Announcements

- Wednesday OH canceled
- Today’s section split into class/OH
- We’re almost there! You got this!
- The final is Saturday. **Start studying now!**
    - Will cover all topics
    - Practice exams are up now
- Review Session Thursday 1-3pm (location TBA, but likely LATHROP 298)

Verifiers

A TM V is a **verifier** for a language L if:

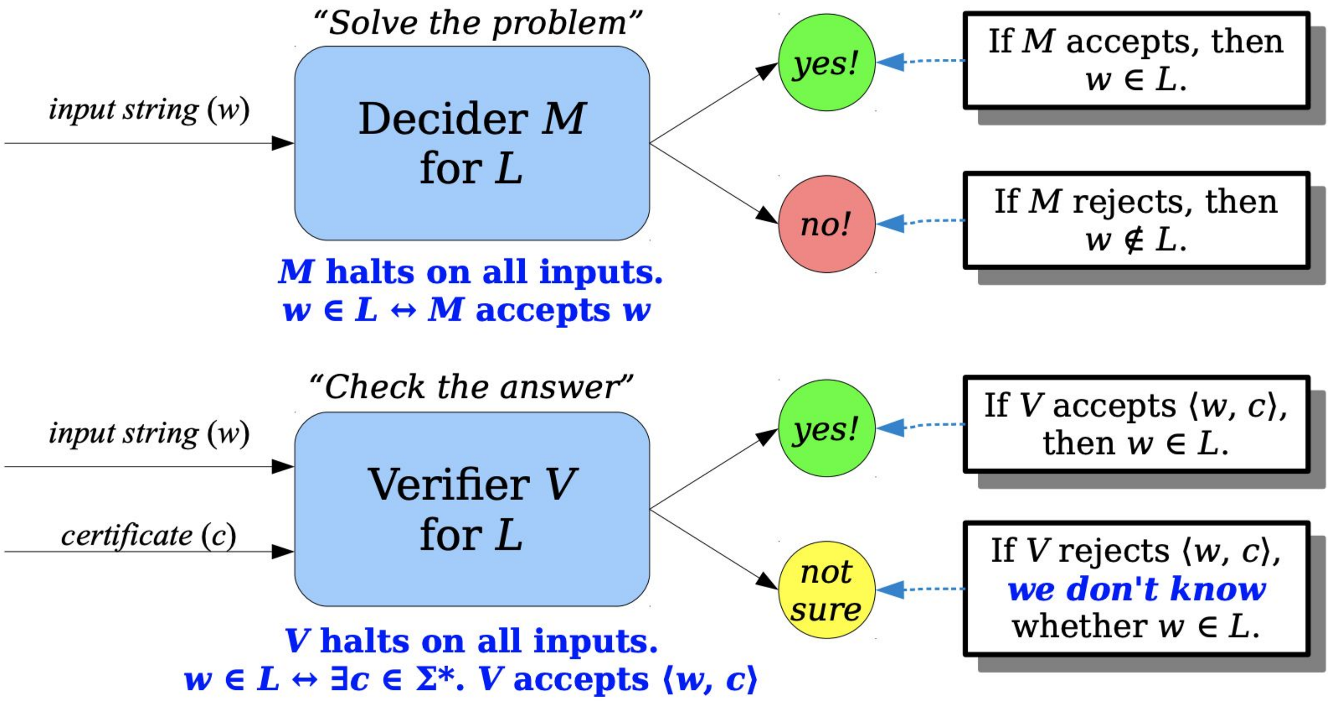
$$\textbf{V halts on all inputs, and}$$
$$\forall w \in \Sigma^*. (w \in L \leftrightarrow \exists c \in \Sigma^*. V \text{ accepts } \langle w, c\rangle)$$

Here is an equivalent definition:

$$\forall w \in \Sigma^*. (w \in L \rightarrow \exists c \in \Sigma^*. V \text{ accepts } \langle w, c\rangle), \text{ and}$$
$$\forall w \in \Sigma^*. (w \notin L \rightarrow \forall c \in \Sigma^*. V \text{ rejects } \langle w, c\rangle)$$

How do we prove or argue something is a verifier?
→ **it's a universal statement!**

Verifiers vs Deciders



Self-Reference

Proving that a language is not decidable by contradiction

General template:

- Assume for the sake of contradiction that L is decidable
- That means there is a decider for L
- Construct a function using the decider that makes sure the decider gives the wrong answer about it

When do we use Self-Reference

As a refresher, the language A_{TM} is

A_{TM} = { ⟨*M*, *w*⟩ | *M* is a TM and *M* accepts *w* }.

