Unsolvable Problems

Part Three

Outline for Today

- **Recap from Last Time**
  - Where are we right now?
- **Non-RE Languages**
  - Self-reference and **RE**.
- **The Class co-RE**
  - A new perspective on problem-solving.
- **Monster Problems**
  - Terrifyingly impossible problems.

Recap from Last Time

The Recursion Theorem

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

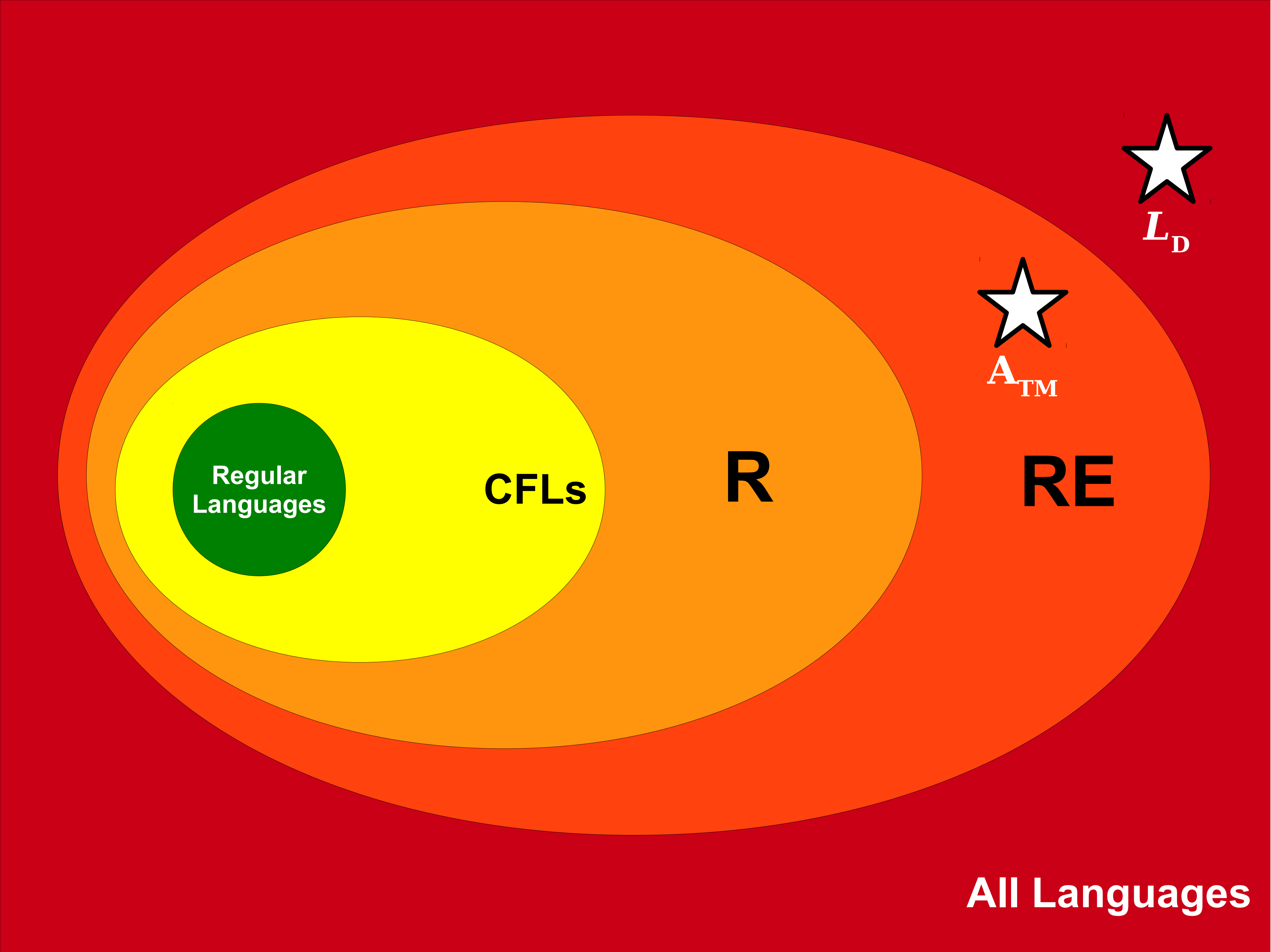
It is possible to construct TMs that perform arbitrary computations on their own descriptions.

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

$A_{TM} \notin \mathbf{R}$

- Let's suppose that A_{TM} is decidable.
- This means that TMs can *decide* whether an arbitrary TM will accept an arbitrary string.
- The following TM is paradoxical:

```
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."
```


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

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:

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

- Question: is *HALT* decidable?

Is *HALT* ∈ **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, $\langle M \rangle$.
            · 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 $w$."
```

Testing Regularity

- Let $L = \{ \langle M \rangle \mid M$ is a TM and $\mathscr{L}(M)$ is regular $\}$
- 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."
```

New Stuff!

Finding Non-**RE** Languages

Finding Non-**RE** Languages

- The approach we've used so far for finding undecidable languages has used the fact that if a language is decidable, then TMs can *decide* whether strings belong to those languages.
- What happens if we want to find *unrecognizable* languages, languages that aren't in **RE**?

$\mathrm{A}_{\mathrm{TM}} \in \mathbf{RE}$

- Earlier, we saw that $\mathrm{A}_{\mathrm{TM}} \in \mathbf{RE}$.
- We proved that $\mathrm{A}_{\mathrm{TM}} \notin \mathbf{R}$ by constructing this TM, which caused a contradiction:

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

A_{TM} ∈ **RE**

- Earlier, we saw that A_{TM} ∈ **RE**.
- Since A_{TM} ∈ **RE**, we know that TMs can *recognize* when a TM accepts a string *w*.
- Why doesn't this machine cause a contradiction as well?

```
M = “On input w:
        · Have M get its own description, ⟨M⟩.
        · Try to recognize if M accepts w.
        · If M will accept w, choose to reject w.
        · If M will not accept w, choose to accept w.”
```

A_{TM} ∈ **RE**

Earlier, we saw that A_{TM} ∈ **RE**.

Since A_{TM} ∈ **RE**, we know that TMs can *recognize* when a TM accepts a string *w*.

Why doesn't this machine cause a contradiction as well?

M = "On input *w*:
- Have *M* get its own description, ⟨*M*⟩.
- Try to ***recognize*** if *M* accepts *w*.
- If *M* will accept *w*, choose to reject *w*.
- If *M* will not accept *w*, choose to accept *w*."

A_{TM} ∈ **RE**

Earlier, we saw that A_{TM} ∈ **RE**.

Since A_{TM} ∈ **RE**, we know that TI
when a TM accepts a string *w*.

Why doesn't this machine cause
well?

Remember:
recognizers can loop infinitely!
This step might never terminate.

M = "On input *w*:
- Have *M* get its own description, ⟨*M*⟩.
- Try to ***recognize*** if *M* accepts *w*.
- If *M* will accept *w*, choose to reject *w*.
- If *M* will not accept *w*, choose to accept *w*."

A_{TM} ∈ **RE**

- Earlier, we saw that A_{TM} ∈ **RE**.
- Since A_{TM} ∈ **RE**, we know that TMs can *recognize* when a TM accepts a string *w*.
- Why doesn't this machine cause a contradiction as well?

```
M = “On input w:
        · Have M get its own description, ⟨M⟩.
        · Try to recognize if M accepts w.
        · If M will accept w, choose to reject w.
        · If M will not accept w, choose to accept w.”
```

A_{TM} ∈ **RE**

- Earlier, we saw that A_{TM} ∈ **RE**.
- Since A_{TM} ∈ **RE**, we know that TMs can *recognize* when a TM accepts a string *w*.
- Why doesn't this machine cause a contradiction as well?

```
M = "On input w:
        · Have M get its own description, ⟨M⟩.
        · Try to recognize if M accepts w. (This step
            might loop forever if M does not accept w).
        · If M will accept w, choose to reject w.
        · If M will not accept w, choose to accept w."
```

$A_{TM} \in$ **RE**

- Earlier, we saw that $A_{TM} \in$ **RE**.
- Since $A_{TM} \in$ **RE**, we know that TMs can *recognize* when a TM accepts a string *w*.
- Why doesn't this machine cause a contradiction as well?

```
M = “On input w:
        · Have M get its own description, ⟨M⟩.
        · Try to recognize if M accepts w. (This step
            might loop forever if M does not accept w).
        · If M will accept w, choose to reject w.
        · If M will not accept w, choose to accept w.”
```

$A_{TM} \in$ **RE**

- Earlier, we saw that $A_{TM} \in$ **RE**.
- Since $A_{TM} \in$ **RE**, we know t
  when a TM accepts a strin
- Why doesn't this machine c
  well?

