Unsolvable Problems

Part Two

Outline for Today

- ***Recap from Last Time***
  - Where are we, again?
- ***A Different Perspective on RE***
  - What exactly does “recognizability” mean?
- ***Verifiers***
  - A different perspective on the **RE** languages.
- ***Beyond RE***
  - Monstrously hard problems!

Recap from Last Time

Self-Referential Programs

- ***Claim:*** Any program can be augmented to include a method called `mySource()` that returns a string representation of its source code.
- ***Theorem:*** It it possible to build Turing machines that get their own encodings and perform arbitrary computations on them.

What does this program do?

```
bool willAccept(string program, string input) {
   /* … some implementation … */
}

int main() {
   string me = mySource();
   string input = getInput();

   if (willAccept(me, input)) {
      reject();
   } else {
      accept();
   }
}
```

What happens if…

… this program accepts its input?
It rejects the input!

… this program doesn't accept its input?
It accepts the input!

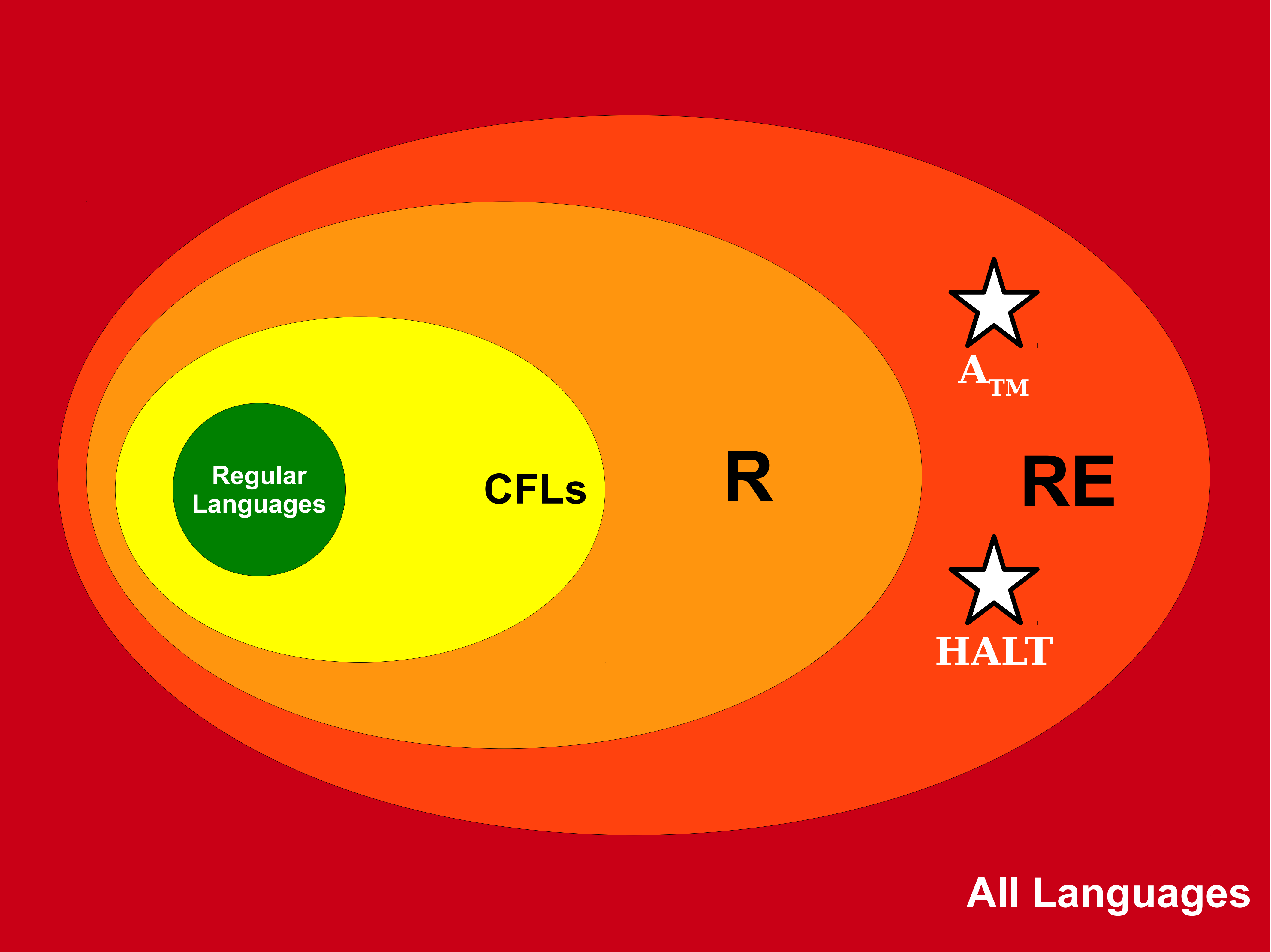
What does this program do?

```
bool willHalt(string program, string input) {
   /* … some implementation … */
}

int main() {
   string me = mySource();
   string input = getInput();

   if (willHalt(me, input)) {
      while (true) {
         // loop infinitely
      }
   } else {
      accept();
   }
}
```

What happens if…

… this program halts on this input?
It loops on the input!

… this program loops on this input?
It halts on the input!



Regular Languages

CFLs

R

A_{TM}

RE

HALT

All Languages

New Stuff!

Beyond **R** and **RE**

Beyond **R** and **RE**

- We've now seen how to use self-reference as a tool for showing undecidability (finding languages not in **R**).
- We still have not broken out of **RE** yet, though.
- To do so, we will need to build up a better intuition for the class **RE**.

What exactly is the class **RE**?

RE, Formally

- Recall that the class **RE** is the class of all recognizable languages:

$$\mathbf{RE} = \{\ L \mid \text{there is a TM } M \text{ where } \mathscr{L}(M) = L\ \}$$

- Since **R** ≠ **RE**, there is no general way to “solve” problems in the class **RE**, if by “solve” you mean “make a computer program that can always tell you the correct answer.”
- So what exactly *are* the sorts of languages in **RE**?

Key Intuition:

A language L is in **RE** if, for any string w, if you are *convinced* that $w \in L$, there is some way you could prove that to someone else.

