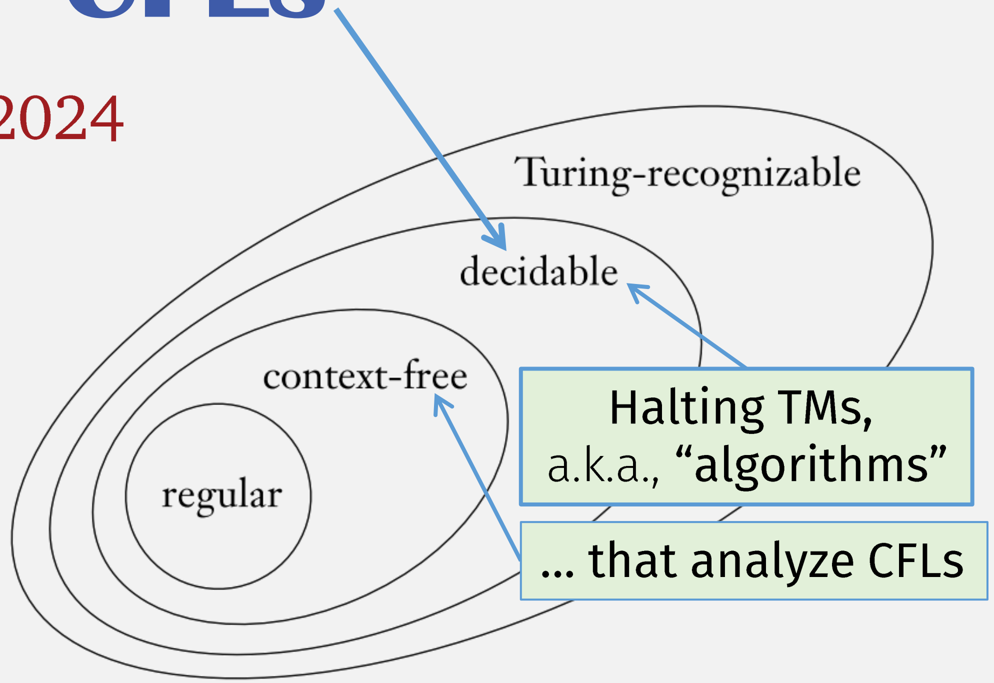
UMB CS 420

Decidability for CFLs

Wednesday, April 17, 2024



Announcements

- HW 8 in
  - ~~Due Wed April 17 12pm noon~~
- HW 9 out
  - Due Wed April 24 12pm noon

<u>4/17 Lecture Participation Question (in GradeScope)</u>

- Which of the following rules are valid for a grammar in **Chomsky Normal Form**?

Last Time: Decider Turing Machines

- 2 classes of Turing Machines
  - **Recognizers** (all TMs): may loop forever
    - TM that **loops** on an input does **not accept** that input
  - **Deciders** (subset of TMs) (**algorithms**) always halt
    - Must **accept** or **reject**

- **Decider** definitions must include a **termination argument:**
  - Explains (informally) why every step in the TM halts
  - (Pay special attention to loops)

Last Time: Algorithms About Regular Langs

- $A_{\text{DFA}} = \{\langle B, w\rangle \mid B \text{ is a DFA that accepts input string } w\}$
  - **Decider**: Simulates DFA by implementing extended δ function
- $A_{\text{NFA}} = \{\langle B, w\rangle \mid B \text{ is an NFA that accepts input string } w\}$
  - **Decider**: Uses **NFA→DFA** decider + A_{DFA} decider
- $A_{\text{REX}} = \{\langle R, w\rangle \mid R \text{ is a regular expression that generates string } w\}$
  - **Decider**: Uses **RegExpr→NFA** decider + A_{NFA} decider
- $E_{\text{DFA}} = \{\langle A\rangle \mid A \text{ is a DFA and } L(A) = \emptyset\}$
  - **Decider**: Reachability algorithm

Lang of the DFA

- $EQ_{\text{DFA}} = \{\langle A, B\rangle \mid A \text{ and } B \text{ are DFAs and } L(A) = L(B)\}$

  - **Decider**: Uses complement and intersection closure construction + E_{DFA} decider

<u>Remember</u>:
TMs ~ programs
Creating TM ~ programming
Previous theorems ~ <u>library</u>

Next: Algorithms (Decider TMs) for CFLs?

