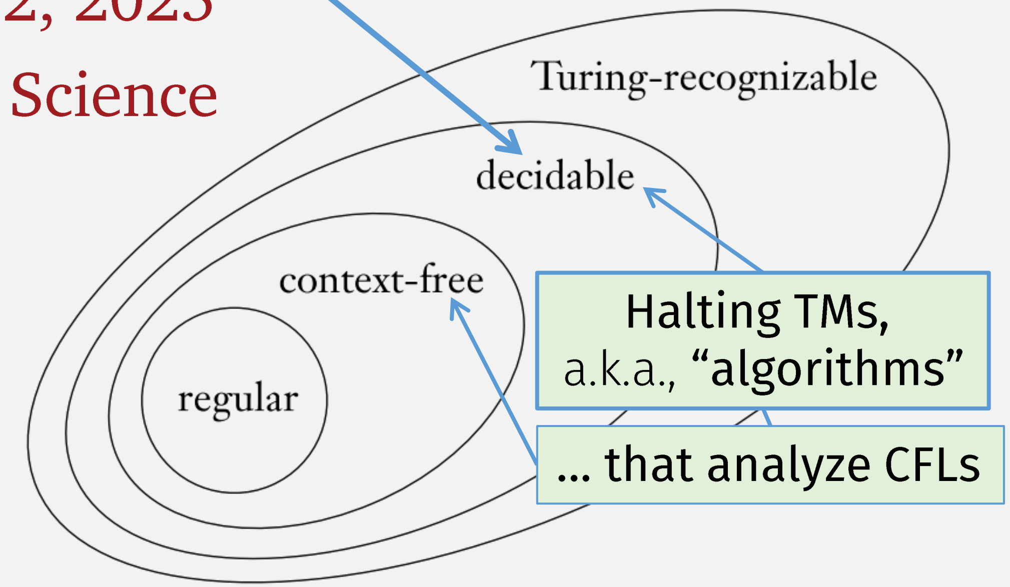
CS 420 / CS 620

Decidability for CFLs

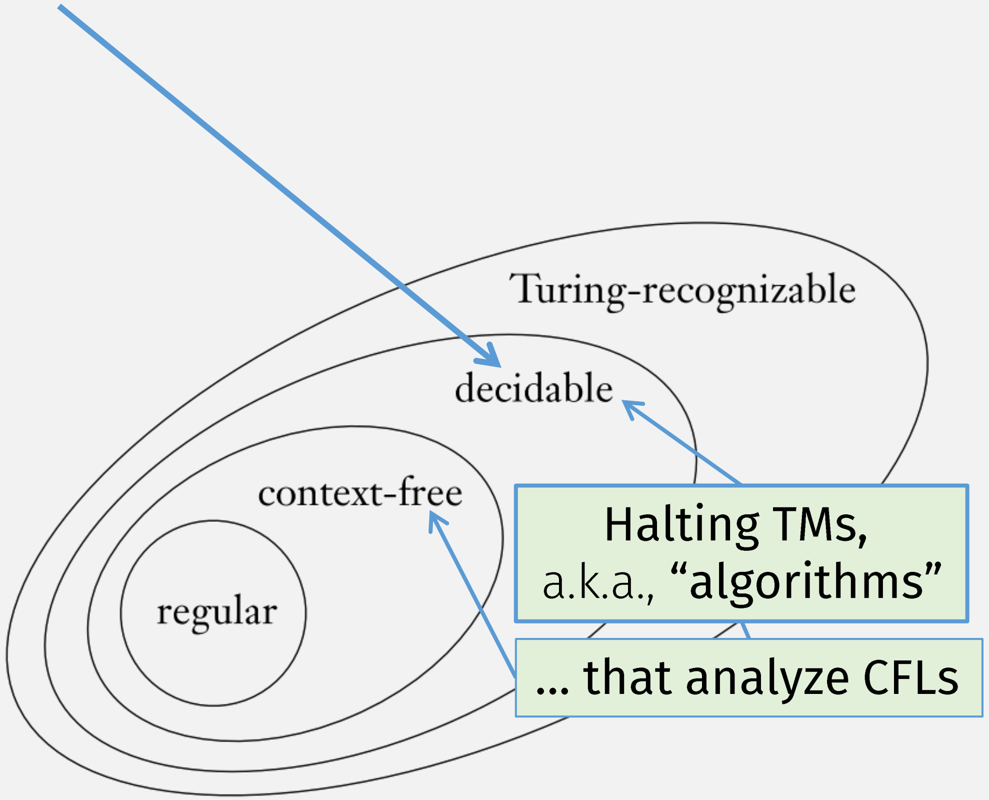
Wednesday, November 12, 2025

UMass Boston Computer Science



Announcements

- HW 10
  - Out: **Mon 11/10 12pm (noon)**
  - Due: **Mon 11/17 12pm (noon)**



Previously

How to Design Deciders

- A **Decider** is a TM ...
  - See previous slides on how to:
    - write a **high-level TM description**
    - ... that uses **encoded** input strings
  - E.g., M = On input <B, w>, where B is a DFA and w is a string: ...

- A **Decider** is a TM ... that must always **halt**
  - Can only: **accept** or **reject**
  - Cannot: go into an infinite loop

- So a **Decider** definition must include: an extra **termination argument:**
  - Explains how every step in the TM halts
  - (Pay special attention to loops)

- Remember our analogy: TMs ~ Programs ... so *Creating* a TM ~ Programm*ing*
  - To design a TM, think of how to write a program (function) that does what you want

Previously

How to Design Deciders, Part 2

Hint:

- **Previous theorems / constructions** are a **"library" of reusable TMs**
- **When creating a TM,** use this **"library"** to help you**!**
  - Just like libraries are useful when programming!
- E.g., **"Library" for DFAs:**
  - **NFA→DFA, RegExpr→NFA**
  - $\text{UNION}_{\text{DFA}}$, STAR_{PDA}, ENC, reverse
  - **Deciders for:** A_{DFA}, A_{NFA}, A_{REX}, …

Creating Computations: Then and Now

Up to now

Given: a **language** → i.e., what a **computation "should do"** → Analogy: **software requirements**

Want to: **construct machine** that recognizes the language → i.e., what a **computation "does"** → Analogy: **write code** that follows requirements