Given a TM *M* and a string *w*, a decider *D* for A_{TM} would need to have this behavior:

- If *M* accepts *w*, then *D* accepts ⟨*M*, *w*⟩.
- If *M* rejects *w*, then *D* rejects ⟨*M*, *w*⟩.
- If *M* loops on *w*, then *D* rejects ⟨*M*, *w*⟩.

This is basically the same set of requirements as U_{TM}, except for what happens if *M* loops on *w*.

Our goal is to prove that there is no way to build a program that meets these requirements.

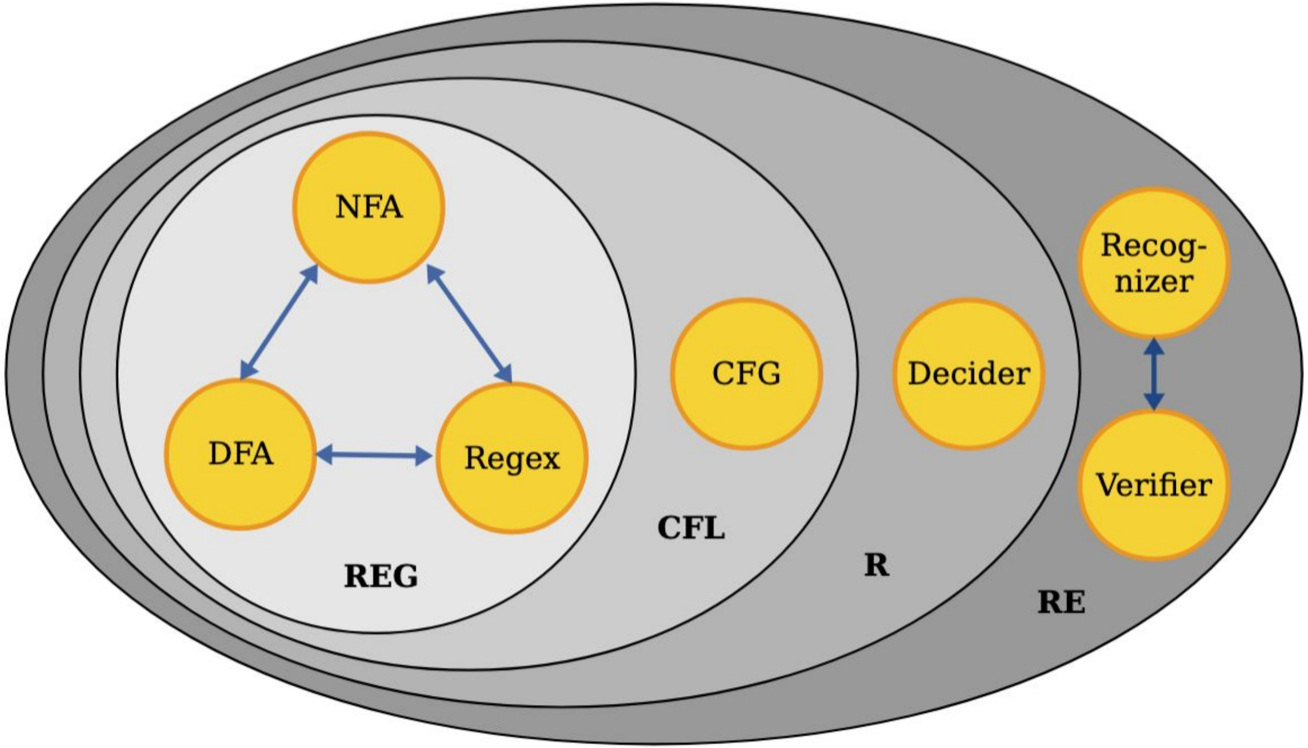