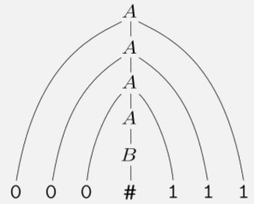
- What can we predict about CFGs or PDAs?

Thm: A_{CFG} is a decidable language

$A_{\text{CFG}} = \{\langle G, w\rangle \mid G \text{ is a CFG that generates string } w\}$

- This is a very practically important problem …
- … equivalent to:
  - **Algorithm** to **parse** "program" w for a programming language with grammar G?

- A Decider for this problem could … ?
  - Try every possible derivation of G, and check if it's equal to w?
  - But this might never halt
    - E.g., what if there are rules like: $\mathbf{S} \rightarrow \mathbf{0S}$ or $\mathbf{S} \rightarrow \mathbf{S}$
  - This TM would be a recognizer but not a decider

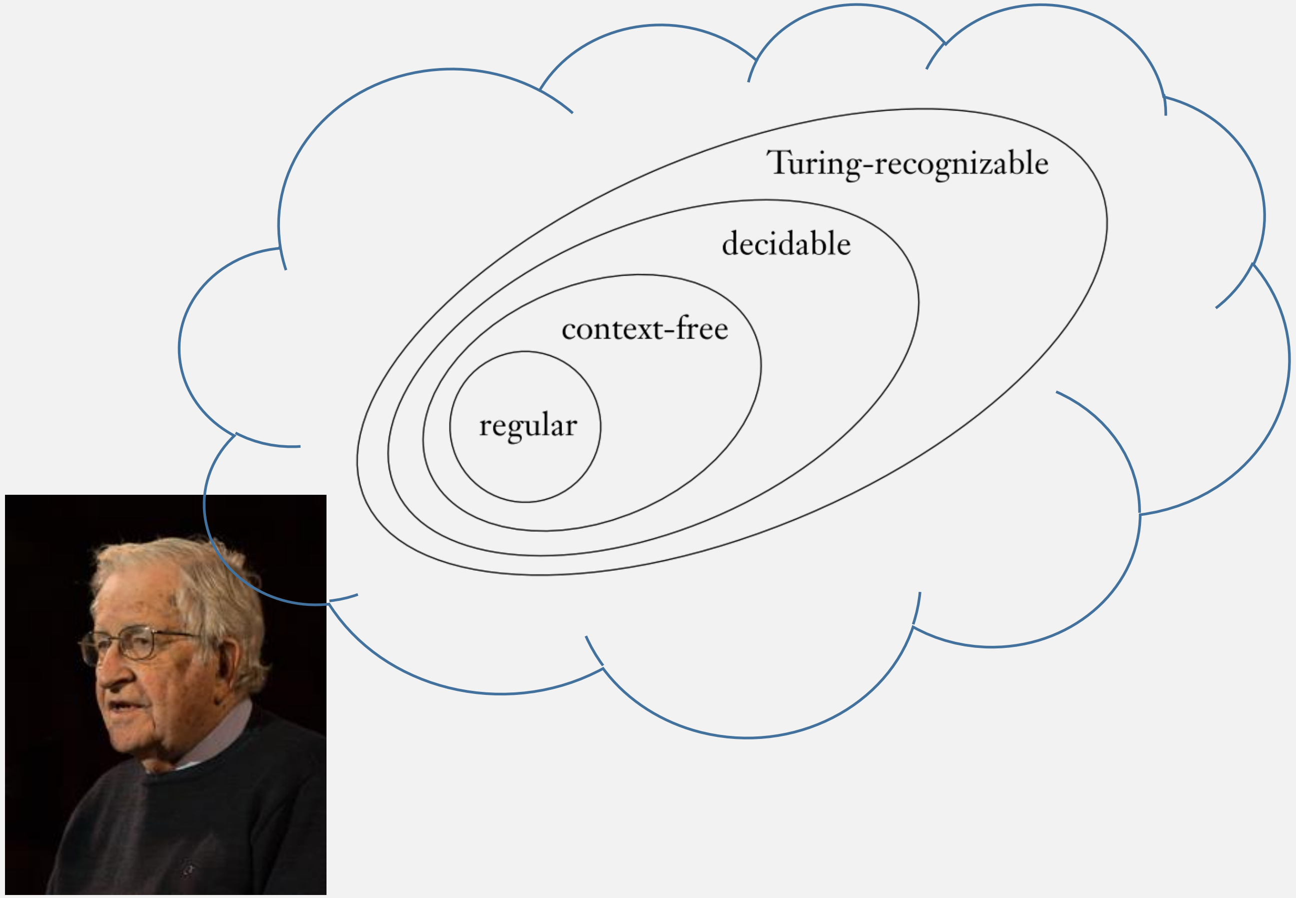


Idea: can the TM stop checking after some length?

- I.e., Is there upper bound on the number of derivation steps?

Chomsky Normal Form

Noam Chomsky



He came up with this <u>hierarchy</u> of languages

Chomsky Normal Form

A context-free grammar is in ***Chomsky normal form*** if every rule is of the form

$$A \to BC$$
$$A \to a$$

(non-start) Variables only

2 rule shapes

Terminals only

where a is any terminal and A, B, and C are any variables—except that B and C may not be the start variable. In addition, we permit the rule $S \to \varepsilon$, where S is the start variable.

Chomsky Normal Form Example

Makes the string long enough

Convert variables to terminals

- $S \rightarrow AB$
