMs can decide whether any TM/string pair belongs to $HALT$.

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

Proof: By contradiction; assume that *HALT* $\in$ **R**. This means that TMs can decide whether any TM/string pair belongs to *HALT*. Equivalently, this means that Turing machines can decide whether an arbitrary TM will halt when run on an arbitrary string.

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

Proof: By contradiction; assume that *HALT* $\in$ **R**. This means that TMs can decide whether any TM/string pair belongs to *HALT*. Equivalently, this means that Turing machines can decide whether an arbitrary TM will halt when run on an arbitrary string. Given this, we could then construct the following TM:

```
M = "On input w:
        Have M obtain its own description, ⟨M⟩.
        Decide whether M will halt on w.
        If M will halt on w, then M chooses to loop.
        If M will loop on w, then M chooses to accept w."
```

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

Proof: By contradiction; assume that $HALT \in \mathbf{R}$. This means that TMs can decide whether any TM/string pair belongs to *HALT*. Equivalently, this means that Turing machines can decide whether an arbitrary TM will halt when run on an arbitrary string. Given this, we could then construct the following TM:

```
M = “On input w:
        Have M obtain its own description, ⟨M⟩.
        Decide whether M will halt on w.
        If M will halt on w, then M chooses to loop.
        If M will loop on w, then M chooses to accept w.”
```

This machine either halts on w or it loops on w.

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

Proof: By contradiction; assume that $HALT \in \mathbf{R}$. This means that TMs can decide whether any TM/string pair belongs to *HALT*. Equivalently, this means that Turing machines can decide whether an arbitrary TM will halt when run on an arbitrary string. Given this, we could then construct the following TM:

```
M = “On input w:
        Have M obtain its own description, ⟨M⟩.
        Decide whether M will halt on w.
        If M will halt on w, then M chooses to loop.
        If M will loop on w, then M chooses to accept w.”
```

This machine either halts on w or it loops on w. If it halts on w, then by its own construction it was supposed to loop on w.

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

Proof: By contradiction; assume that $HALT \in \mathbf{R}$. This means that TMs can decide whether any TM/string pair belongs to *HALT*. Equivalently, this means that Turing machines can decide whether an arbitrary TM will halt when run on an arbitrary string. Given this, we could then construct the following TM:

M = "On input w:
Have M obtain its own description, $\langle M \rangle$.
Decide whether M will halt on w.
If M will halt on w, then M chooses to loop.
If M will loop on w, then M chooses to accept w."

This machine either halts on w or it loops on w. If it halts on w, then by its own construction it was supposed to loop on w. If it loops on w, then by its own construction if was supposed to halt on w.

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

Proof: By contradiction; assume that *HALT* $\in$ **R**. This means that TMs can decide whether any TM/string pair belongs to *HALT*. Equivalently, this means that Turing machines can decide whether an arbitrary TM will halt when run on an arbitrary string. Given this, we could then construct the following TM:

```
$M$ = "On input $w$:
        Have $M$ obtain its own description, $\langle M \rangle$.
        Decide whether $M$ will halt on $w$.
        If $M$ will halt on $w$, then $M$ chooses to loop.
        If $M$ will loop on $w$, then $M$ chooses to accept $w$."
```

This machine either halts on *w* or it loops on *w*. If it halts on *w*, then by its own construction it was supposed to loop on *w*. If it loops on *w*, then by its own construction if was supposed to halt on *w*.

In both cases we reach a contradiction, so our assumption must have been wrong.

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

Proof: By contradiction; assume that $HALT \in \mathbf{R}$. This means that TMs can decide whether any TM/string pair belongs to *HALT*. Equivalently, this means that Turing machines can decide whether an arbitrary TM will halt when run on an arbitrary string. Given this, we could then construct the following TM:

```
M = "On input w:
        Have M obtain its own description, ⟨M⟩.
        Decide whether M will halt on w.
        If M will halt on w, then M chooses to loop.
        If M will loop on w, then M chooses to accept w."
```

This machine either halts on w or it loops on w. If it halts on w, then by its own construction it was supposed to loop on w. If it loops on w, then by its own construction if was supposed to halt on w.

In both cases we reach a contradiction, so our assumption must have been wrong. Therefore, $HALT \notin \mathbf{R}$.