The Lava Diagram

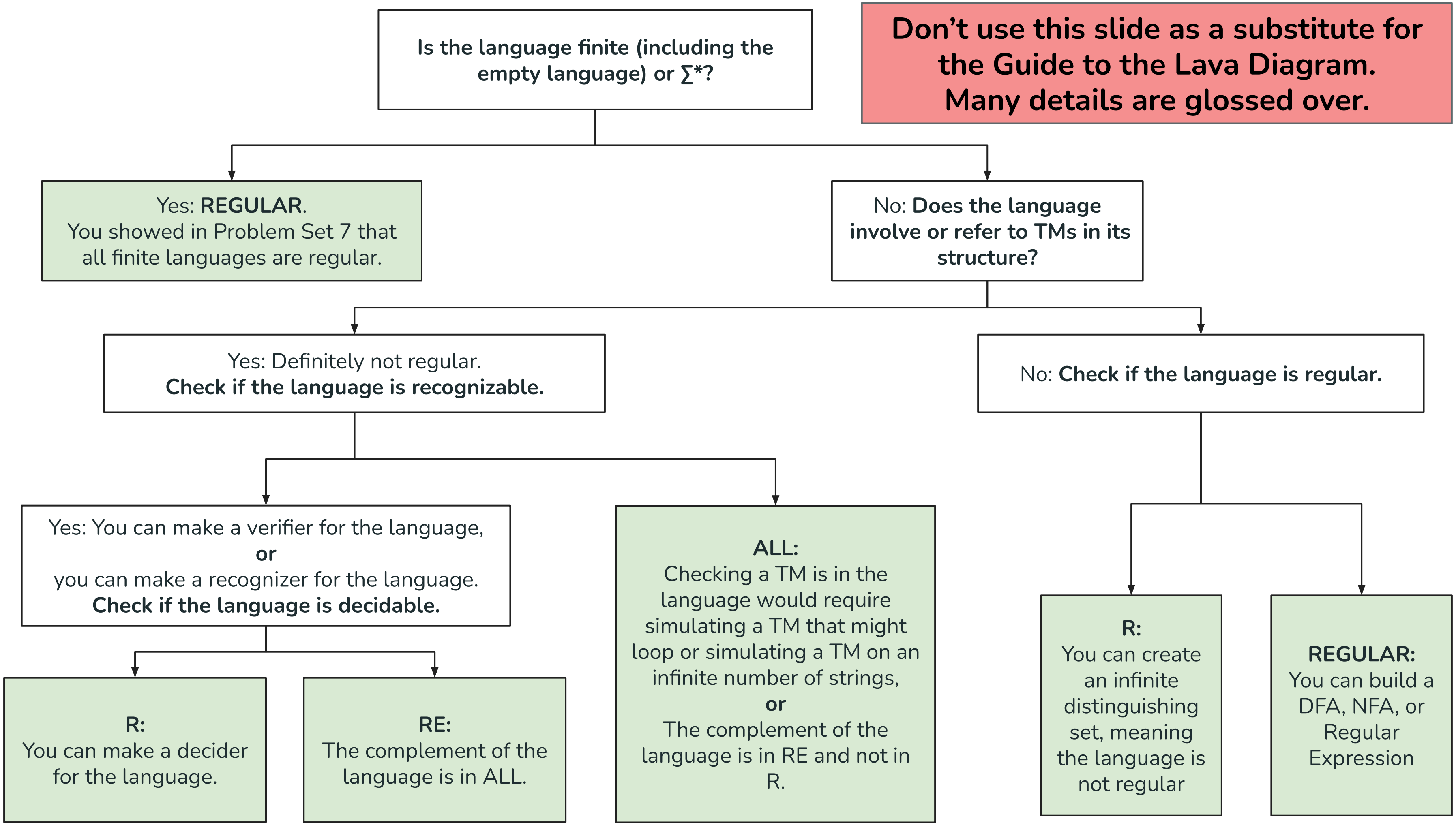
- **REG**, the regular languages (languages with a DFA, NFA, or regex) are a strict subset of
- **R**, the decidable languages (languages with a decider), which are a strict subset of
- **RE**, the recognizable languages (languages with a recognizer), which are a strict subset of
- **All languages**

The Lava Diagram Problems

Check out the properties of each of these computing devices!