- $B \rightarrow AB$
- $A \rightarrow$ a
- $B \rightarrow$ b

- To generate string of length: 2
  - Use S rule: 1 time; Use A or B rules: 2 times
  - $S \Rightarrow AB \Rightarrow \text{a}B \Rightarrow \text{ab}$
  - Derivation total steps: 1 + 2 = 3
- To generate string of length: 3
  - Use S rule: 1 time; A rule: 1 time; A or B rules: 3 times
  - $S \Rightarrow AB \Rightarrow AAB \Rightarrow \text{a}AB \Rightarrow \text{aa}B \Rightarrow \text{aab}$
  - Derivation total steps: 1 + 1 + 3 = 5
- To generate string of length: 4
  - Use S rule: 1 time ; A rule: 2 times; A or B rules: 4 times
  - $S \Rightarrow AB \Rightarrow AAB \Rightarrow AAAB \Rightarrow \text{a}AAB \Rightarrow \text{aa}AB \Rightarrow \text{aaa}B \Rightarrow \text{aaab}$
  - Derivation total steps: 3 + 4 = 7
- …

A context-free grammar is in ***Chomsky normal form*** if every rule is of the form

$$A \rightarrow BC$$
$$A \rightarrow a$$

2 rule shapes

where a is any terminal and A, B, and C are any variables—except that B and C may not be the start variable. In addition, we permit the rule $S \rightarrow \varepsilon$, where S is the start variable.

Chomsky Normal Form: Number of Steps

To generate a string of length n:

$n - 1$ steps: to generate n variables — Makes the string long enough

$+ n$ steps: to turn each variable into a terminal — Convert string to terminals

Total: $2n - 1$ steps

(A *finite* number of steps!)