Need to: **write Examples Table** to "prove" machine recognizes the language → i.e., does **computation "do"** what it **"should do"?** → Analogy: **write tests** to "prove" code "works"

Now

Given: a **machine1** and (something about) a **language**

Analogy: **code** and its **requirements**

Want to: **construct** ~~machine~~ **terminating machine2** that computes whether **machine1 recognizes language**

Naïve solution, write infinite tests: **run machine1 …**

- for **every string** in **language** and **check if accept**
- for **every string not** in **language** and **check if reject**

Analogy: (algorithm) **code** to **prove** (no quotes!) whether **other code "works"** … … without **running it**, i.e., **prediction!**

Last Time

Algorithms About Regular Langs

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

E_{DFA} **Decider:** graph reachability algorithm
(is there any path from start state to accept state)

Given: a **machine1** and a **language**

terminating

Want to: **construct machine2** that computes whether **machine1 recognizes language**

Last Time

Algorithms About Regular Langs

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

Given: **machine**(s) and (something about their) **language**, i.e., their expected "run" behavior

Want to: **construct machine** that ~~computes~~ whether **machine**(s) have that "run" behavior

terminating

predicts

EQ_{DFA} **Decider:** Use **neg, union, intersection closure constructions** + E_{DFA} **decider** to determine when **symmetric difference is ∅**

Next: Algorithms (Decider TMs) for CFLs?

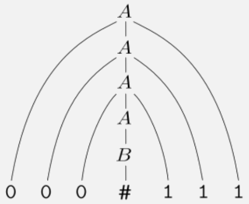
- What can we **predict** about CFG or PDA computation?

<u>Thm</u>: 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** determining: possible to **<u>parse</u>** "program" w for **a programming language** with **grammar** G?