Verification

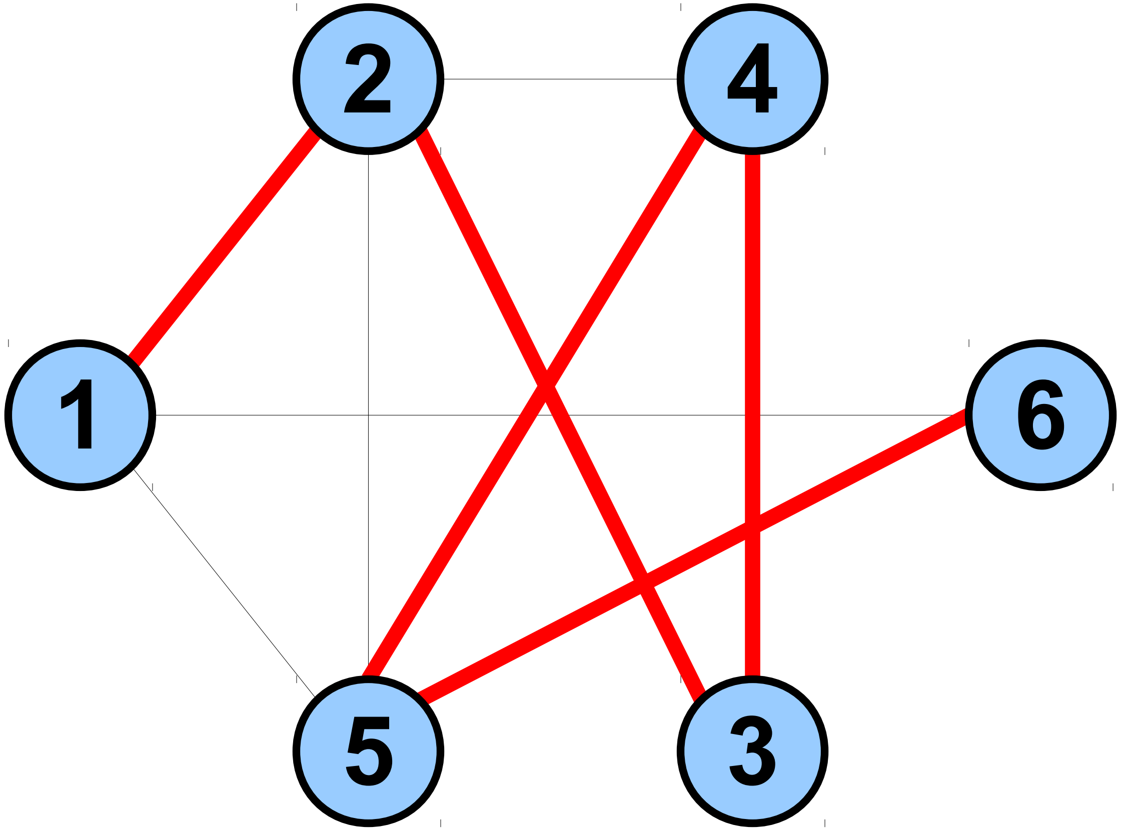
- ***Recall:*** When focusing on the **RE** languages, we need to abandon the idea that we can “solve” the problems we're looking at.
- Rather than *solving problems*, we can think about *checking answers*.

Verification

2	5	**7**	9	**6**	4	**1**	8	3
4	9	1	8	7	**3**	6	**5**	**2**
3	8	6	**1**	2	**5**	**9**	4	**7**
6	4	**5**	7	**3**	2	**8**	1	**9**
7	**1**	9	5	4	8	3	**2**	6
8	3	**2**	6	**1**	9	**5**	7	**4**
1	6	**3**	**2**	5	**7**	4	9	**8**
5	**7**	8	**4**	9	6	2	3	1
9	2	**4**	3	**8**	1	**7**	6	5

Does this Sudoku puzzle
have a solution?

Verification



Is there a simple path that goes through every node exactly once?

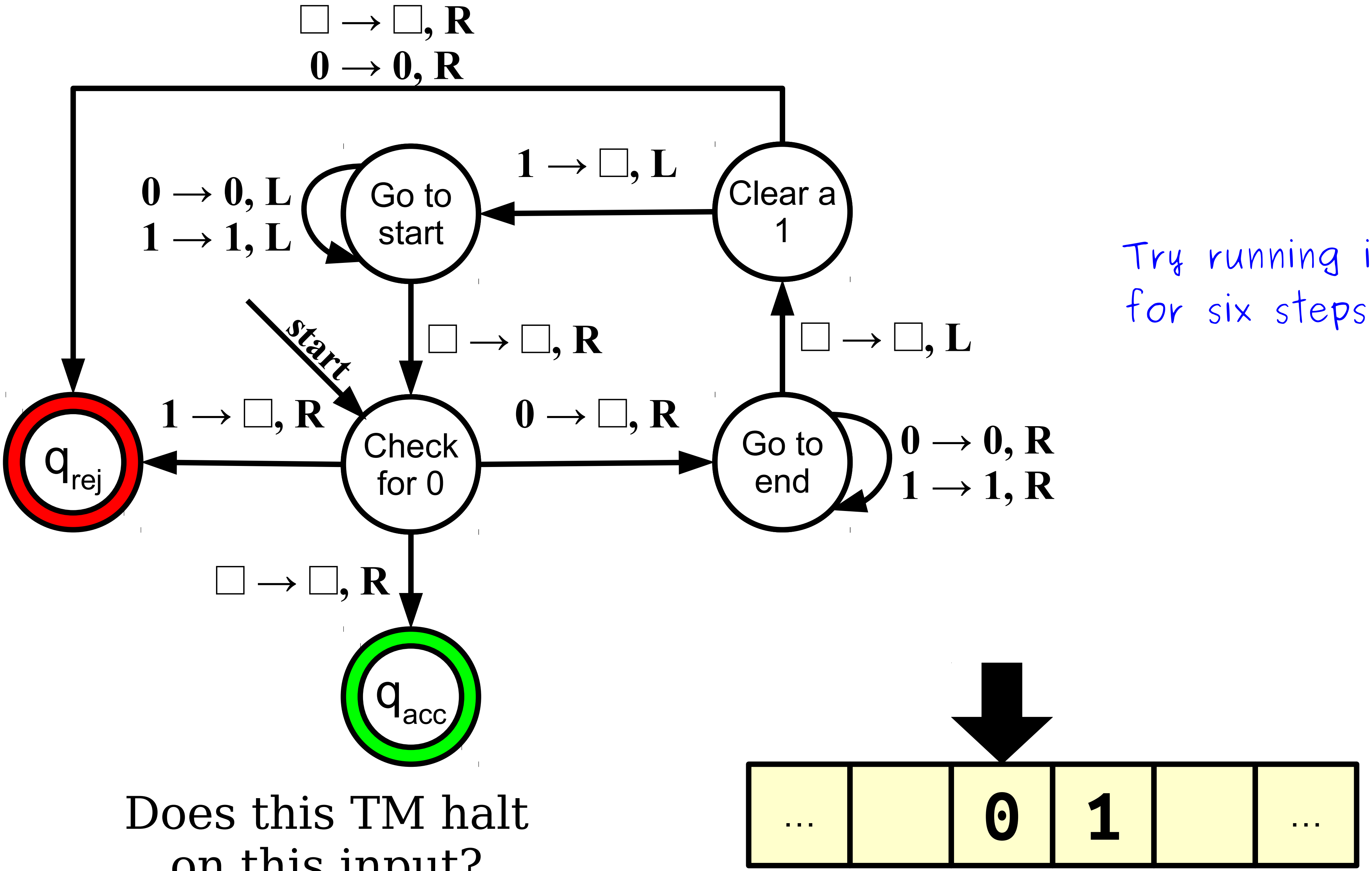
Verification

11

Try running fourteen steps of the Hailstone sequence.

Does the hailstone sequence terminate for this number?

Verification



…		0	1		…

Try running it for six steps.

Does this TM halt on this input?