Whichever is the **least** complex that can still represent that language will be its category.




Don't use this slide as a substitute for the Guide to the Lava Diagram. Many details are glossed over.
Is the language finite (including the empty language) or ∑*?
Yes: REGULAR. You showed in Problem Set 7 that all finite languages are regular.
No: Does the language involve or refer to TMs in its structure?
Yes: Definitely not regular. Check if the language is recognizable.
No: Check if the language is regular.
Yes: You can make a verifier for the language, or you can make a recognizer for the language. Check if the language is decidable.
ALL: Checking a TM is in the language would require simulating a TM that might loop or simulating a TM on an infinite number of strings, or The complement of the language is in RE and not in R.
R: You can create an infinite distinguishing set, meaning the language is not regular
REGULAR: You can build a DFA, NFA, or Regular Expression
R: You can make a decider for the language.
RE: The complement of the language is in ALL.

CS103ACE: Lava Diagram "Cheat Sheet"

	A language L is in this category if any of these are true	A language L is NOT in this category if any of these are true
RE	• There is a verifier for L – intuitively, given any string in L, we can provide proof that it's in L • There is a recognizer for L – intuitively, given any string in L, we can certainly say that it is in L in a finite amount of time	• To determine that a string in L is actually in L, we have to check a TM's result on infinitely many strings, or we have to check that a TM loops • L has no verifier – Intuitively, even if you know a string is in the language, you cannot convince anyone of this "Canonical" unrecognizable language: L_D
R	• L is in **RE** *and* $\overline{L}$ is in **RE** • There is a decider for L – intuitively, given any string at all, we can certainly say if it is in L or not in a finite amount of time	• $\overline{L}$ is not in **RE** • A decider for L is impossible to build since we can trick it using self-reference (usually comes up when L's input is a TM) • L is a "regulatory problem" about computer programs (usually) – most, but not all, problems of the form "does program X have [behavior Y]" are undecidable "Canonical" recognizable but undecidable languages: A_{TM}, $HALT$
REG	• There is a DFA, NFA, or regex for L • $\overline{L}$ is regular • We only need to remember a finite amount of information to solve L • L contains a finite number of strings (note: there are also infinite regular languages)	• L has an infinite distinguishing set (via Myhill-Nerode) • Intuitively, we can't solve L while only keeping track of a finite amount of information/states "Canonical" decidable but nonregular languages: $\{a^n b^n \mid n \in \mathbb{N}\}$, palindromes

What about CFGs, P, and NP? **REG** is a strict subset of the CFLs, which are a strict subset of **P**, which is a strict subset of **R**. (**NP** is also a strict subset of **R**, but you don't need to know this.) ("A is a strict subset of B" means $A \subseteq B$ and $A \neq B$.)