Chomsky normal form

$A \rightarrow BC$ — Use n-1 times

$A \rightarrow a$ — Use n times

Thm: A_{CFG} is a decidable language

$A_{\mathsf{CFG}} = \{\langle G, w\rangle \mid G \text{ is a CFG that generates string } w\}$

Proof: create the decider:

$S =$ "On input $\langle G, w\rangle$, where G is a CFG and w is a string:

1. Convert G to an equivalent grammar in Chomsky normal form.
2. List all derivations with $2n-1$ steps, where n is the length of w; except if $n = 0$, then instead list all derivations with one step.
3. If any of these derivations generate w, *accept*; if not, *reject*."

We first need to prove this is true for all CFGs!

Step 1: Conversion to Chomsky Normal Form is an algorithm …
Step 2:
Step 3:

Termination argument?

Thm: Every CFG has a Chomsky Normal Form

Proof: Create algorithm to convert any CFG into Chomsky Normal Form

Chomsky normal form

$A \to BC$

$A \to a$

1. Add new start variable S_0 that does not appear on any RHS
   - I.e., add rule $S_0 \to S$, where S is old start var

$$S \to ASA \mid \mathtt{a}B$$
$$A \to B \mid S$$
$$B \to \mathtt{b} \mid \varepsilon$$

⇒

$$\boldsymbol{S_0 \to S}$$
$$S \to ASA \mid \mathtt{a}B$$
$$A \to B \mid S$$
$$B \to \mathtt{b} \mid \varepsilon$$

Thm: Every CFG has a Chomsky Normal Form

Chomsky normal form

$A \to BC$
$A \to a$

1. Add new start variable S_0 that does not appear on any RHS
   - I.e., add rule $S_0 \to S$, where S is old start var
2. Remove all "empty" rules of the form $A \to \varepsilon$
   - A must not be the start variable
   - Then for every rule with A on RHS, add new rule with A deleted
     - E.g., If $R \to uAv$ is a rule, add $R \to uv$
   - Must cover all combinations if A appears more than once in a RHS
     - E.g., if $R \to uAvAw$ is a rule, add 3 rules: $R \to uvAw$, $R \to uAvw$, $R \to uvw$

$$\begin{aligned} S_0 &\to S \\ S &\to ASA \mid \mathtt{a}B \mid \mathbf{a} \\ A &\to B \mid S \mid \boldsymbol{\varepsilon} \\ B &\to \mathtt{b} \mid \varepsilon \end{aligned}$$

Then, add

First, remove

$$\begin{aligned} S_0 &\to S \\ S &\to ASA \mid \mathtt{a}B \mid \mathtt{a} \mid \boldsymbol{SA} \mid \boldsymbol{AS} \mid \boldsymbol{S} \\ A &\to B \mid S \mid \varepsilon \\ B &\to \mathtt{b} \end{aligned}$$

Then add, to account for possibly empty A

Then, remove

Thm: Every CFG has a Chomsky Normal Form

Chomsky normal form

$A \to BC$

$A \to a$

1. Add new start variable S_0 that does not appear on any RHS
   - I.e., add rule $S_0 \to S$, where S is old start var
2. Remove all "empty" rules of the form $A \to \varepsilon$
   - A must not be the start variable
   - Then for every rule with A on RHS, add new rule with A deleted
     - E.g., If $R \to uAv$ is a rule, add $R \to uv$
   - Must cover all combinations if A appears more than once in a RHS
     - E.g., if $R \to uAvAw$ is a rule, add 3 rules: $R \to uvAw$, $R \to uAvw$, $R \to uvw$
3. Remove all "unit" rules of the form $A \to B$
   - Then, for every rule $B \to u$, add rule $A \to u$

$$
\begin{aligned}
S_0 &\to S \\
S &\to ASA \mid aB \mid a \mid SA \mid AS \mid S \\
A &\to B \mid S \\
B &\to b
\end{aligned}
$$