Remember – a recognizer only loops if the answer to the question is "no."

M = "On input *w*:
  - Have *M* get its own description, ⟨*M*⟩.
  - Try to ***recognize*** if *M* accepts *w*. *(This step might loop forever if M does not accept w).*
  - If *M* will accept *w*, choose to reject *w*.
  - If *M* will not accept *w*, choose to accept *w*."

A_{TM} ∈ **RE**

- Earlier, we saw that A_{TM} ∈ **RE**.
- Since A_{TM} ∈ **RE**, we know that TMs can *recognize* when a TM accepts a string *w*.
- Why doesn't this machine cause a contradiction as well?

```
M = "On input w:
        · Have M get its own description, ⟨M⟩.
        · Try to recognize if M accepts w. (This step
            might loop forever if M does not accept w).
        · If M will accept w, choose to reject w.
        · If M will not accept w, choose to accept w."
```

A_{TM} ∈ **RE**

- Earlier, we saw that A_{TM} ∈ **RE**.
- Since A_{TM} ∈ **RE**, we know
when a TM accepts a strin
- Why doesn't this machine
well?

If *M doesn't* accept *w*, but the recognition step loops infinitely, but that's consistent with what's supposed to happen.

```
M = “On input w:
      · Have M get its own description, ⟨M⟩.
      · Try to recognize if M accepts w. (This step
          might loop forever if M does not accept w).
      · If M will accept w, choose to reject w.
      · If M will not accept w, choose to accept w.”
```

Decision versus Recognition

- If a language is decidable, we can put instructions like this in our TMs:

  ***Decide* whether *X* is true.**

- If a language is merely recognizable, we can put this equivalent instruction in our TMs:

  Try to *recognize* whether *X* is true.

- This introduces a tricky element into the mix.

Decision versus Recognition

- If we have a TM do the following:

  Try to *recognize* whether *X* is true.

  then one of two things can happen.
- First, we might get back an answer. In that case, the answer is definitively correct.
- Second, this step might loop forever. In that case, we know that *X* is false, but the machine can't act on it (because it's stuck in an infinite loop.)
- When proving that a language is not **RE** using self-referential TMs, it's critically important to remember that the recognition step might loop forever.

Revisiting a Proof

- We already proved that L_D is not an **RE** language by using a proof by diagonalization.
- Can we use the recursion theorem and self-reference to prove this a different way?

Revisiting a Proof

- Let's suppose that $L_D \in$ **RE**, meaning that a TM can recognize when an arbitrary TM does not accept its own encoding.
- We need to build a machine such that
  - if it's supposed to accept its own encoding, instead it doesn't.
  - if it was supposed to not accept its own encoding, instead it does.

Revisiting a Proof

- Assume that $L_D \in$ **RE**.
- What does this machine do?

```
M = “On input w:
        · Have M get its own description, ⟨M⟩.
        · Try to recognize if M does not accept ⟨M⟩.
            (This step might loop forever if M does
            accept ⟨M⟩).
        · If M will accept ⟨M⟩, choose to reject w.
        · If M will not accept ⟨M⟩, choose to accept w.”
```

M = "On input w:

- Have M get its own description, $\langle M \rangle$.
- Try to ***recognize*** if M does not accept $\langle M \rangle$.
  (This step might loop forever if M does accept $\langle M \rangle$*).*
- If M will accept $\langle M \rangle$, choose to reject w.
- If M will not accept $\langle M \rangle$, choose to accept w."

M = "On input w:

- Have M get its own description, $\langle M \rangle$.
- Try to ***recognize*** if M does not accept $\langle M \rangle$.
  (This step might loop forever if M does accept $\langle M \rangle$).
- If M will accept $\langle M \rangle$, choose to reject w.
- If M will not accept $\langle M \rangle$, choose to accept w."

Focus on the recognition step. There are two possible outcomes:

```
$M$ = "On input $w$:
    · Have $M$ get its own description, $\langle M \rangle$.
    · Try to ***recognize*** if $M$ does not accept $\langle M \rangle$.
      *(This step might loop forever if M does
      accept ⟨M⟩).*
    · If $M$ will accept $\langle M \rangle$, choose to reject $w$.
    · If $M$ will not accept $\langle M \rangle$, choose to accept $w$."
```

Focus on the recognition step. There are two possible outcomes:

The recognition step finishes and we know whether M accepts $\langle M \rangle$:

M = "On input w:
- Have M get its own description, $\langle M \rangle$.
- Try to ***recognize*** if M does not accept $\langle M \rangle$.
  (This step might loop forever if M does accept ⟨M⟩).
- If M will accept $\langle M \rangle$, choose to reject w.
- If M will not accept $\langle M \rangle$, choose to accept w."

Focus on the recognition step. There are two possible outcomes:

The recognition step finishes and we know whether M accepts $\langle M \rangle$:

The recognition step loops forever:

M = "On input w:

- Have M get its own description, $\langle M \rangle$.
- Try to ***recognize*** if M does not accept $\langle M \rangle$.
  (This step might loop forever if M does accept $\langle M \rangle$).
- If M will accept $\langle M \rangle$, choose to reject w.
- If M will not accept $\langle M \rangle$, choose to accept w."

Focus on the recognition step. There are two possible outcomes:

The recognition step finishes and we know whether M accepts $\langle M \rangle$:
    If M will not accept $\langle M \rangle$, then M accepts everything

The recognition step loops forever:

M = "On input w:
- Have M get its own description, $\langle M \rangle$.
- Try to ***recognize*** if M does not accept $\langle M \rangle$.
  (This step might loop forever if M does accept $\langle M \rangle$).
- If M will accept $\langle M \rangle$, choose to reject w.
- If M will not accept $\langle M \rangle$, choose to accept w."

Focus on the recognition step. There are two possible outcomes:

The recognition step finishes and we know whether M accepts $\langle M \rangle$:
If M will not accept $\langle M \rangle$, then M accepts everything, including $\langle M \rangle$

The recognition step loops forever:

M = "On input w:
- Have M get its own description, $\langle M\rangle$.
- Try to ***recognize*** if M does not accept $\langle M\rangle$.
  (This step might loop forever if M does accept ⟨M⟩).
- If M will accept $\langle M\rangle$, choose to reject w.
- If M will not accept $\langle M\rangle$, choose to accept w."

Focus on the recognition step. There are two possible outcomes:

The recognition step finishes and we know whether M accepts $\langle M\rangle$:
If M will not accept $\langle M\rangle$, then M accepts everything,
including $\langle M\rangle$ – *a contradiction!*

The recognition step loops forever:

M = "On input w:
- Have M get its own description, $\langle M \rangle$.
- Try to ***recognize*** if M does not accept $\langle M \rangle$.
  (This step might loop forever if M does accept $\langle M \rangle$).
- If M will accept $\langle M \rangle$, choose to reject w.
- If M will not accept $\langle M \rangle$, choose to accept w."

Focus on the recognition step. There are two possible outcomes:

The recognition step finishes and we know whether M accepts $\langle M \rangle$:

If M will not accept $\langle M \rangle$, then M accepts everything, including $\langle M \rangle$ – *a contradiction!*

If M will accept $\langle M \rangle$, then M rejects everything

The recognition step loops forever:

M = "On input w:
- Have M get its own description, $\langle M \rangle$.
- Try to ***recognize*** if M does not accept $\langle M \rangle$.
  (This step might loop forever if M does accept $\langle M \rangle$).
- If M will accept $\langle M \rangle$, choose to reject w.
- If M will not accept $\langle M \rangle$, choose to accept w."

Focus on the recognition step. There are two possible outcomes:

The recognition step finishes and we know whether M accepts $\langle M \rangle$:
  If M will not accept $\langle M \rangle$, then M accepts everything,
    including $\langle M \rangle$ – *a contradiction!*
  If M will accept $\langle M \rangle$, then M rejects everything,
    including $\langle M \rangle$

The recognition step loops forever:

M = "On input w:

- Have M get its own description, $\langle M \rangle$.
- Try to ***recognize*** if M does not accept $\langle M \rangle$.
  (This step might loop forever if M does accept $\langle M \rangle$*).*
- If M will accept $\langle M \rangle$, choose to reject w.
- If M will not accept $\langle M \rangle$, choose to accept w."