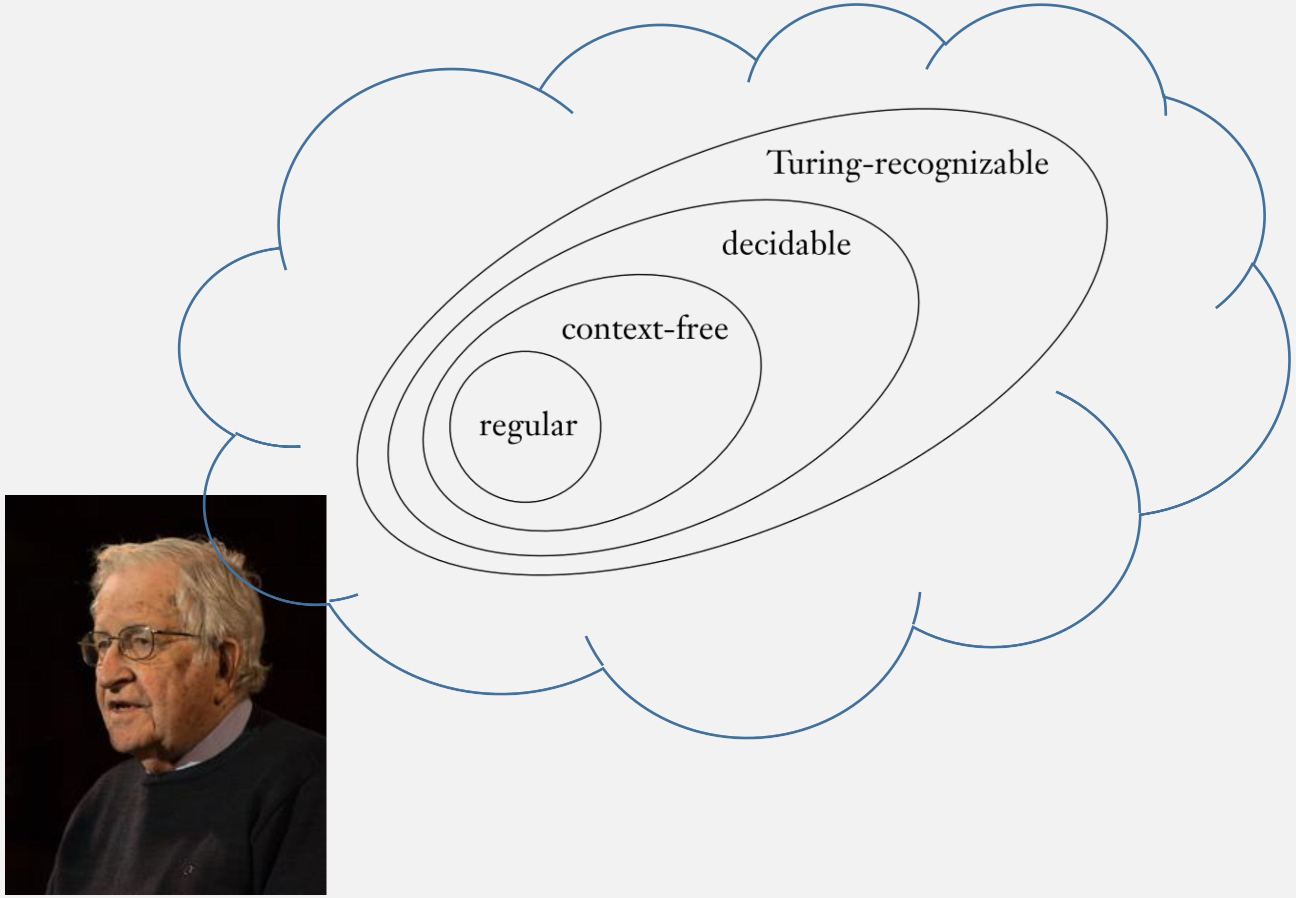
- **A Decider for this problem** could … ?
  - **Try every possible derivation of** G, **and check if it's equal to** w?
  - But this **<u>might never halt</u>**
    - E.g., what if there are **rules like:** $\mathbf{S} \rightarrow \mathbf{0S}$ or $\mathbf{S} \rightarrow \mathbf{S}$
      - (This **TM**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



<u>Idea</u>: **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 hierarchy 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

- $S \to AB$
- $B \to AB$
- $A \to \mathbf{a}$
- $B \to \mathbf{b}$

Makes the string long enough

Convert variables to terminals

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

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

☑
$$A \to BC$$
$$A \to a$$

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 $S \to \varepsilon$, 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, key step: **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

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

⇒

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

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 $R \to uv$ (A is deleted)
   - Must cover all combinations of deletions if A appears more than once in a RHS
     - E.g., if $R \to uAvAw$ is a rule, add 3 rules: $R \to uvAw$, $R \to uAvw$, $R \to uvw$

deleted A | deleted A | deleted As

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

First, remove

Then add (for deleted Bs)

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

Then add (for deleted As)

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 of deletions 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 B and S RHSs to A

Termination argument of this algorithm?