Remove, no add (same variable)

$$
\begin{aligned}
S_0 &\to S \mid \mathbf{ASA} \mid \mathbf{aB} \mid \mathbf{a} \mid \mathbf{SA} \mid \mathbf{AS} \\
S &\to ASA \mid aB \mid a \mid SA \mid AS \\
A &\to B \mid S \\
B &\to b
\end{aligned}
$$

Remove, then add S RHSs to S_0

$$
\begin{aligned}
S_0 &\to ASA \mid aB \mid a \mid SA \mid AS \\
S &\to ASA \mid aB \mid a \mid SA \mid AS \\
A &\to S \mid b \mid \mathbf{ASA} \mid \mathbf{aB} \mid \mathbf{a} \mid \mathbf{SA} \mid \mathbf{AS} \\
B &\to b
\end{aligned}
$$

Remove, then add S RHSs to A

Termination argument of this algorithm?

Thm: Every CFG has a Chomsky Normal Form

Chomsky normal form

$$\begin{aligned} A &\rightarrow BC \\ A &\rightarrow a \end{aligned}$$

1. Add new start variable S_0 that does not appear on any RHS
   - I.e., add rule $S_0 \rightarrow S$, where S is old start var
2. Remove all "empty" rules of the form $A \rightarrow \varepsilon$
   - A must not be the start variable
   - Then for every rule with A on RHS, add new rule with A deleted
     - E.g., If $R \rightarrow uAv$ is a rule, add $R \rightarrow uv$
   - Must cover all combinations if A appears more than once in a RHS
     - E.g., if $R \rightarrow uAvAw$ is a rule, add 3 rules: $R \rightarrow uvAw$, $R \rightarrow uAvw$, $R \rightarrow uvw$
3. Remove all "unit" rules of the form $A \rightarrow B$
   - Then, for every rule $B \rightarrow u$, add rule $A \rightarrow u$
4. Split up rules with RHS longer than length 2
   - E.g., $A \rightarrow wxyz$ becomes $A \rightarrow wB$, $B \rightarrow xC$, $C \rightarrow yz$
5. Replace all terminals on RHS with new rule
   - E.g., for above, add $W \rightarrow w$, $X \rightarrow x$, $Y \rightarrow y$, $Z \rightarrow z$

$$\begin{aligned} S_0 &\rightarrow ASA \mid \mathtt{a}B \mid \mathtt{a} \mid SA \mid AS \\ S &\rightarrow ASA \mid \mathtt{a}B \mid \mathtt{a} \mid SA \mid AS \\ A &\rightarrow \mathtt{b} \mid ASA \mid \mathtt{a}B \mid \mathtt{a} \mid SA \mid AS \\ B &\rightarrow \mathtt{b} \end{aligned}$$

$$\begin{aligned} S_0 &\rightarrow AA_1 \mid UB \mid \mathtt{a} \mid SA \mid AS \\ S &\rightarrow AA_1 \mid UB \mid \mathtt{a} \mid SA \mid AS \\ A &\rightarrow \mathtt{b} \mid AA_1 \mid UB \mid \mathtt{a} \mid SA \mid AS \\ A_1 &\rightarrow SA \\ U &\rightarrow \mathtt{a} \\ B &\rightarrow \mathtt{b} \end{aligned}$$

Thm: A_{CFG} is a decidable language

$$A_{\text{CFG}} = \{\langle G, w\rangle \mid G \text{ is a CFG that generates string } w\}$$

Proof: create the decider:

$S =$ "On input $\langle G, w\rangle$, where G is a CFG and w is a string:

1. Convert G to an equivalent grammar in Chomsky normal form.
2. List all derivations with $2n - 1$ steps, where n is the length of w; except if $n = 0$, then instead list all derivations with one step.
3. If any of these derivations generate w, *accept*; if not, *reject*."