Verification

- In each of the preceding cases, we were given some problem and some evidence supporting the claim that the answer is “yes.”
- Given the correct evidence, we can be certain that the answer is indeed “yes.”
- Given incorrect evidence, we aren't sure whether the answer is “yes.”
  - Maybe there's *no* evidence saying that the answer is “yes,” or maybe there is some evidence, but just not the evidence we were given.
- Let's formalize this idea.

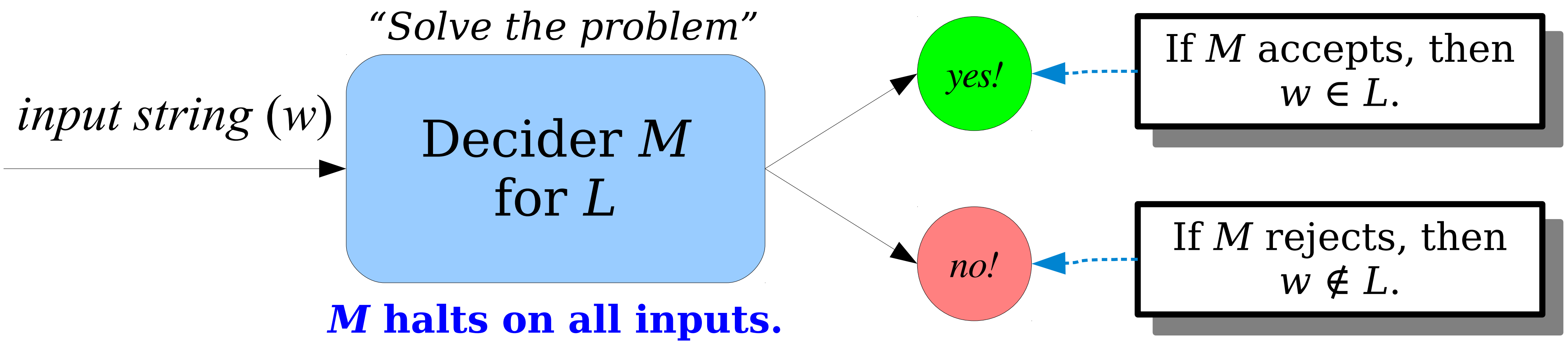
Verifiers

- A ***verifier*** for a language L is a TM V with the following properties:
  - V halts on all inputs.
  - For any string $w \in \Sigma^*$, the following is true:

$$\boldsymbol{w \in L \quad \leftrightarrow \quad \exists c \in \Sigma^*.\ V \text{ accepts } \langle w, c \rangle}$$

- A string c where V accepts $\langle w, c \rangle$ is called a ***certificate*** for w.
- Intuitively, what does this mean?

Deciders and Verifiers



“Solve the problem”

input string (w)

Decider M for L

yes!

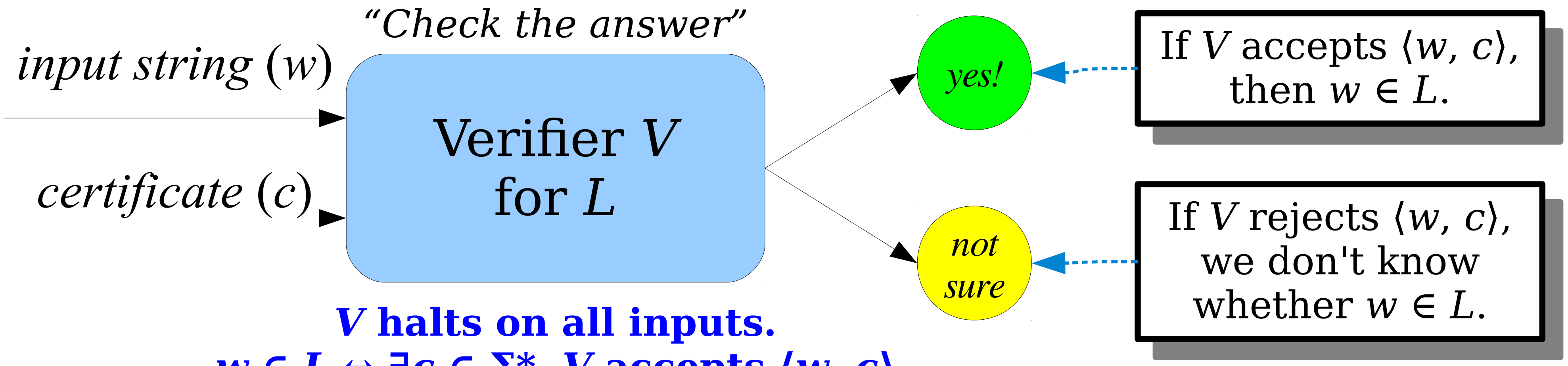
no!

If M accepts, then $w \in L$.

If M rejects, then $w \notin L$.

M halts on all inputs.

$w \in L \leftrightarrow M$ accepts w



“Check the answer”

input string (w)

certificate (c)

Verifier V for L

yes!

not sure

If V accepts $\langle w, c \rangle$, then $w \in L$.

If V rejects $\langle w, c \rangle$, we don't know whether $w \in L$.

V halts on all inputs.

$w \in L \leftrightarrow \exists c \in \Sigma^*.\ V$ accepts $\langle w, c \rangle$

Verifiers

- A ***verifier*** for a language L is a TM V with the following properties:
  - V halts on all inputs.
  - For any string $w \in \Sigma^*$, the following is true:

$$\boldsymbol{w \in L \quad \leftrightarrow \quad \exists c \in \Sigma^*.\ V \text{ accepts } \langle w, c\rangle}$$

- Some notes about V:
  - If V accepts $\langle w, c\rangle$, then we're guaranteed $w \in L$.
  - If V does not accept $\langle w, c\rangle$, then either
    - $w \in L$, but you gave the wrong c, or
    - $w \notin L$, so no possible c will work.

Verifiers

- A ***verifier*** for a language L is a TM V with the following properties:
  - V halts on all inputs.
  - For any string $w \in \Sigma^*$, the following is true:

$$\boldsymbol{w \in L \quad \leftrightarrow \quad \exists c \in \Sigma^*.\ V \text{ accepts } \langle w, c \rangle}$$

- Some notes about V:
  - Notice that c is existentially quantified. Any string $w \in L$ must have at least one c that causes V to accept, and possibly more.
  - V is required to halt, so given any potential certificate c for w, you can check whether the certificate is correct.

Verifiers

- A ***verifier*** for a language L is a TM V with the following properties:
  - V halts on all inputs.
  - For any string $w \in \Sigma^*$, the following is true:

$$\boldsymbol{w \in L \quad \leftrightarrow \quad \exists c \in \Sigma^*.\ V \text{ accepts } \langle w, c\rangle}$$

- Some notes about V:
  - Notice that $\mathscr{L}(V) \neq L$. *(Good question: what <u>is</u> $\mathscr{L}(V)$?)*
  - The job of V is just to check certificates, not to decide membership in L.

Some Verifiers

- Let L be the following language:

$$L = \{\ \langle n \rangle \mid n \in \mathbb{N} \text{ and the hailstone sequence terminates for } n\ \}$$

- Let's see how to build a verifier for L.

