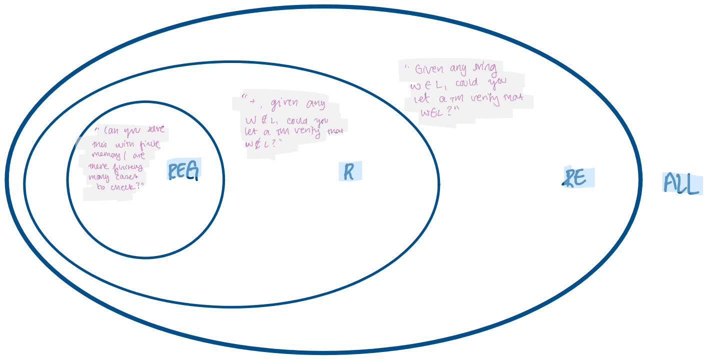
Guide to the Lava Diagram

Friday, August 14, 2020 8:00 AM



- Generally, when asked to place a language in ^, work your way inwards from (ALL to REG)

RE (Recognizable)

- = languages w/ verifiers
- "If I knew that any given string was in L, could I prove it"?
- Try thinking of a possible verifier — remember, not allowed to loop!
  - $L_1 = \{ \langle M \rangle \mid M$ is a TM & $|\mathcal{L}(M)| \geq 2 \}$ = every TM that accepts ≥2 strings. A possible verifiers for a given M would be any two strings it accepts (verify correct) and a step count limit (to make sure it doesn't loop trying to prove correct) → $L_1 \in RE$
  - $L_2 = \{ \langle M \rangle \mid M$ is a TM & $|\mathcal{L}(M)| = 2 \}$ = even if you knew a TM accepted two specific strings, to prove that it doesn't accept any others would require running it on infinitely more strings. It would either reject or loop, but never accept → $L_2 \notin RE$

any language that asks for a minimum requirement is usually verifiable

any language that has a specific maximum requirement would need checking an infinite input set, probably not verifiable.

R (Decidable)

- = languages w/ deciders → (you could take two verifiers for L & $\overline{L}$ to build a decider for L)
- = languages that are in RE & their complement in RE! / "If I knew a string not in L wasn't in L, could I ~~prove~~ prove it wasn't in L?"
  - L_1 from above — it's hard to prove, of the infinite strings for each TM, that only less than two would accept. You can't run TM on infinite strings & hope for an answer that can only be reached after running the 'last' string, it would loop or reject, & never accept → $\overline{L_1} \notin RE$ → $L_1 \notin R$

to actually a language is undecidable, you should still use the self-referential trick.

REG (Regular)

- = languages w/ DFAs / NFAs / Regexes.
- = languages that can be solved with finite memory – are there a finite number of things to remember about input? Could you solve it without an infinite tape?
  - ↳ the Myhill-Nerode theorem relies on this principle, & proves when languages really need infinite memory. You could also just use it for this intuition
  - $L_3 = \{ a^n b^n \mid n \in \mathbb{N}$ & $n \geq 1000 \}$ - really easy to prove if any string is in / not in L_3 with a verifier. But seeing as n can be infinitely large, and that to check a string input, you would have to remember how many a's are in it & that number could be infinite, $L_3 \notin REG$
  - $L_4 = \{ a^n b^n \mid n \in \mathbb{N}$ & $n < 1000 \}$ - again, easily verifiable. But also
    - ↳ to read any input you only need to remember if it had up to 1001 a's ~~&~~ b's (if it has more, automatically reject)
    - ↳ there are a finite number of strings in this lang, &
    - → you could think of a DFA/NFA/Regex design for this. It could be ~~a~~ super long & ugly regex (ab ∪ aabb ∪ ...), or a DFA like that too.

as proven in PSET 5, every finite language is regular.

Practice Final #4, Problem 5.iii

3. Σ = all alphanumeric chars,
   $\{ w \in \Sigma^* \mid w$ contains at least 2 lower-case, ≥2 upper-case, ≥ digit, doesn't end w/ a digit, & $|w| \geq 8 \}$
   - is verifiable ✓ complement isn't verifiable ✗ (could search through an infinite string looking for minimum requirements that aren't there) → RE

   apparently it's regular... w can't be infinite length, it could just be any large finite length

   don't use the can't-search-infinity reasoning when input is a finite string from an infinite set!

4. $\{ \langle P \rangle \mid P$ is a syntactly correct Java program $\}$
   - needs to have matching braces, semicolons, etc.
   - I can already imagine using the self-referential trick on this
   - verifiable, b/c could just point out index of each reg. char, but complement isn't (same reason)
   - → RE ✗ R
     (not regular though)

5. $\{ \langle U_{TM} \rangle \}$ ✗ "this language is verifiable b/c it's finite" is all the explanation given. The language has a single element
   - not verifiable – if I had a string & had to prove it was $\langle U_{TM} \rangle$, I would have to check every ⟨M, w⟩ pair on U_{TM}, which is infinite → ∉ RE
     ✗ check for if input == ⟨M, w⟩ is different from input = U_{TM}.

6. $\mathcal{L}(U_{TM}) = A_{TM}$, $A_{TM} \in RE$ ✓
   - no way to prove that w not accepted by M definitely (could be looping), so not in R.

7. $\overline{\mathcal{L}(U_{TM})} = \overline{A_{TM}}$, ^ → ∉ RE ✓, would require proving that M doesn't accept w, could just be taking a long time to accept.

8. $\{ \langle M, w \rangle$, U_{TM} rejects $\langle M, w \rangle \}$ = M rejects w
   - verifiable, give the step count until rejecting step as certificate ✓
   - complement not verifiable b/c if it loops, you can't prove it's looping & not just taking long time to reject → ∈ RE ✓
     (specify it's not in R)