We first need to prove this is true for all CFGs! ☑

Termination argument:
Step 1: any CFG has only a finite # rules
Step 2: $2n$-1 = finite # of derivations to check
Step 3: checking finite number of derivations

Thm: E_{CFG} is a decidable language

$E_{\text{CFG}} = \{\langle G\rangle \mid G \text{ is a CFG and } L(G) = \emptyset\}$

Recall:

$E_{\text{DFA}} = \{\langle A\rangle \mid A \text{ is a DFA and } L(A) = \emptyset\}$

T = "On input $\langle A\rangle$, where A is a DFA:

1. Mark the start state of A.
2. Repeat until no new states get marked:
3. Mark any state that has a transition coming into it from any state that is already marked.
4. If no accept state is marked, *accept*; otherwise, *reject*."

"Reachability" (of accept state from start state) algorithm

Can we compute "reachability" for a CFG?

Thm: E_{CFG} is a decidable language

$E_{\mathsf{CFG}} = \{\langle G \rangle | \ G \text{ is a CFG and } L(G) = \emptyset\}$

Proof: create **decider** that calculates reachability for grammar G

- Go backwards, start from **terminals**, to avoid getting stuck in looping rules

$R =$ "On input $\langle G \rangle$, where G is a CFG:

1. Mark all terminal symbols in G.
2. Repeat until no new variables get marked:
3. Mark any variable A where G has a rule $A \rightarrow U_1 U_2 \cdots U_k$ and each symbol $U_1, \ldots, U_k$ has already been marked.
4. If the start variable is not marked, *accept*; otherwise, *reject*."

Loop marks 1 new variable on each iteration or stops: it eventually terminates because there are a finite # of variables

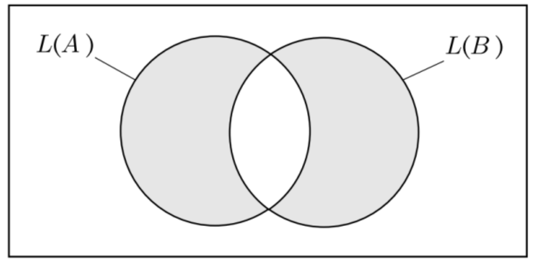
Termination argument?

<u>Thm</u>: EQ_{CFG} is a decidable language?

$EQ_{\mathsf{CFG}} = \{\langle G, H\rangle \mid G \text{ and } H \text{ are CFGs and } L(G) = L(H)\}$

<u>Recall</u>: $EQ_{\mathsf{DFA}} = \{\langle A, B\rangle \mid A \text{ and } B \text{ are DFAs and } L(A) = L(B)\}$

- Used Symmetric Difference



$L(C) = \emptyset \text{ iff } L(A) = L(B)$

  - where C = complement, union, intersection of machines A and B
- Can't do this for CFLs!
  - Intersection and complement are <u>not closed</u> for CFLs!!!

Intersection of CFLs is Not Closed!

Proof (by contradiction), Assume **intersection is closed for CFLs**

- Then **intersection of these CFLs should be a CFL:**

$$A = \{\mathtt{a}^m \mathtt{b}^n \mathtt{c}^n \mid m, n \geq 0\}$$

$$B = \{\mathtt{a}^n \mathtt{b}^n \mathtt{c}^m \mid m, n \geq 0\}$$

- But $A \cap B = \{\mathtt{a}^n \mathtt{b}^n \mathtt{c}^n \mid n \geq 0\}$

- … **which is not a CFL!** (So we have a contradiction)

Complement of a CFL is not Closed!

- Assume CFLs closed under complement, then:

$$\text{if } G_1 \text{ and } G_2 \text{ context-free}$$

$$\overline{L(G_1)} \text{ and } \overline{L(G_2)} \text{ context-free}$$ From the assumption

$$\overline{L(G_1)} \cup \overline{L(G_2)} \text{ context-free}$$ Union of CFLs is closed

$$\overline{\overline{L(G_1)} \cup \overline{L(G_2)}} \text{ context-free}$$ From the assumption

$$L(G_1) \cap L(G_2) \text{ context-free}$$ DeMorgan's Law!