Some Verifiers

- Let L be the following language:

$$L = \{\ \langle n \rangle \mid n \in \mathbb{N} \text{ and the hailstone sequence terminates for } n\ \}$$

```
private boolean checkHailstone(int n, int k) {
    for (int i = 0; i < k; i++) {
        if (n % 2 == 0) n /= 2;
        else n = 3*n + 1;
    }
    return n == 1;
}
```

- Do you see why $\langle n \rangle \in L$ iff there is some k such that `checkHailstone(n, k)` returns true?
- Do you see why `checkHailstone` always halts?

Some Verifiers

- Consider *HALT*:

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

- Let's see how to build a verifier for *HALT*.

Some Verifiers

- Consider *HALT*:

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

```
private boolean checkHalt(TM M, string w, int k) {
    simulate M running on w for k steps;
    if (M is in an accepting state) return true;
    if (M is in a rejecting state) return true;
    return false;
}
```

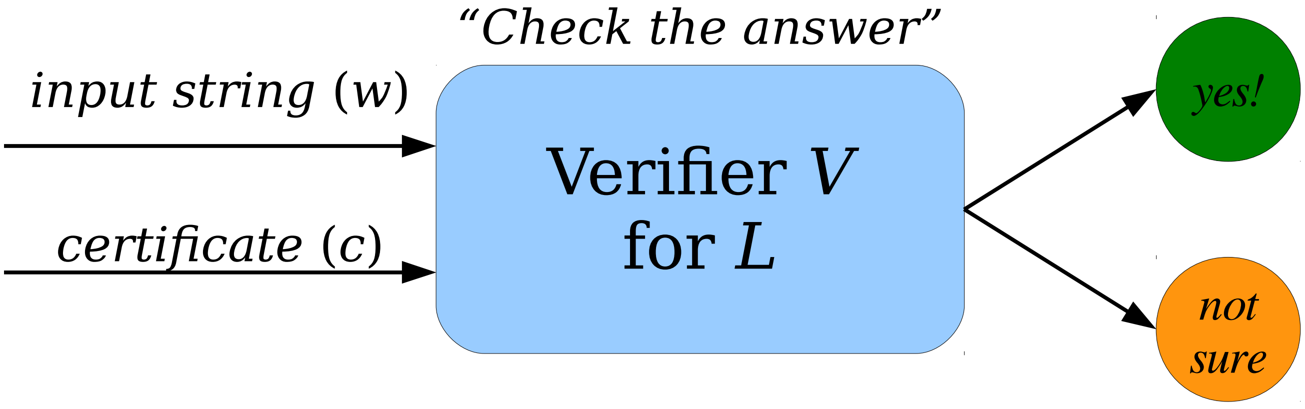
- Do you see why *M* halts on *w* iff there is some *k* such that `checkHalt(M, w, k)` returns true?
- Do you see why `checkHalt` always halts?

What languages are verifiable?

Theorem: If L is a language, then there is a verifier for L if and only if $L \in \mathbf{RE}$.

Verifiers and **RE**

- ***Theorem:*** If there is a verifier V for a language L, then $L \in$ **RE**.
- ***Proof goal:*** Given a verifier V for a language L, find a way to construct a recognizer M for L.



"Check the answer"

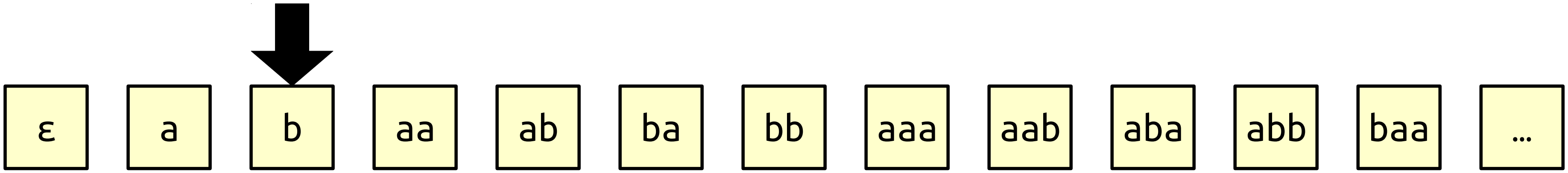
input string (w)

certificate (c)

Verifier V for L

yes!

not sure



ε	a	b	aa	ab	ba	bb	aaa	aab	aba	abb	baa	…

Verifiers and **RE**

- ***Theorem:*** If there is a verifier V for a language L, then $L \in$ **RE**.
- ***Proof idea:*** Build a recognizer that tries every possible certificate to see if $w \in L$.
- ***Proof sketch:*** Consider this program:

```
private boolean isInL(string w) {
    for (i from 0 to ∞) {
       for (each string c of length i) {
           if (V accepts ⟨w, c⟩) return true;
       }
    }
}
```

If $w \in L$, then there is a $c \in \Sigma^*$ such that V accepts $\langle w, c \rangle$. This program tries all possible strings as certificates, so it eventually finds c, watches V accept $\langle w, c \rangle$, then and accepts w. If $w \notin L$, there is no $c \in \Sigma^*$ where V accepts $\langle w, c \rangle$, so this program loops on w. ■

Verifiers and **RE**

- ***Theorem:*** If $L \in$ **RE**, then there is a verifier for L.
- ***Proof goal:*** Beginning with a recognizer M for the language L, show how to construct a verifier V for L.
- The machine M will accept w if $w \in L$. If M doesn't accept w, then $w \notin L$.
- A certificate is supposed to be some sort of “proof” that the string is in the language. Since the only thing we know about L is that M is a recognizer for it, our certificate would have to tell us something about what M does.
- We need to choose a certificate with the following properties:
    - We can decide in finite time whether a certificate is right or wrong.
    - A “good” certificate proves that $w \in L$ (meaning M accepts w)
    - A “bad” certificate never proves $w \in L$.
- ***Idea:*** If M accepts w, it will do so in finitely many steps. What if our certificate is the number of steps?

Verifiers and **RE**

- ***Theorem:*** If $L \in$ **RE**, then there is a verifier for L.
- ***Proof sketch:*** Let L be an **RE** language and let M be a recognizer for it. Then show that this is a verifier for L:

```
private boolean checkInL(string w, int k) {
    run M on w for k steps;
    if (M is in an accepting state) return true;
    else return false;
}
```

If $w \in L$, then M accepts w in some number of steps (call it c). Calling `checkInL(w, c)` then returns true. Conversely, if there is a k where `checkInL(w, k)` returns true, then M accepts w, so $w \in L$. ■

RE and Proofs

- Verifiers and recognizers give two different perspectives on the “proof” intuition for **RE**.
- Verifiers are explicitly built to check proofs that strings are in the language.
    - If you know that some string w belongs to the language and you have the proof of it, you can convince someone else that $w \in L$.
- You can think of a recognizer as a device that “searches” for a proof that $w \in L$.
    - If it finds it, great!
    - If not, it might loop forever.

RE and Proofs

- If the **RE** languages represent languages where membership can be proven, what does a non-**RE** language look like?
- Intuitively, a language is *not* in **RE** if there is no general way to prove that a given string $w \in L$ actually belongs to L.
- In other words, even if you knew that a string was in the language, you may never be able to convince anyone of it!