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

Proof: By contradiction; assume that $HALT \in \mathbf{R}$. This means that TMs can decide whether any TM/string pair belongs to *HALT*. Equivalently, this means that Turing machines can decide whether an arbitrary TM will halt when run on an arbitrary string. Given this, we could then construct the following TM:

```
M = “On input w:
        Have M obtain its own description, ⟨M⟩.
        Decide whether M will halt on w.
        If M will halt on w, then M chooses to loop.
        If M will loop on w, then M chooses to accept w.”
```

This machine either halts on w or it loops on w. If it halts on w, then by its own construction it was supposed to loop on w. If it loops on w, then by its own construction if was supposed to halt on w.

In both cases we reach a contradiction, so our assumption must have been wrong. Therefore, $HALT \notin \mathbf{R}$. ■

The General Pattern

- There seems to be general pattern here for proving undecidability:
  - Assume that the language in question, usually a language about TMs, is decidable.
  - Build a machine that *decides* whether it has the property, then chooses to do something contrary to that property.
  - Conclude that something is terribly wrong, meaning that original language wasn't decidable.
- The second step is usually the trickiest.

A Trickier Example

- All regular languages are also **R** and **RE** languages, so some TMs have regular languages.
- Not all **R** or **RE** languages are regular, so many TMs have nonregular languages.
- Consider the following language:

  $L = \{ \langle M \rangle \mid M \text{ is a TM and } \mathscr{L}(M) \text{ is regular } \}$
- Is this language decidable?

Testing Regularity

- On the one hand, it might seem like this is easy to check – just see if the TM is equivalent to a DFA!
- On the other hand, that's actually really, really hard. TMs can do all sorts of crazy things before accepting or rejecting, so how would you be able to tell whether there's a DFA that does the same thing as a TM?

Testing Regularity

- Let's see if we can prove that this language is undecidable.
- Suppose that a TM can *decide* whether a given TM has a regular language.
- Could we build a TM that does the following?
    - *Decide* whether its own language is regular.
    - If so, choose to have a nonregular language.
    - If not, choose to have a regular language.

Testing Regularity

- Could we build a TM that does the following?
  - *Decide* whether its own language is regular.
  - If so, choose to have a nonregular language.
  - If not, choose to have a regular language.
- The challenge here is that the language of the TM depends on the TMs behavior across every possible string it can be run on, not just a single input string.

Testing Regularity

- Assume for the sake of contradiction that a TM can *decide* whether a TM has a regular language.
- Consider this TM:

```
M = “On input w:
        · Have M get its own description, ⟨M⟩.
        · Decide whether ℒ(M) is regular.
        · If ℒ(M) is regular:
            · If w has the form aⁿbⁿ, M accepts.
            · If not, M rejects.
        · If ℒ(M) is not regular, then M rejects.”
```

Testing Regularity

- Assume for the sake o
  a TM can *decide* whet
  regular language.
- Consider this TM:

If M decides that it has a regular language, what will the overall language of the machine be?

```
M = "On input w:
        · Have M get its own description, ⟨M⟩.
        · Decide whether ℒ(M) is regular.
        · If ℒ(M) is regular:
            · If w has the form aⁿbⁿ, M accepts.
            · If not, M rejects.
        · If ℒ(M) is not regular, then M rejects."
```

Testing Regularity

- Assume for the sake o
  a TM can *decide* whet
  regular language.
- Consider this TM:

If M decides that it has a regular language, what will the overall language of the machine be?

```
M = "On input w:
        · Have M get its own description, ⟨M⟩.
        · Decide whether $\mathscr{L}(M)$ is regular.
        · If $\mathscr{L}(M)$ is regular:
            · If w has the form $a^n b^n$, M accepts.
            · If not, M rejects.
        · If $\mathscr{L}(M)$ is not regular, then M rejects."
```

Testing Regularity

- Assume for the sake o
  a TM can *decide* whet
  regular language.
- Consider this TM:

If M decides that it has a regular language, what will the overall language of the machine be?