Focus on the recognition step. There are two possible outcomes:

The recognition step finishes and we know whether M accepts $\langle M \rangle$:

If M will not accept $\langle M \rangle$, then M accepts everything,
including $\langle M \rangle$ – *a contradiction!*

If M will accept $\langle M \rangle$, then M rejects everything,
including $\langle M \rangle$ – *a contradiction!*

The recognition step loops forever:

M = "On input w:
- Have M get its own description, $\langle M \rangle$.
- Try to ***recognize*** if M does not accept $\langle M \rangle$.
  (This step might loop forever if M does accept ⟨M⟩).
- If M will accept $\langle M \rangle$, choose to reject w.
- If M will not accept $\langle M \rangle$, choose to accept w."

Focus on the recognition step. There are two possible outcomes:

The recognition step finishes and we know whether M accepts $\langle M \rangle$:
If M will not accept $\langle M \rangle$, then M accepts everything, including $\langle M \rangle$ – *a contradiction!*
If M will accept $\langle M \rangle$, then M rejects everything, including $\langle M \rangle$ – *a contradiction!*

The recognition step loops forever:
This means that M accepts $\langle M \rangle$.

M = “On input w:
- Have M get its own description, $\langle M \rangle$.
- Try to ***recognize*** if M does not accept $\langle M \rangle$.
  (This step might loop forever if M does accept $\langle M \rangle$*).*
- If M will accept $\langle M \rangle$, choose to reject w.
- If M will not accept $\langle M \rangle$, choose to accept w.”

Focus on the recognition step. There are two possible outcomes:

The recognition step finishes and we know whether M accepts $\langle M \rangle$:
If M will not accept $\langle M \rangle$, then M accepts everything,
including $\langle M \rangle$ – *a contradiction!*
If M will accept $\langle M \rangle$, then M rejects everything,
including $\langle M \rangle$ – *a contradiction!*

The recognition step loops forever:
This means that M accepts $\langle M \rangle$.
But the recognition step loops, so M loops on everything

M = "On input w:

- Have M get its own description, $\langle M \rangle$.
- Try to ***recognize*** if M does not accept $\langle M \rangle$.
  (This step might loop forever if M does accept $\langle M \rangle$*).*
- If M will accept $\langle M \rangle$, choose to reject w.
- If M will not accept $\langle M \rangle$, choose to accept w."

Focus on the recognition step. There are two possible outcomes:

The recognition step finishes and we know whether M accepts $\langle M \rangle$:
    If M will not accept $\langle M \rangle$, then M accepts everything,
        including $\langle M \rangle$ – *a contradiction!*
    If M will accept $\langle M \rangle$, then M rejects everything,
        including $\langle M \rangle$ – *a contradiction!*

The recognition step loops forever:
    This means that M accepts $\langle M \rangle$.
    But the recognition step loops, so M loops on everything,
        including $\langle M \rangle$

M = "On input w:

- Have M get its own description, $\langle M \rangle$.
- Try to ***recognize*** if M does not accept $\langle M \rangle$.
  (This step might loop forever if M does accept $\langle M \rangle$).
- If M will accept $\langle M \rangle$, choose to reject w.
- If M will not accept $\langle M \rangle$, choose to accept w."

Focus on the recognition step. There are two possible outcomes:

The recognition step finishes and we know whether M accepts $\langle M \rangle$:
  If M will not accept $\langle M \rangle$, then M accepts everything,
    including $\langle M \rangle$ – *a contradiction!*
  If M will accept $\langle M \rangle$, then M rejects everything,
    including $\langle M \rangle$ – *a contradiction!*

The recognition step loops forever:
  This means that M accepts $\langle M \rangle$.
  But the recognition step loops, so M loops on everything,
    including $\langle M \rangle$ – *a contradiction!*

Theorem: $L_D \notin$ **RE**.

Proof: By contradiction; assume that $L_D \in$ **RE**. This means that TMs can recognize when arbitrary TMs do not accept their own encodings. Given this, we could construct the following TM:

```
M = "On input w:
        Have M obtain its own description, ⟨M⟩.
        Try to recognize if M will not accept ⟨M⟩.
            (This step might loop forever if M accepts ⟨M⟩.)
        If M will accept ⟨M⟩, then M chooses to reject ⟨M⟩.
        If M will not accept ⟨M⟩, then M chooses to accept ⟨M⟩."
```

Consider what happens if we run M on $\langle M \rangle$. If M accepts $\langle M \rangle$, then M was supposed to not accept $\langle M \rangle$. If M rejects $\langle M \rangle$, then M was supposed to accept $\langle M \rangle$. Otherwise, M loops on $\langle M \rangle$. The only way M loops is if the recognition step loops infinitely, in which case we know that M was supposed to accept $\langle M \rangle$.

In all cases we reach a contradiction, so our assumption must have been wrong. Therefore, $L_D \notin$ **RE**. ■

Time-Out for Announcements!

I honestly don't have any announcements.
That was kind of a bluff.

Your Questions!

“How long does it take you to make the slides showing the transitions of the Turing machines?”

“What is htiek?”

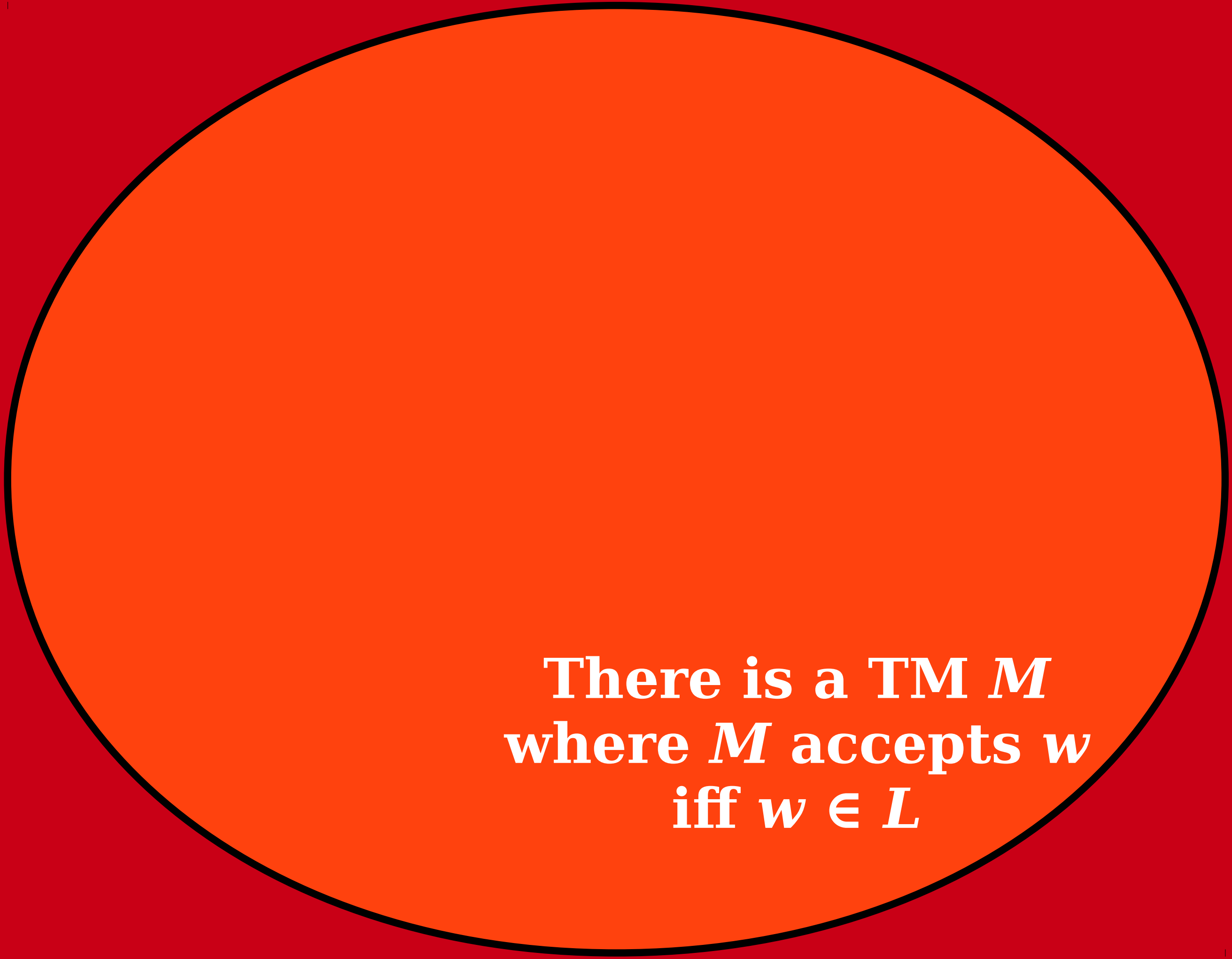
“Why do you like tournaments so much? It's shown up in two problem sets, a couple practice problems, and the second midterm.”