Time-Out for Announcements!

Second Midterm Graded

- We’ve graded the second midterm exam and emailed out scores yesterday afternoon.
- The exams themselves are available for pickup after class today.
  - Reading this online? Unclaimed exams will be in the first floor of the Gates building. Enter through the side entrance next to Herrin marked “Stanford Engineering Venture Fund Laboratories” and look in the filing cabinets to your right.
- Overall, we are extremely impressed by how everyone did on this exam. More on that later!

Problem Sets

- Problem Set Seven solutions are now up online. We’ll return graded PS7’s later this week.
- Problem Set Eight is due on Friday.
- Problem Set Nine will go out on Friday. It will be due next ***Wednesday*** (the last day of class).
  - No late days can be used on PS9.
  - PS9 is shorter than the other problem sets this quarter.
  - We’ll get the finalized office hours timetable posted as soon as we can.

A Reminder: Honor Code

- We're coming up to the point in the quarter where we start to see a marked uptick in the number of assignments submitted that are pretty obviously copied from old solution sets.
- I know that a lot of you are stressed, tired, and worn out at this point. However, please, please, *please* don't do this. The costs are really high and it's super easy to spot.
- ***To the campus community at large:*** if someone you know (a friend, a dormmate, a problem set partner, etc.) is really suffering, please reach out to them and make sure they're okay. It's easy for people to get isolated and overwhelmed at this point in the quarter, and (from experience) that can be an awful, awful feeling.

Final Exam Logistics

- Our final exam is one week from Friday. It’ll be from 3:30PM – 6:30PM. Rooms are divvied up by last (family) name:
  - `Abb – Kan`: Go to Bishop Auditorium.
  - `Kar – Zuc`: Go to Cemex Auditorium.
- Exam is cumulative and all topics from the lectures and problem sets are fair game.
- The exam focus is roughly 50/50 between discrete math topics (PS1 – PS5) and computability/complexity topics (PS6 – PS9).
- As with the midterms, the exam is closed-book, closed-computer, and limited-note. You can bring a single, double-sided, 8.5” × 11” sheet of notes with you to the exam.
- Students with OAE accommodations: please contact us by Friday if you haven’t yet done so.

Preparing for the Exam

- Last week, we released EPP9, EPP10, and EPP11. Solutions are now available online.
- We've just released four practice final exams. They're based on past final exams, with a few minor modifications.
    - Practice Final 1: Fall 2016 final exam.
    - Practice Final 2: Winter 2016 final exam.
    - Practice Final 3: Spring 2015 final exam.
    - Practice Final 4: Fall 2015 final exam.
- Solutions will go out on Friday.
- Need some more practice? Let us know how we can help out!

Your Questions

“What's your best piece of relationship advice?”

Communication is key! From experience, open and honest communication is essential to being in a relationship. If you can't communicate openly and honestly, then things will build up, you'll misread situations, and it's easy for things to spiral out of control.

Remember that you're part of a team and that the point of a relationship is for the whole to be greater than the sum of the parts. So talk about things that matter to you, be supportive, and be accommodating, and have fun creating new life experiences with someone!

“Why are the proofwriting guidelines in this class so much different and more rigid than math classes (or standard mathematical proofs in general)?”

One of the goals of CS103 is to get you to transition from "pre-formal" mathematics to "formal" mathematics. We want you to be able to ask the question "why is this true?" in a way that really cuts to the core of the issue and forces you to ask questions like "how is this defined?" and "is this really true, or is this just something I think is true?"

Our only way to measure whether you're capable of doing this is to see whether you actually do it when we ask you to. We're precise with our grading because we want to make sure that you're asking the right questions, that you're formulating an answer in a way that calls back to definitions, and that you're not skipping over key, nontrivial steps.

All the available evidence we have suggests that this is working phenomenally well this quarter. The exam scores are higher now than they've been in years and we're seeing significantly fewer logic errors.

Downstream, classes often assume you've already made the pre-formal to formal transition, and they'll grade proofs more for insight than execution because there's a baseline belief that you could do these lower-level proofs if you needed to. But even in that case, the sorts of questions you'd need to ask to really probe the material – why is this true? how does this follow from the definition? is this actually even true? – follow from these skills.

"How can we pick ourself up in this class when it feels like everyone else does so well? Is this class similar in rigor to higher-level CS classes?"

For starters, sorry to hear that you're feeling this!

A good first step is to figure out what areas you need to work on. Find something you did right, and find something you didn't do right. Are you in a spot where you can answer why one is right and one is wrong? If not, stop by office hours and ask a clarifying question! Are you making a lot of minor sporadic errors, or is there some root underlying skill or technique you need to focus on? Get input from the course staff on where you should focus your efforts, then use the available resources (extra practice problems, CS103A materials, course reader, etc.) to practice those specific skills.

As to the rigor question – most downstream courses are less rigorous in what they expect of your proofwriting, but they will still expect you to think rigorously and precisely. Understanding what questions to ask, and how to be skeptical of claims, and knowing how to argue that some result is true will still be important, and those are skills that are closely correlated with what we're looking for.

Back to CS103!