(Algorithm only loops over finite num of rules)

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 of deletions 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 $A \to u$
4. Split up rules with RHS longer than length 2
   - E.g., $A \to wxyz$ becomes $A \to wB$, $B \to xC$, $C \to yz$
5. Replace all terminals on RHS with new rule
   - E.g., for above, add $W \to w$, $X \to x$, $Y \to y$, $Z \to z$

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

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

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

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

Recall:

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

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

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

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 $A \rightarrow U_1U_2\cdots U_k$ 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

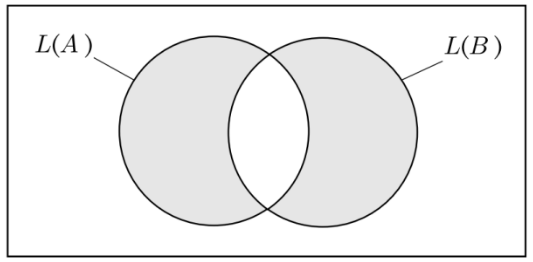
Termination argument?

Thm: EQ_{CFG} is a decidable language?

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

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

- 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 not closed 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:**

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

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

IF-THEN **stmt** (for proving "closed" ops):

If A and B are **CFLs, then** $A \cap B$ is a **CFL**

- 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!

IF-THEN stmt:

If A is a CFL, then $\overline{A}$ is a CFL

- Assume: CFLs closed under complement

Then: if G_1 and G_2 context-free

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

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

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

$L(G_1) \cap L(G_2)$ 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 <u>any</u> TM to do this!!!)
  - (details later)

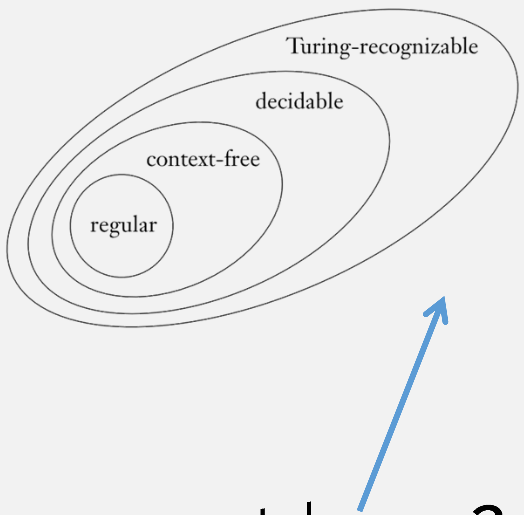
- I.e., this is an **<u>impossible computation</u>!**

(has no machine that recognizes it!)

Summary Algorithms About CFLs

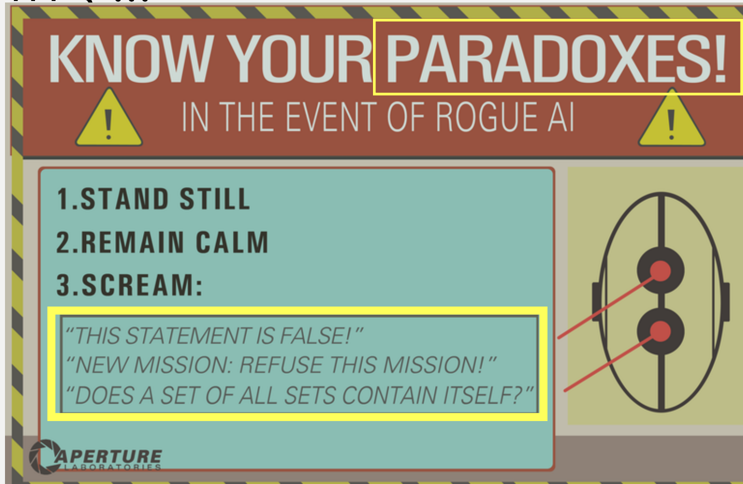
- $A_{\text{CFG}} = \{\langle G, w\rangle \mid G \text{ is a CFG that generates string } w\}$
  - **Decider:** Convert grammar to Chomsky Normal Form
  - Then **check all possible derivations up to length 2|*w*| - 1 steps**
- $E_{\text{CFG}} = \{\langle G\rangle \mid G \text{ is a CFG and } L(G) = \emptyset\}$
  - **Decider:** Compute "reachability" of start variable from terminals
- $EQ_{\text{CFG}} = \{\langle G, H\rangle \mid G \text{ and } H \text{ are CFGs and } L(G) = L(H)\}$
  - **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 …
  - Thought: 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?

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