$\{ \mathtt{a}^n\mathtt{b}^n \mid n \in \mathbb{N} \}$

```
M = "On input w:
        · Have M get its own description, ⟨M⟩.
        · Decide whether ℒ(M) is regular.
        · If ℒ(M) is regular:
            · If w has the form aⁿbⁿ, M accepts.
            · If not, M rejects.
        · If ℒ(M) is not regular, then M rejects."
```

Testing Regularity

- Assume for the sake o
  a TM can *decide* whet
  regular language.
- Consider this TM:

If M decides that it has a regular language, what will the overall language of the machine be?

$\{ \mathbf{a}^n\mathbf{b}^n \mid n \in \mathbb{N} \}$

```
$M$ = "On input $w$:
        · Have $M$ get its own description, $\langle M \rangle$.
        · Decide whether $\mathscr{L}(M)$ is regular.
        · If $\mathscr{L}(M)$ is regular:
            · If $w$ has the form $\mathbf{a}^n\mathbf{b}^n$, $M$ accepts.
            · If not, $M$ rejects.
        · If $\mathscr{L}(M)$ is not regular, then $M$ rejects."
```

Testing Regularity

- Assume for the sake o a TM can *decide* whet regular language.
- Consider this TM:

If M decides that it has a nonregular language, what will the overall language of the machine be?

```
M = "On input w:
        · Have M get its own description, ⟨M⟩.
        · Decide whether ℒ(M) is regular.
        · If ℒ(M) is regular:
            · If w has the form aⁿbⁿ, M accepts.
            · If not, M rejects.
        · If ℒ(M) is not regular, then M rejects."
```

Testing Regularity

- Assume for the sake o
  a TM can *decide* whet
  regular language.
- Consider this TM:

If M decides that it has a nonregular language, what will the overall language of the machine be?

```
M = "On input w:
          · Have M get its own description, ⟨M⟩.
          · Decide whether ℒ(M) is regular.
          · If ℒ(M) is regular:
              · If w has the form aⁿbⁿ, M accepts.
              · If not, M rejects.
          · If ℒ(M) is not regular, then M rejects."
```

Testing Regularity

- Assume for the sake o a TM can *decide* whet regular language.
- Consider this TM:

If M decides that it has a nonregular language, what will the overall language of the machine be?

∅

```
M = "On input w:
        · Have M get its own description, ⟨M⟩.
        · Decide whether 𝓛(M) is regular.
        · If 𝓛(M) is regular:
            · If w has the form aⁿbⁿ, M accepts.
            · If not, M rejects.
        · If 𝓛(M) is not regular, then M rejects."
```

Testing Regularity

- Assume for the sake o a TM can *decide* whet regular language.
- Consider this TM:

If M decides that it has a nonregular language, what will the overall language of the machine be?

$\emptyset$

```
M = "On input w:
        · Have M get its own description, ⟨M⟩.
        · Decide whether $\mathscr{L}(M)$ is regular.
        · If $\mathscr{L}(M)$ is regular:
            · If w has the form $a^n b^n$, M accepts.
            · If not, M rejects.
        · If $\mathscr{L}(M)$ is not regular, then M rejects."
```

Theorem: Let $L = \{ \langle M \rangle \mid M \text{ is a TM and } \mathscr{L}(M) \text{ is regular } \}$. Then $L \notin \mathbf{R}$.

Proof: By contradiction; assume that $L \in \mathbf{R}$. This means that TMs can decide whether arbitrary Turing machines have regular languages. Given this, we could then construct the following TM:

```
M = "On input w:
        Have M obtain its own description, ⟨M⟩.
        Decide whether 𝓛(M) is regular.
        If 𝓛(M) is regular:
            If w has the form aⁿbⁿ, then M accepts w.
            Otherwise, M rejects w.
        If 𝓛(M) is not regular, then M rejects w"
```

Notice that by construction, $\mathscr{L}(M)$ will be $\{\mathtt{a}^n\mathtt{b}^n \mid n \in \mathbb{N} \}$ (if the first branch is true) or $\emptyset$ (if the second branch is true). Now, if $\mathscr{L}(M) = \{\mathtt{a}^n\mathtt{b}^n \mid n \in \mathbb{N}\}$ (a nonregular language), then M was supposed to have a regular language. Similarly, if $\mathscr{L}(M) = \emptyset$ (a regular language), then M was supposed to have a nonregular language.

In both cases we reach a contradiction, so our assumption must have been wrong. Therefore, $L \notin \mathbf{R}$. ■