Finding Non-**RE** Languages

Finding Non-**RE** Languages

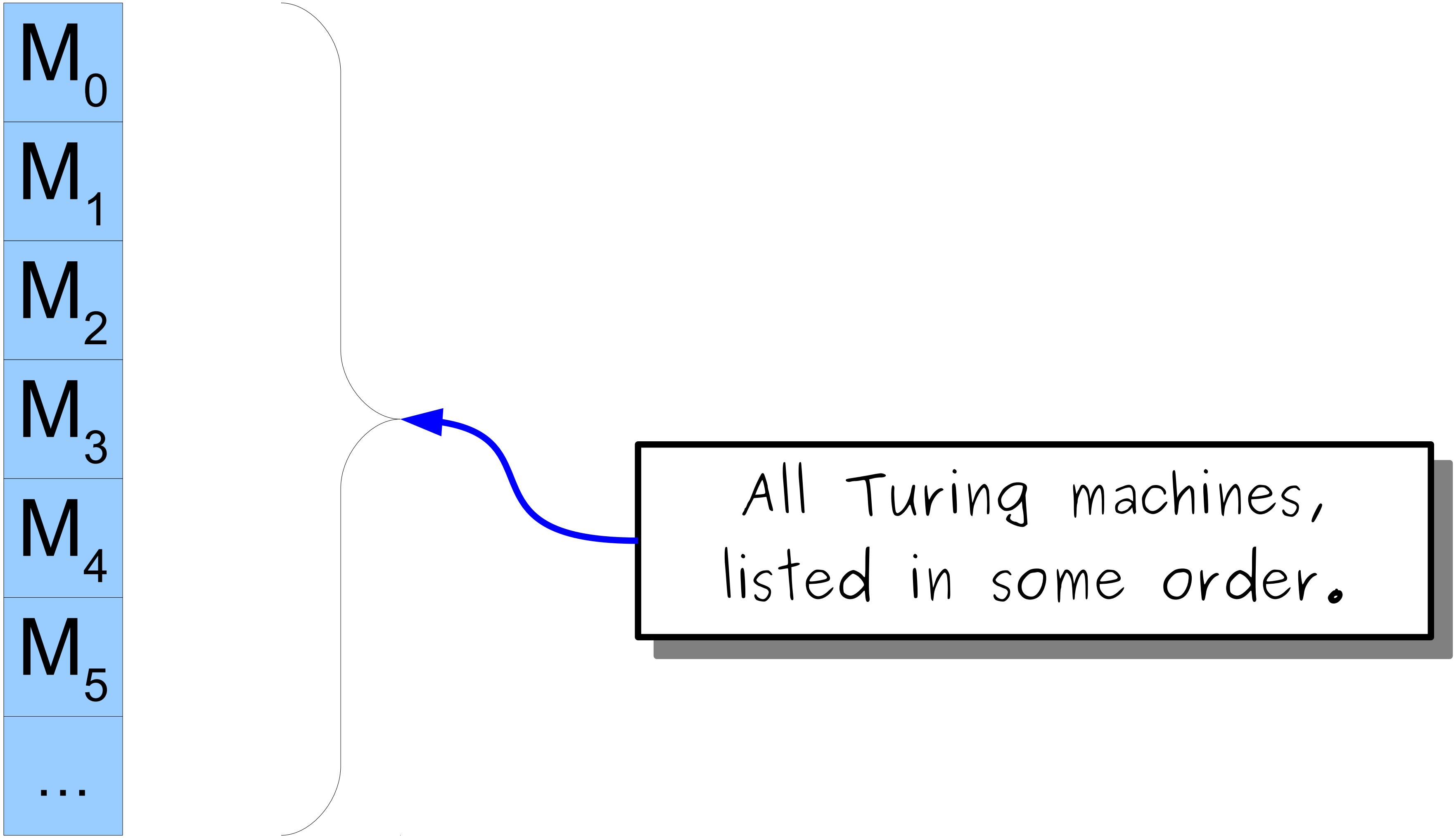
- Right now, we know that non-**RE** languages exist, but we have no idea what they look like.
- How might we find one?
- The answer brings us all the way back to our very first lecture!

Languages, TMs, and TM Encodings

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

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

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



M_0

M_1

M_2

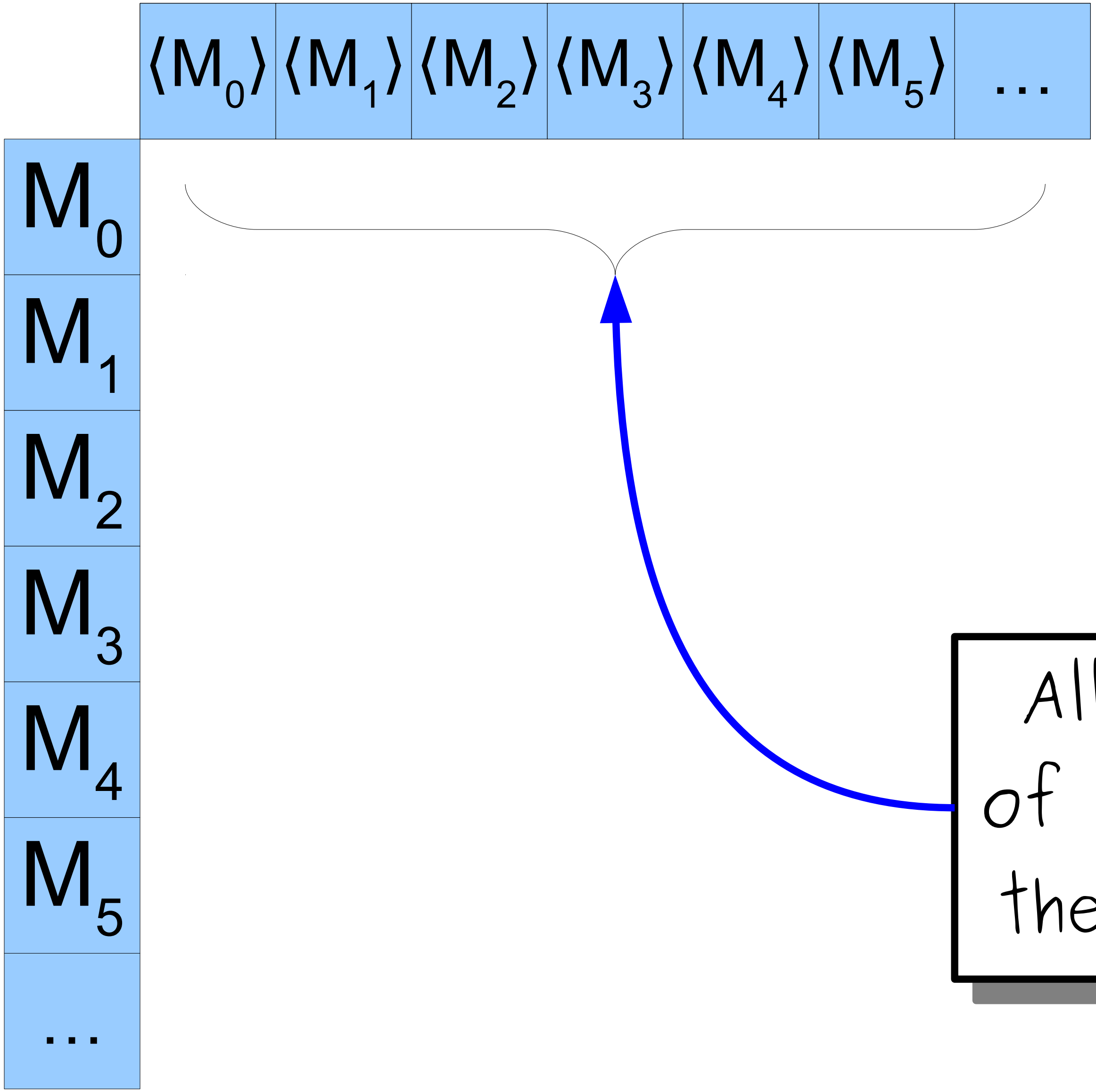
M_3

M_4

M_5

...

All Turing machines, listed in some order.



All descriptions of TMs, listed in the same order.