Non-RE Languages are unsolvable

The ***diagonalization language***, which we denote L_D, is defined as

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

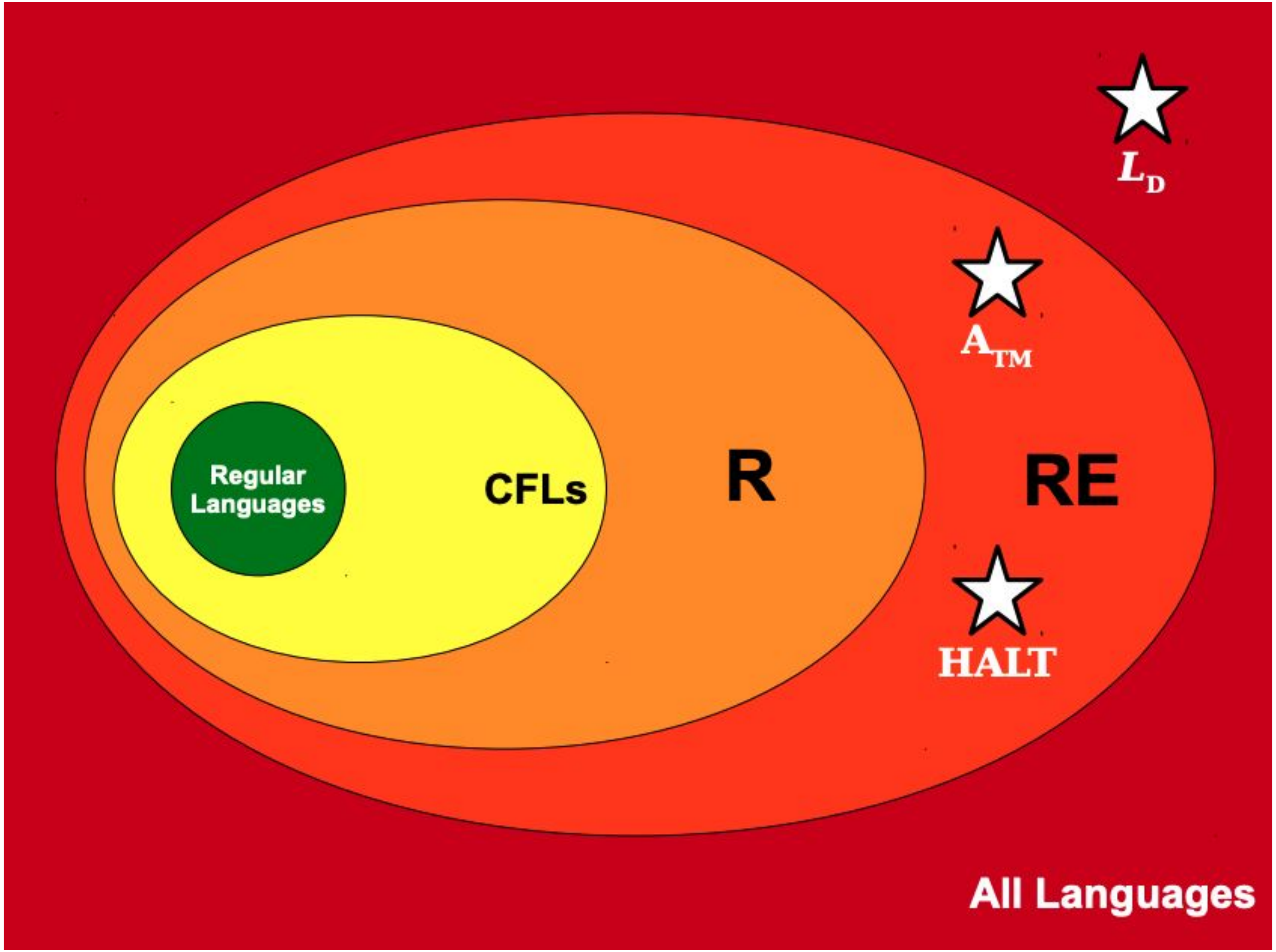
That is, L_D is the set of descriptions of Turing machines that do not accept themselves.

If you did build a TM for L_D, called M_{LD}, would M_{LD} accept $\langle M_{LD} \rangle$?

This is a paradox! If M is not in L(M), it must be in L_D, but L_D=L(M), so this is impossible!

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



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

Some things to remember from CS 103

- Mathematics
    - How mathematicians argue for truth using proofs
    - Using highly abstract concepts and formal definitions (sets, functions, graphs) to model important and cool things
    - Math is way more than just solving equations!
    - **You** can do interesting and rigorous math!
- Computability
    - Problems that can't be solved by computers
    - The P vs. NP problem

Some things to remember from CS 103

- If you enjoyed parts of this class...
  - CS is a great place for you!
  - If you like computation theory, try CS 154
  - If you liked finite automata, try designing more in EE108
- If you did **NOT** enjoy this class
  - **CS is a great place for you!**
  - This is just one course in an extremely broad area
- 161 is a cool combination of 106B and 103