Back to CS103!

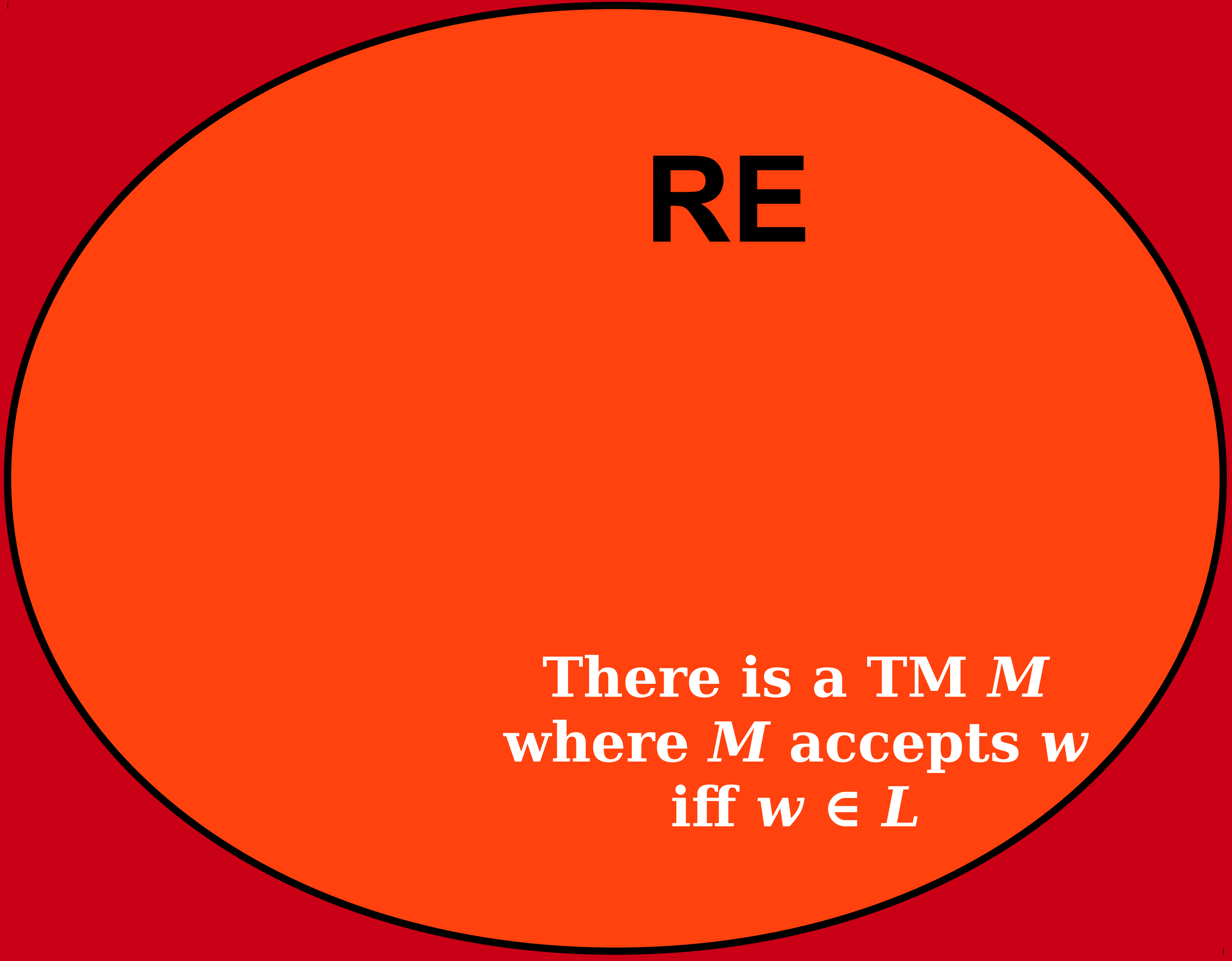
Resolving an Asymmetry

The Limits of Computability

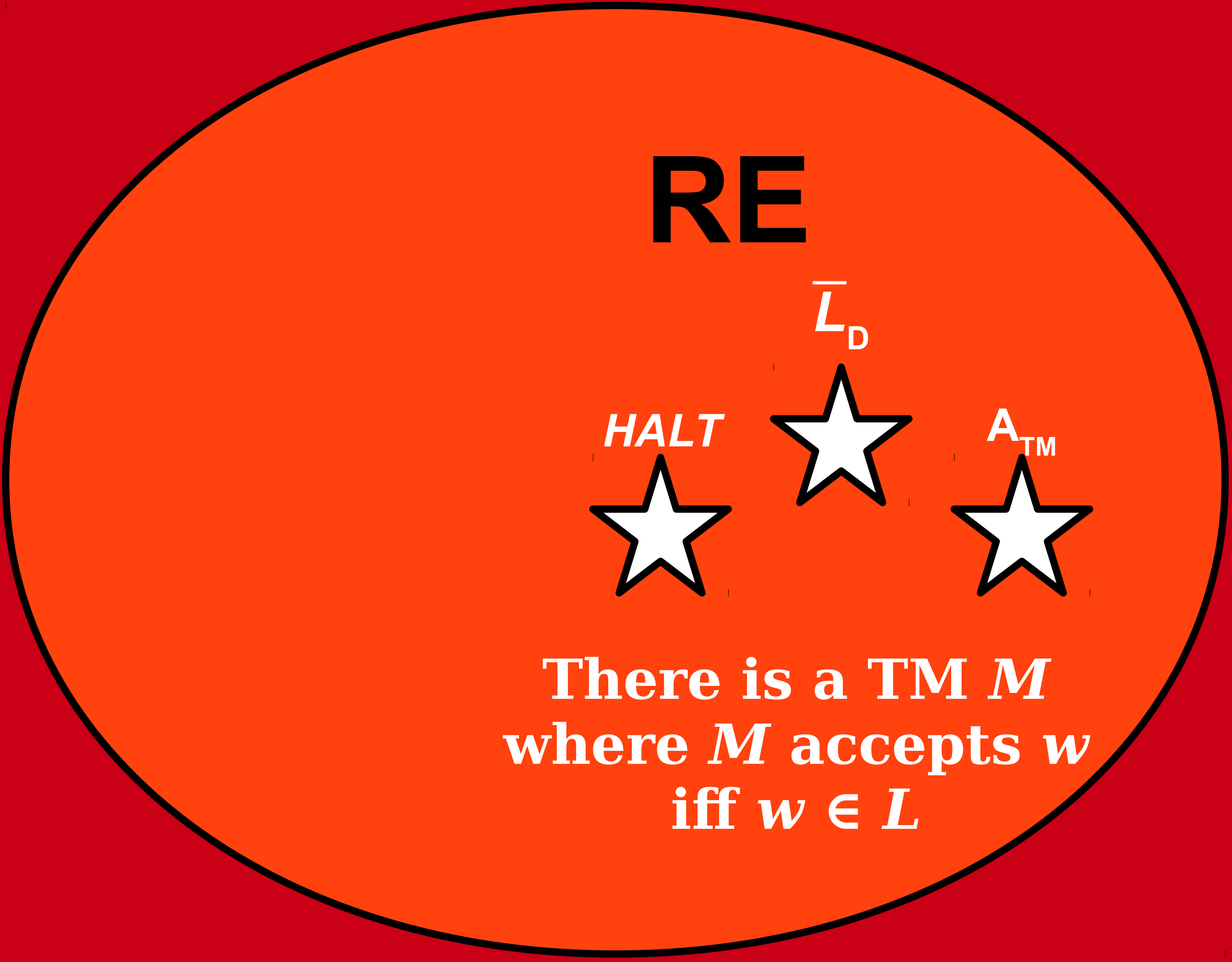
The Limits of Computability



The Limits of Computability

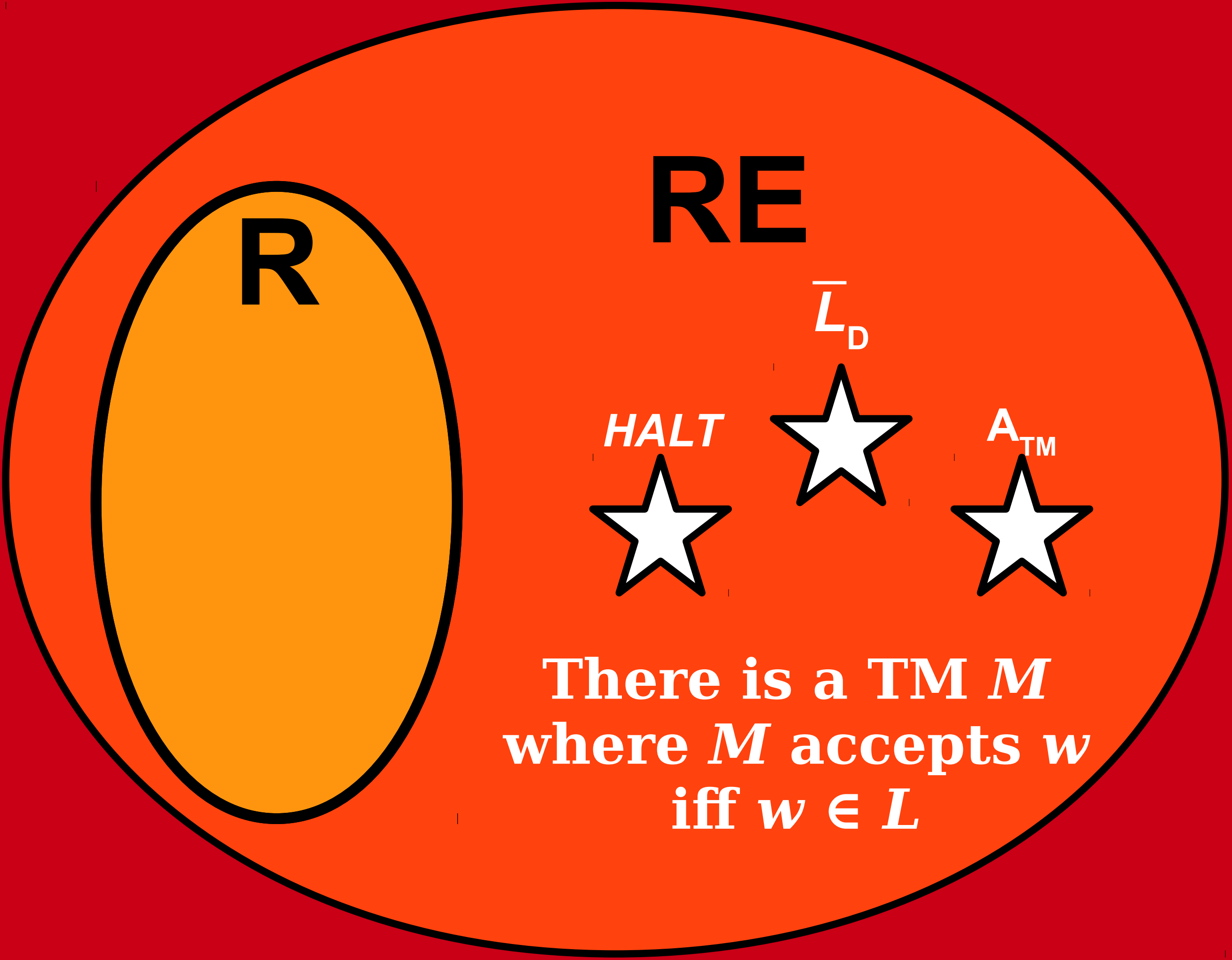


The Limits of Computability

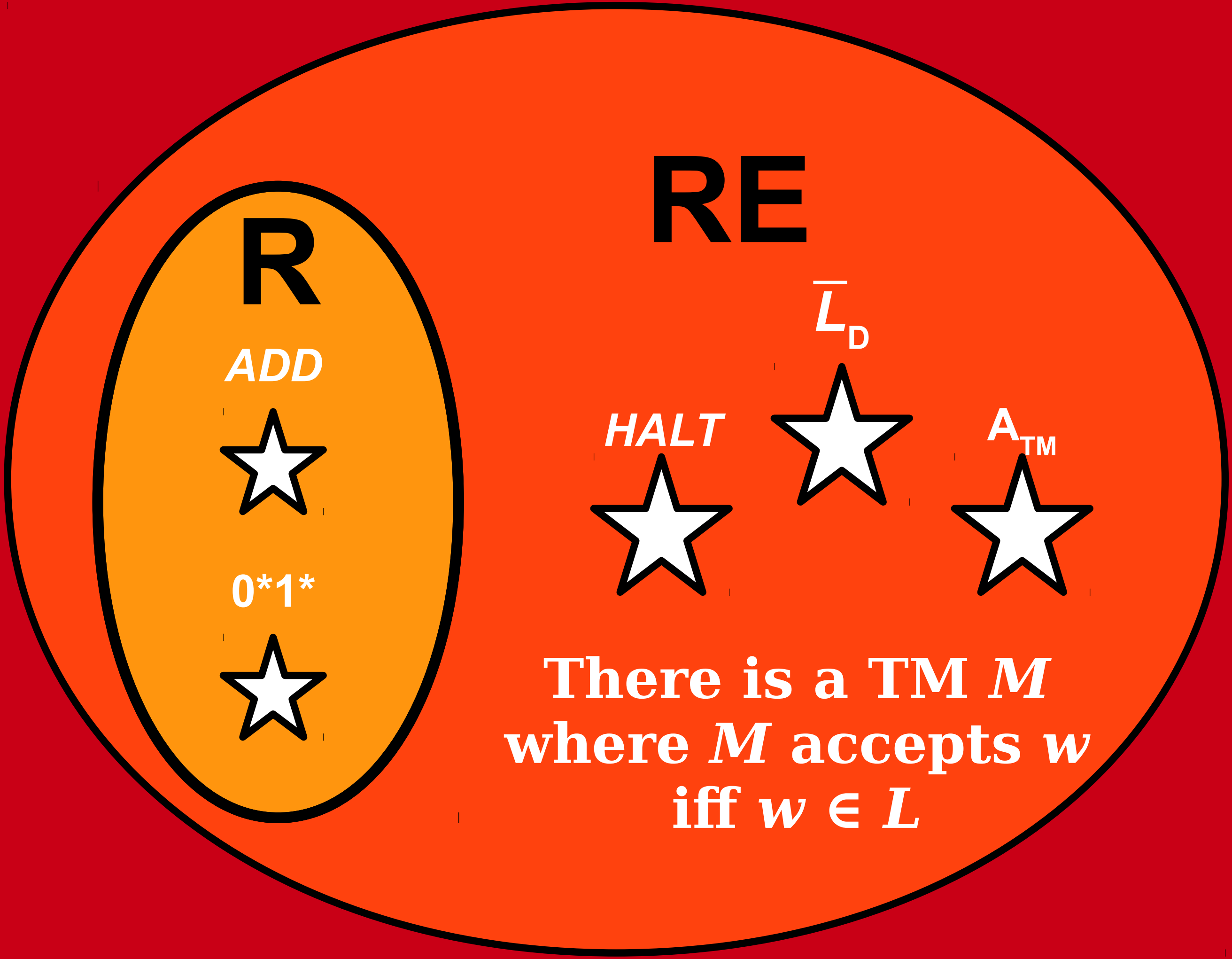


There is a TM *M* where *M* accepts *w* iff *w* ∈ *L*

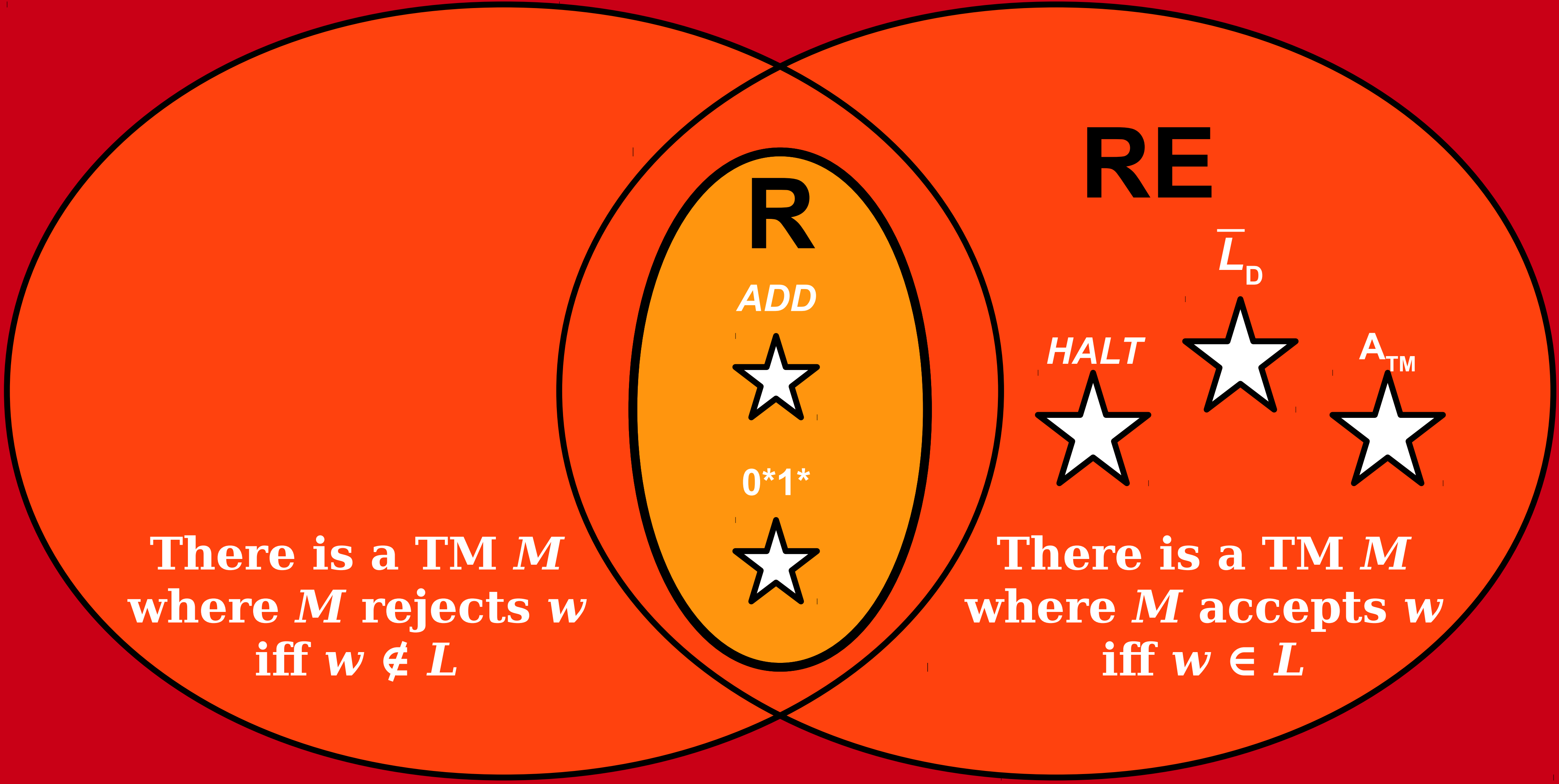
The Limits of Computability



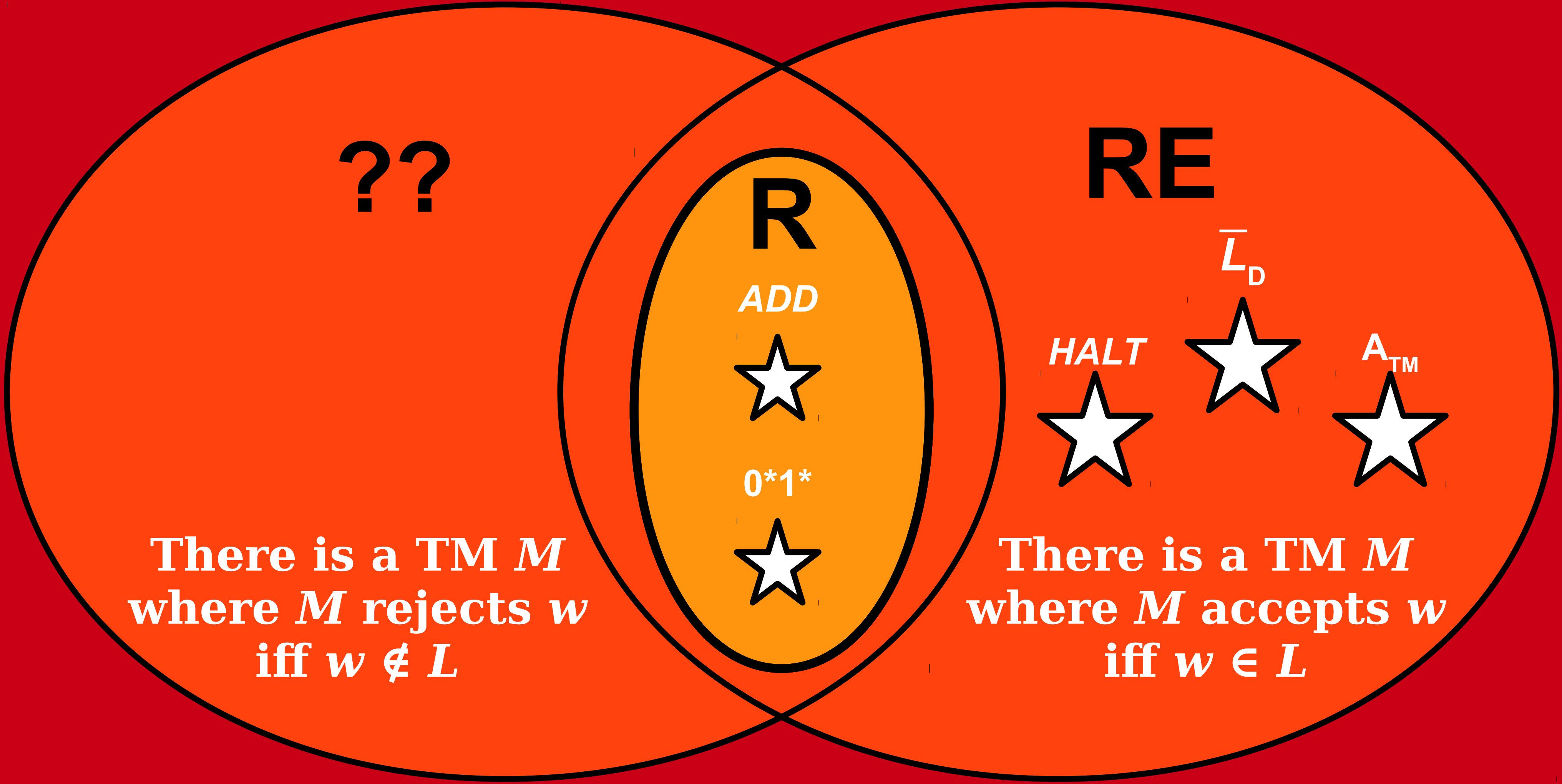
The Limits of Computability



The Limits of Computability


R
ADD
0*1*
RE
$\overline{L}_D$
HALT
A_{TM}
There is a TM M where M rejects w iff $w \notin L$
There is a TM M where M accepts w iff $w \in L$

The Limits of Computability


??
R
RE
ADD
0*1*
HALT
$\overline{L}_D$
A_{TM}
There is a TM *M* where *M* rejects *w* iff *w* ∉ *L*
There is a TM *M* where *M* accepts *w* iff *w* ∈ *L*

A New Complexity Class

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

RE and co-**RE**

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

Another View of co-**RE**

- ***Theorem:*** A language L is in co-**RE** if and only if $\overline{L}$ is in **RE**.
- ***Proof Sketch:*** If L is in co-**RE**, there's a TM that can reject everything not in L. Equivalently, this machine rejects everything in $\overline{L}$. So just switch the accept and reject states of the TM, and now it accepts exactly the strings in $\overline{L}$. ■