But intersection is not closed for CFLS (prev slide)

Thm: EQ_{CFG} is a decidable language?

$EQ_{\text{CFG}} = \{\langle G, H \rangle \mid G \text{ and } H \text{ are CFGs and } L(G) = L(H)\}$

- No! 
  - There's no algorithm to decide whether two grammars are equivalent!

- **It's not recognizable either!** (Can't create any TM to do this!!!)
  - (details later)

- I.e., **this is an impossible computation!**

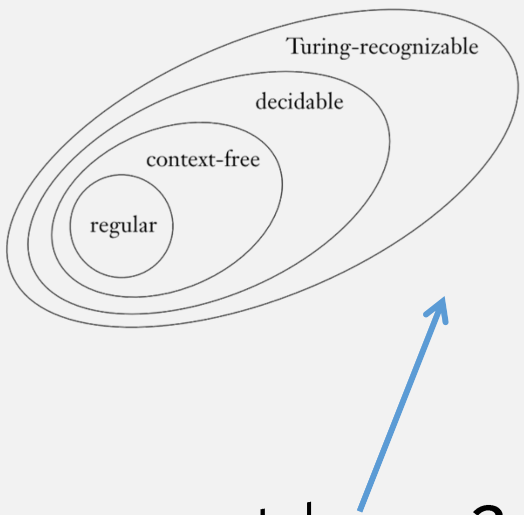
Summary Algorithms About CFLs

- $A_{\text{CFG}} = \{\langle G, w \rangle |\ G \text{ is a CFG that generates string } w\}$
  - **Decider:** Convert grammar to Chomsky Normal Form
  - Then **check all possible derivations up to length 2|w| - 1 steps**

- $E_{\text{CFG}} = \{\langle G \rangle |\ G \text{ is a CFG and } L(G) = \emptyset\}$
  - **Decider:** Compute "reachability" of start variable from terminals

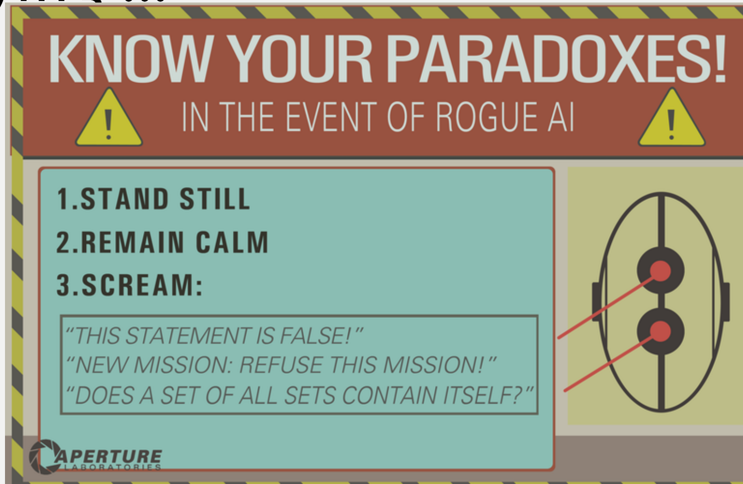
- $EQ_{\text{CFG}} = \{\langle G, H \rangle |\ G \text{ and } H \text{ are CFGs and } L(G) = L(H)\}$
  - **We couldn't prove that this is decidable!**
  - (So you cant use this theorem when creating another decider)

The Limits of Turing Machines?



- **TMs represent all possible "computations"**
  - I.e., **any (Python, Java, …) program you write is a TM**
- But **some things are not computable?** I.e., some langs are out here **?**
- **To explore the limits of computation, we have been studying …
  … computation about other computation …**
  - <u>Thought</u>: **Is there a decider (algorithm) to determine whether a TM is an decider?**

Hmmm, this doesn't feel right …



Next time: Is A_{TM} decidable?

$A_{\mathsf{TM}} = \{\langle M, w\rangle \mid M \text{ is a TM and } M \text{ accepts } w\}$