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

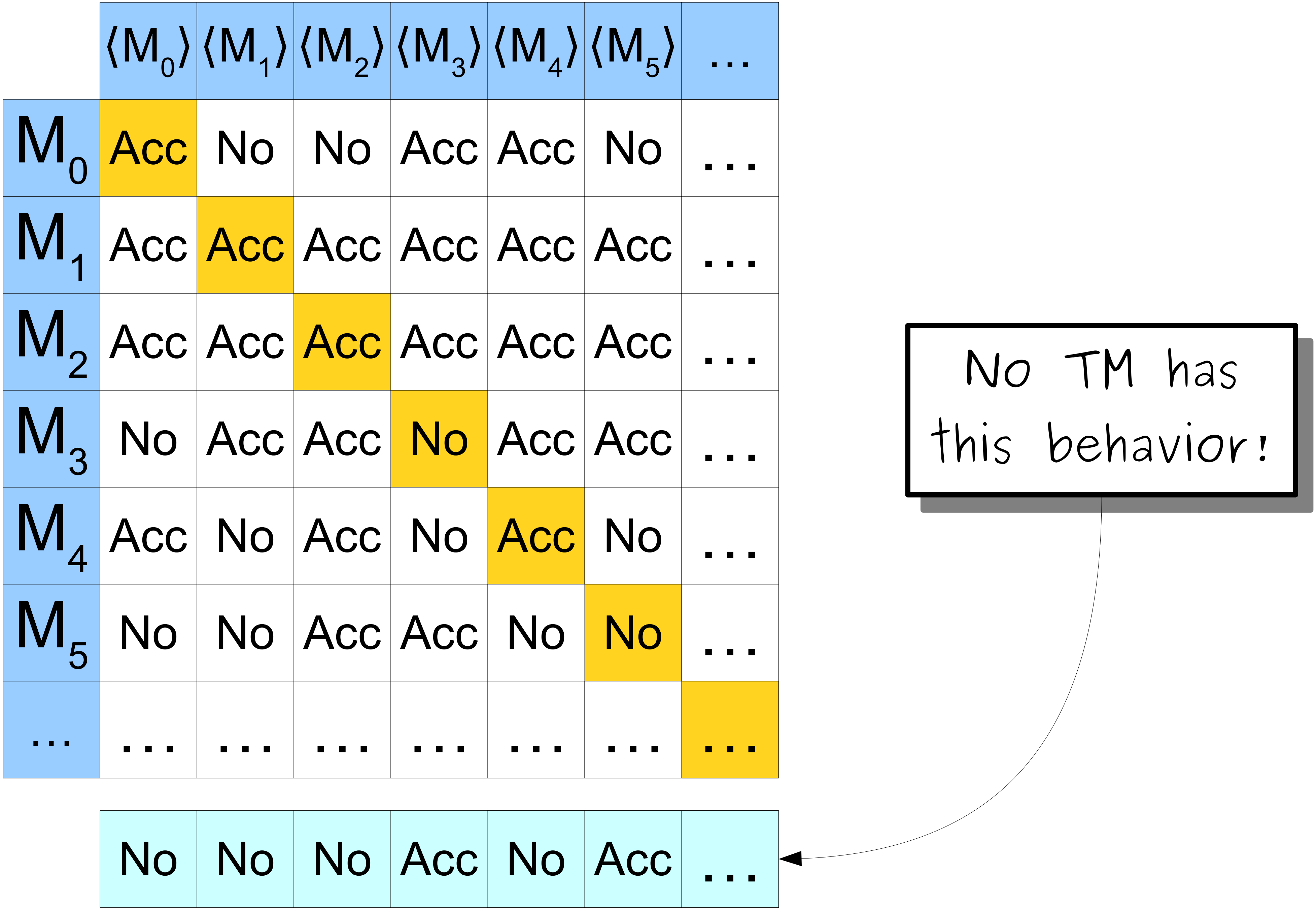
	$\langle M_0 \rangle$	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\langle M_5 \rangle$	…
M_0	Acc	No	No	Acc	Acc	No	…
M_1	Acc	Acc	Acc	Acc	Acc	Acc	…
M_2	Acc	Acc	Acc	Acc	Acc	Acc	…
M_3	No	Acc	Acc	No	Acc	Acc	…
M_4	Acc	No	Acc	No	Acc	No	…
M_5	No	No	Acc	Acc	No	No	…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

Acc	Acc	Acc	No	Acc	No	…

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

No	No	No	Acc	No	Acc	…

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



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

No	No	No	Acc	No	Acc	…

No TM has this behavior!

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

No	No	No	Acc	No	Acc	…

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

Diagonalization Revisited

- 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.

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

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

Proof: By contradiction; assume that $L_D \in$ **RE**. Then there must be some TM R such that $\mathscr{L}(R) = L_D$.

Since $\mathscr{L}(R) = L_D$, we know that if M is any TM, then

$$\langle M \rangle \in L_D \text{ iff } \langle M \rangle \in \mathscr{L}(R). \qquad (1)$$

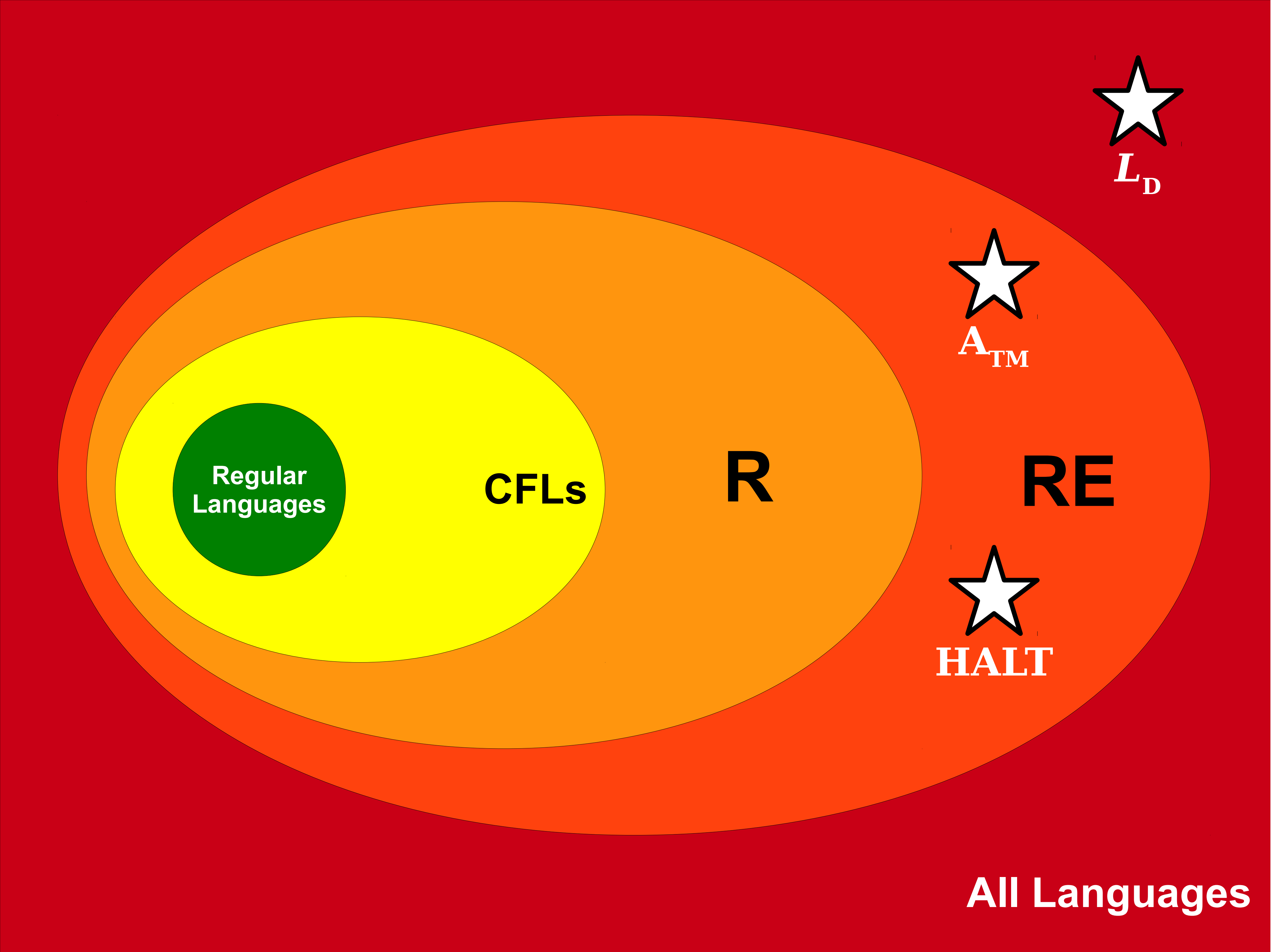
From the definition of L_D, we see that $\langle M \rangle \in L_D$ iff $\langle M \rangle \notin \mathscr{L}(M)$. Combining this with statement (1) tells us that

$$\langle M \rangle \notin \mathscr{L}(M) \text{ iff } \langle M \rangle \in \mathscr{L}(R). \qquad (2)$$