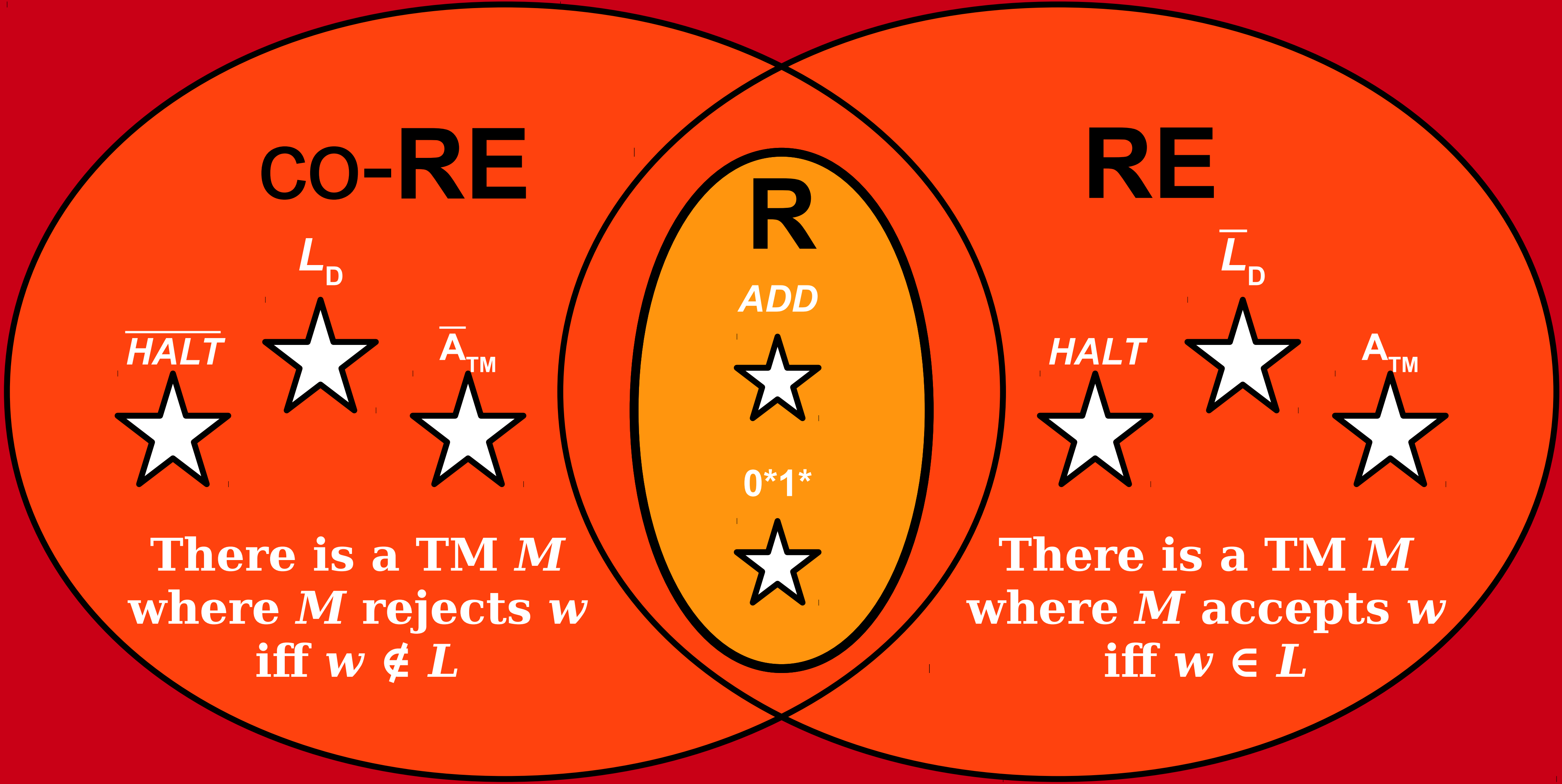
RE and co-**RE** Languages

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

RE and co-**RE** Languages

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

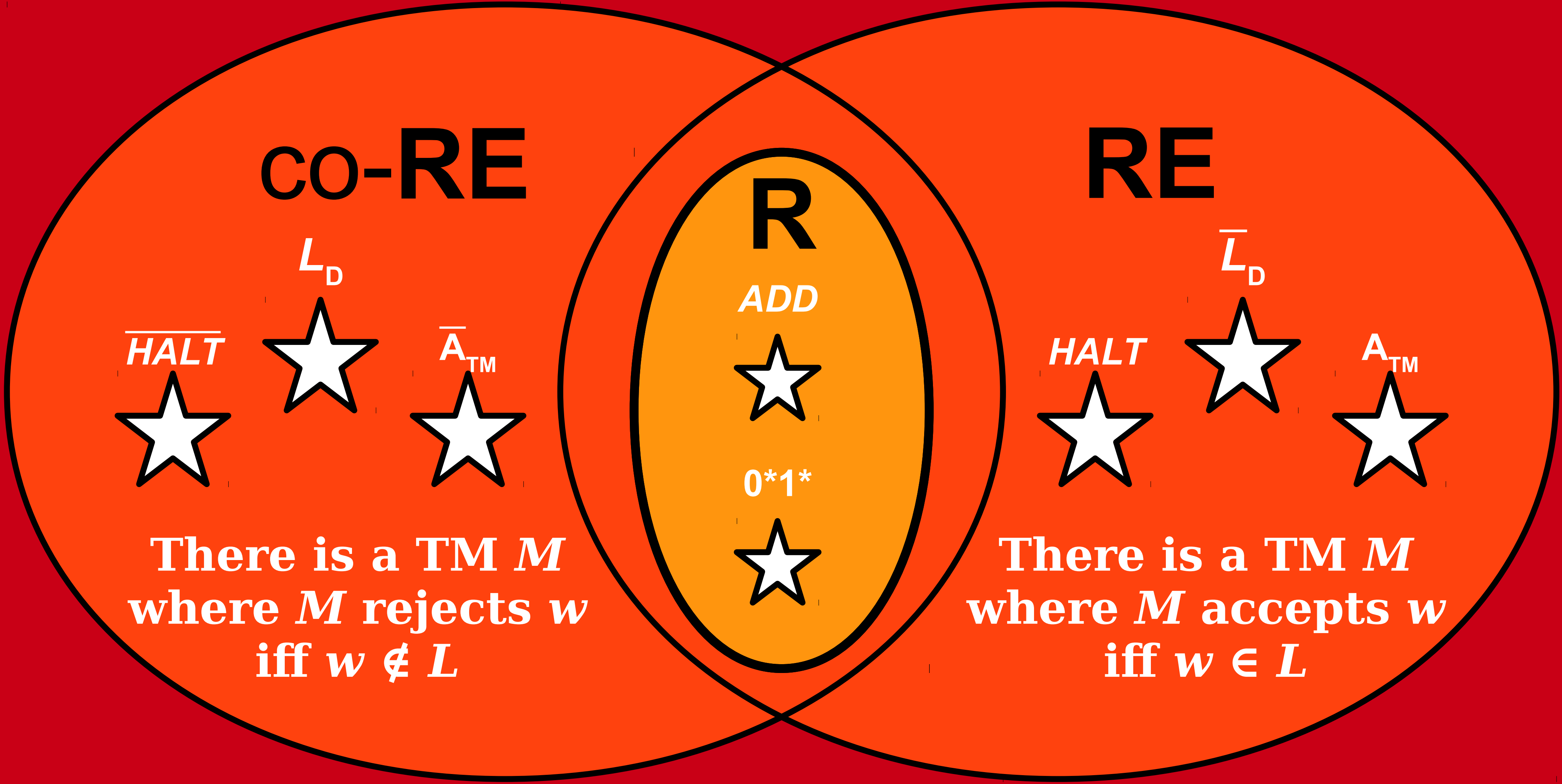
The Limits of Computability


co-RE
L_D
$\overline{HALT}$
$\overline{A}_{TM}$
R
ADD
0*1*
RE
$\overline{L}_D$
HALT
A_{TM}
There is a TM *M* where *M* rejects *w* iff *w* ∉ *L*
There is a TM *M* where *M* accepts *w* iff *w* ∈ *L*