Statement (2) holds for any TM M, so in particular it should hold for R itself. This means that

$$\langle R \rangle \notin \mathscr{L}(R) \text{ iff } \langle R \rangle \in \mathscr{L}(R). \qquad (3)$$

This is clearly impossible. We have reached a contradiction, so our assumption must have been wrong. Thus $L_D \notin$ **RE**. ■


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

What This Means

- On a deeper philosophical level, the fact that non-**RE** languages exist supports the following claim:

  There are statements that are true but not provable.

- Intuitively, given any non-**RE** language, there will be some string in the language that *cannot* be proven to be in the language.
- This result can be formalized as a result called ***Gödel's incompleteness theorem***, one of the most important mathematical results of all time.
- Want to learn more? Take Phil 152 or CS154!

What This Means

- On a more philosophical note, you could interpret the previous result in the following way:

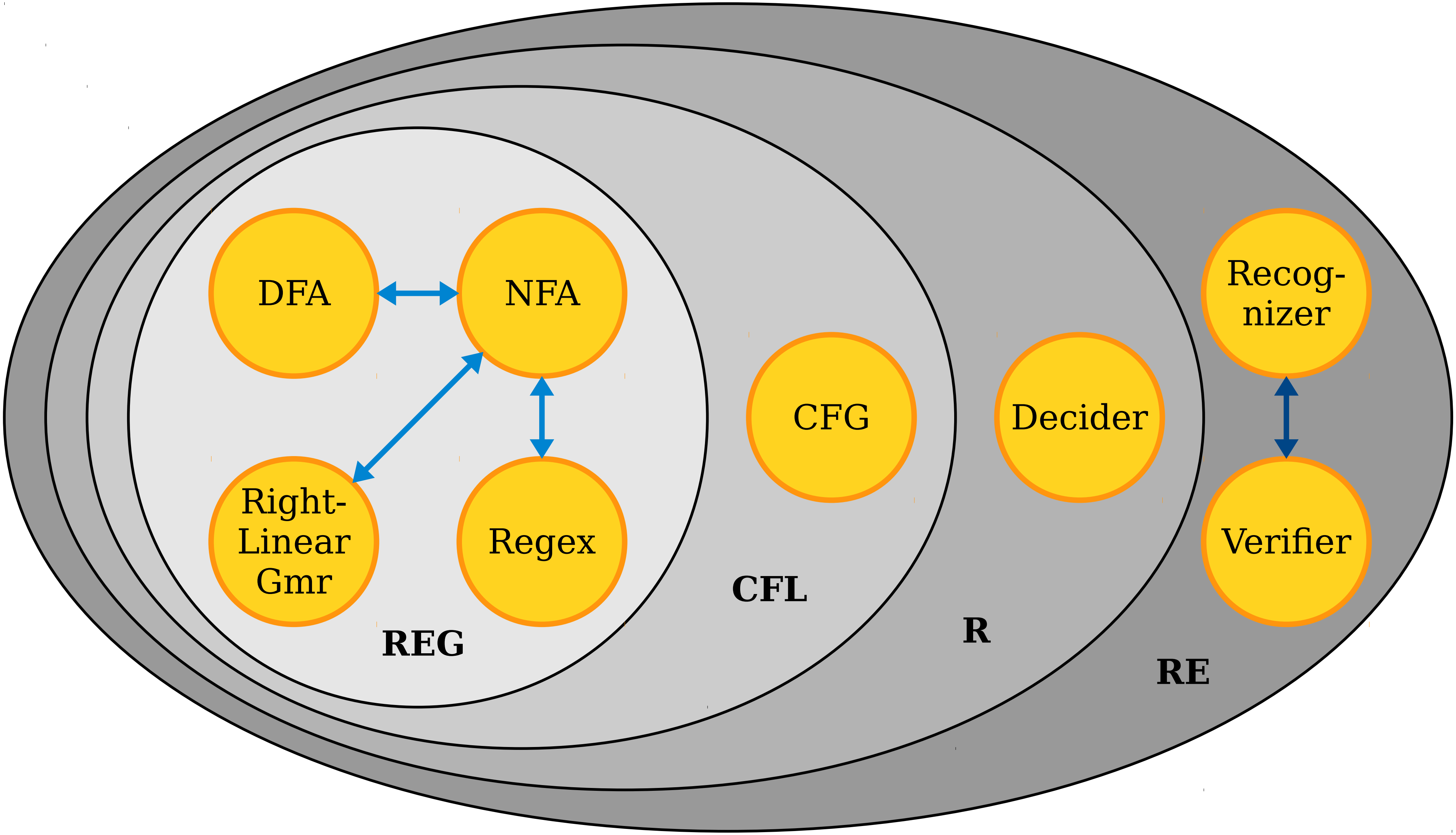
There are inherent limits about what mathematics can teach us.

- There's no automatic way to do math. There are true statements that we can't prove.
- That doesn't mean that mathematics is worthless. It just means that we need to temper our expectations about it.

Where We Stand

- We've just done a crazy, whirlwind tour of computability theory:
  - ***The Church-Turing thesis*** tells us that TMs give us a mechanism for studying computation in the abstract.
  - ***Universal computers*** – computers as we know them – are not just a stroke of luck. The existence of the universal TM ensures that such computers must exist.
  - ***Self-reference*** is an inherent consequence of computational power.
  - ***Undecidable problems*** exist partially as a consequence of the above and indicate that there are statements whose truth can't be determined by computational processes.
  - ***Unrecognizable problems*** are out there and can be discovered via diagonalization. They imply there are limits to mathematical proof.

The Big Picture



Where We've Been

- The class **R** represents problems that can be solved by a computer.
- The class **RE** represents problems where “yes” answers can be verified by a computer.

Where We're Going

- The class **P** represents problems that can be solved *efficiently* by a computer.
- The class **NP** represents problems where “yes” answers can be verified *efficiently* by a computer.

Next Time

- ***Introduction to Complexity Theory***
  - Not all decidable problems are created equal!
- ***The Classes P and NP***
  - Two fundamental and important complexity classes.
- ***The P $\overset{?}{=}$ NP Question***
  - A literal million-dollar question!