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

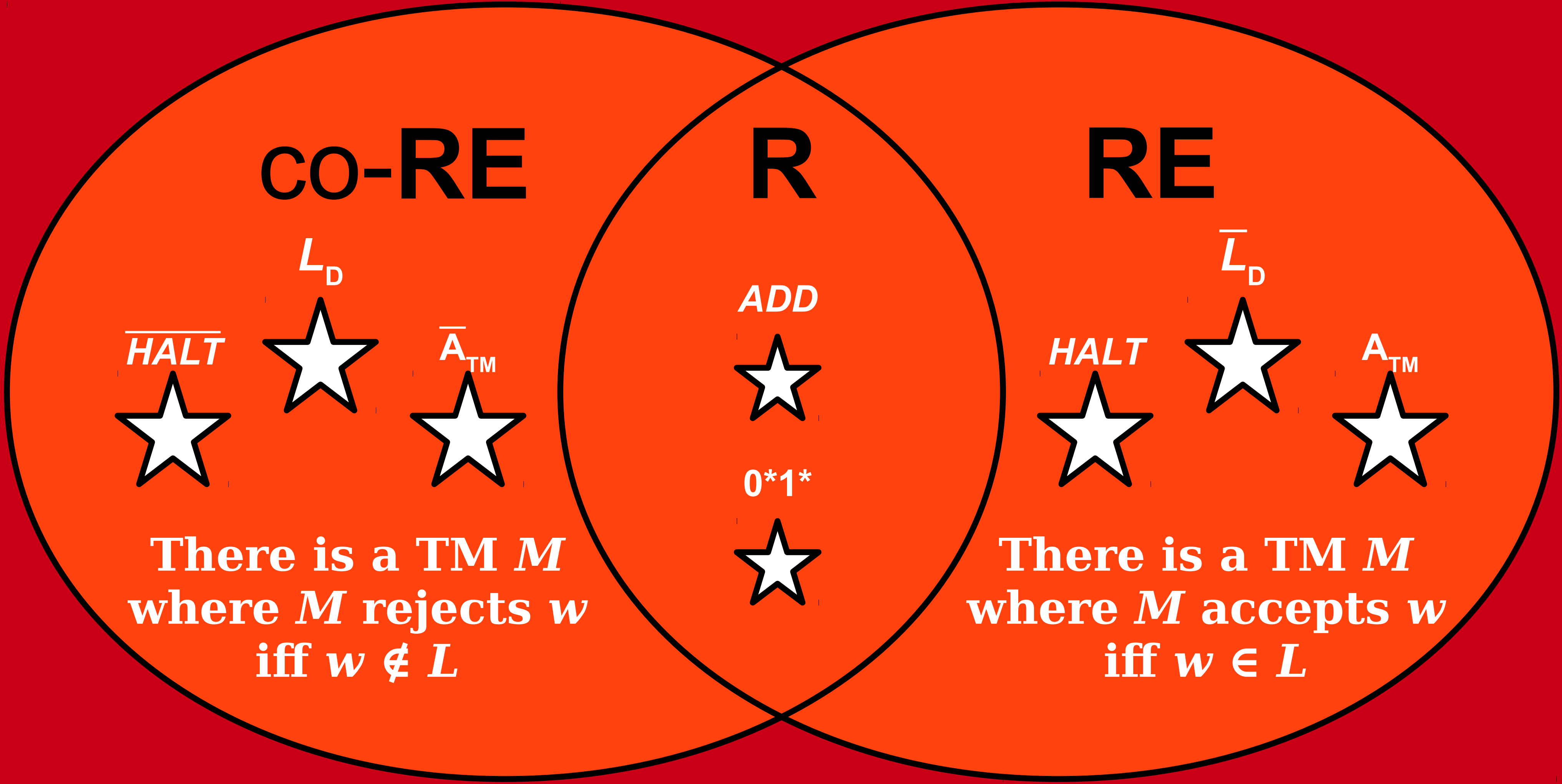
- Every language in **R** is in both **RE** and co-**RE**.
- Why?
  - A decider for L accepts all $w \in L$ and rejects all $w \notin L$.
- In other words, **R** $\subseteq$ **RE** $\cap$ co-**RE**.
- **Question:** Does **R** $=$ **RE** $\cap$ co-**RE**?

Which Picture is Correct?



co-RE
R
RE
L_D
$\overline{HALT}$
$\overline{A}_{TM}$
ADD
0*1*
$\overline{L}_D$
HALT
A_{TM}
There is a TM *M* where *M* rejects *w* iff *w* ∉ *L*
There is a TM *M* where *M* accepts *w* iff *w* ∈ *L*

Which Picture is Correct?

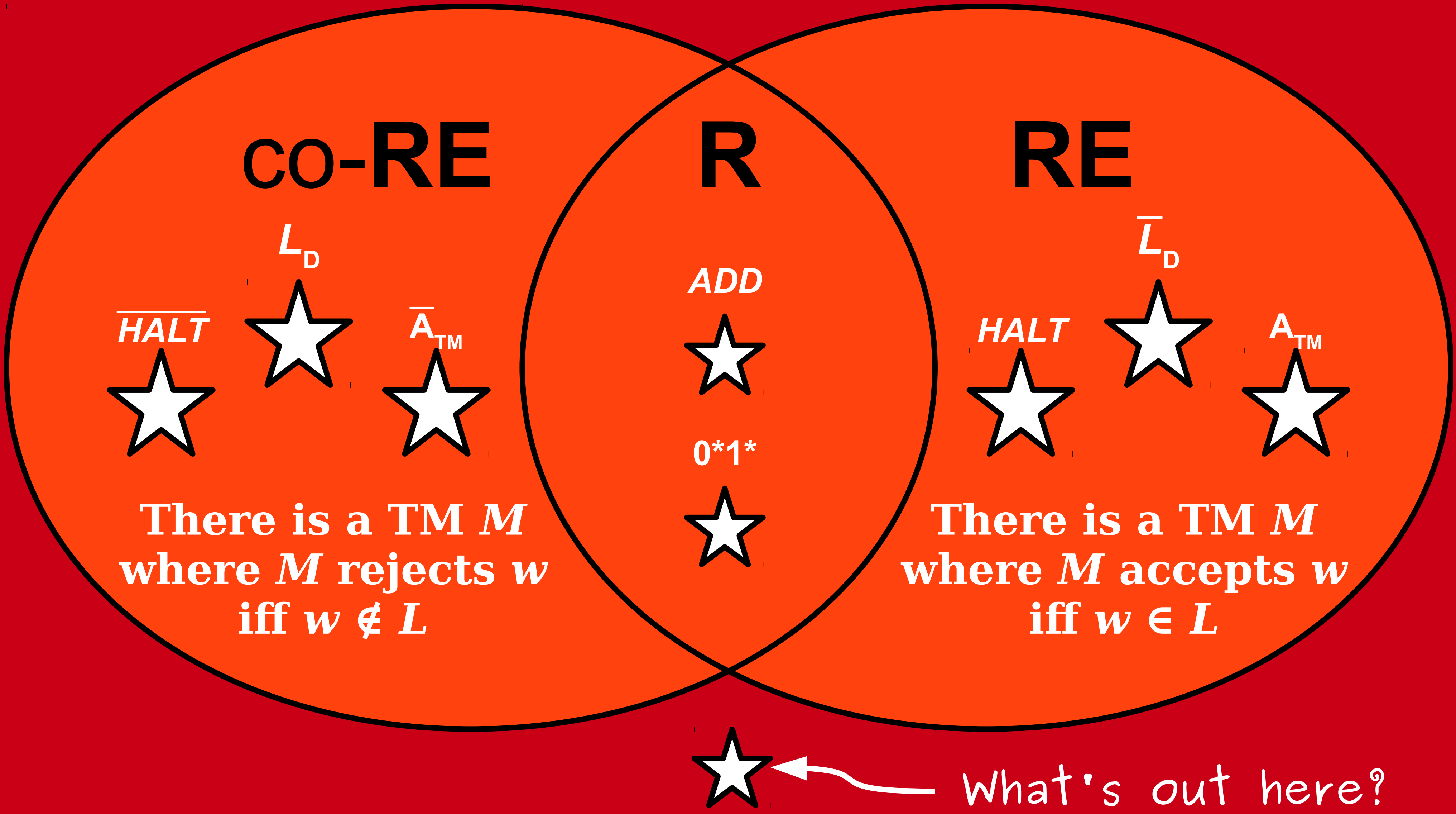

co-RE
R
RE
L_D
$\overline{HALT}$
$\overline{A}_{TM}$
ADD
0*1*
$\overline{L}_D$
HALT
A_{TM}
There is a TM *M* where *M* rejects *w* iff *w* ∉ *L*
There is a TM *M* where *M* accepts *w* iff *w* ∈ *L*

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

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

```
D = “On input w:
          Run M on w and M̄ on w in parallel
             by alternatingly simulating one
             step of M and one step of M̄.
          If M accepts w, accept.
          If M̄ rejects w, reject.
```

The Limits of Computability


co-RE
R
RE
L_D
$\overline{HALT}$
$\overline{A}_{TM}$
ADD
0*1*
$\overline{L}_D$
HALT
A_{TM}
There is a TM M where M rejects w iff $w \notin L$
There is a TM M where M accepts w iff $w \in L$
What's out here?

Next Time

- **The Complement of A_{TM}**
  - How hard is this problem?
- **Monster Problems**
  - Problems vastly beyond the reach of computing power.
- **Verifiers**
  - A different perspective on **RE** languages.
- **Nondeterministic TMs**
  - We've “conveniently” ignored what happens when you add nondeterminism to TMs. What do they do?