Decidability for CFLs

Monday, March 21, 2022
Announcements

• Welcome back!

• HW 6 in
  • Due Sun 3/20 11:59pm

• HW 7 out
  • Due Sun 3/27 11:59pm

• No using Chegg please
Last Time: Algorithms About Regular Langs

- \( A_{DFA} = \{ \langle B, w \rangle \mid B \text{ is a DFA that accepts input string } w \} \)
  - Decider: Implements extended \( \delta \) function

- \( A_{NFA} = \{ \langle B, w \rangle \mid B \text{ is an NFA that accepts input string } w \} \)
  - Decider: Uses NFA→DFA decider + \( A_{DFA} \) decider

- \( A_{REX} = \{ \langle R, w \rangle \mid R \text{ is a regular expression that generates string } w \} \)
  - Decider: Uses RegExpr→NFA decider + \( A_{NFA} \) decider

- \( E_{DFA} = \{ \langle A \rangle \mid A \text{ is a DFA and } L(A) = \emptyset \} \)
  - Decider: Reachability algorithm

- \( EQ_{DFA} = \{ \langle A, B \rangle \mid A \text{ and } B \text{ are DFAs and } L(A) = L(B) \} \)
  - Decider: Uses complement and intersection closure construction + \( E_{DFA} \)

Remember:

TMs = programs
Creating TM = programming
Previous theorems = library
**Thm:** $A_{\text{CFG}}$ 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 $w$ in (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: $S \rightarrow \emptyset S$ or $S \rightarrow S$
    - 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 (sort of) invented this course too!
A context-free grammar is in **Chomsky normal form** if every rule is of the form

\[
A \rightarrow BC \quad \text{(non-start) Variables only}
\]

\[
A \rightarrow a \quad \text{2 rule shapes}
\]

\[
\text{where } a \text{ is any terminal and } A, B, \text{ and } C \text{ are any variables—except that } B \text{ and } C \text{ may not be the start variable. In addition, we permit the rule } S \rightarrow \varepsilon, \text{ where } S \text{ is the start variable.}
\]
Chomsky Normal Form Example

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

- **To generate string of length: 2**
  - Use $S$ rule: 1 time; $A$ or $B$ rules: 2 times
  - $S \Rightarrow AB \Rightarrow aB \Rightarrow ab$
  - Derivation total steps: $1 + 2 = 3$

- **To generate string of length: 3**
  - Use $S$ rule: 2 times; $A$ or $B$ rules: 3 times
  - $S \Rightarrow AS \Rightarrow AAB \Rightarrow aAB \Rightarrow aaB \Rightarrow aab$
  - Derivation total steps: $2 + 3 = 5$

- **To generate string of length: 4**
  - Use $S$ rule: 3 times; $A$ or $B$ rules: 4 times
  - $S \Rightarrow AS \Rightarrow AAS \Rightarrow AAAB \Rightarrow aAAB \Rightarrow aaAB \Rightarrow aaaB \Rightarrow aaab$
  - Derivation total steps: $3 + 4 = 7$

...
Chomsky Normal Form: Number of Steps

To generate a string of length $n$:

- $n - 1$ steps: to generate $n$ variables
- $+ n$ steps: to turn each variable into a terminal

Total: $2n - 1$ steps

(A finite number of steps)
**Thm:** \( A_{\text{CFG}} \) 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:

\( S = \text{"On input } \langle G, w \rangle, \text{ where } G \text{ is a CFG and } w \text{ is a string:} \)

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.”

**Termination argument?**
Thm: Every CFG has a Chomsky Normal Form

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

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

\[
\begin{align*}
S & \to ASA \mid aB \\
A & \to B \mid S \\
B & \to b \mid \varepsilon \\
\end{align*}
\]

\[
\begin{align*}
S_0 & \to S \\
S & \to ASA \mid aB \\
A & \to B \mid S \\
B & \to b \mid \varepsilon \\
\end{align*}
\]
Thm: Every CFG has a Chomsky Normal Form

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: $R \rightarrow uvAw$, $R \rightarrow uAvw$, $R \rightarrow uvw$
Thm: Every CFG has a Chomsky Normal Form

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 $A \rightarrow u$
Thm: Every CFG has a Chomsky Normal Form

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 $W \rightarrow w, X \rightarrow x, Y \rightarrow y, Z \rightarrow z$
**Thm:** $A_{CFG}$ is a decidable language

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

**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.”

**Termination argument:**

**Step 1:** any CFG has only a finite # rules
**Step 2:** $2n - 1 = \text{finite # of derivations to check}$
**Step 3:** only 1 step
**Thm:** $E_{CFG}$ is a decidable language.

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

Recall:

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

$T = \text{“On input } \langle A \rangle, \text{ where A is a DFA:}$$
1. \text{Mark the start state of A.}$
2. \text{Repeat until no new states get marked:}$
3. \text{Mark any state that has a transition coming into it from any state that is already marked.}$
4. \text{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: $E_{\text{CFG}}$ 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 $G$

- Except go \textbf{backwards}, start from \textbf{terminals}, to avoid \textbf{looping}

$$R = \text{“On input } \langle G \rangle, \text{ where } G \text{ is a CFG:}$$

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, \textit{accept}; otherwise, \textit{reject}.”

Termination argument?
Thm: $\text{EQ}_{\text{CFG}}$ is a decidable language?

$\text{EQ}_{\text{CFG}} = \{(G, H) \mid G \text{ and } H \text{ are CFGs and } L(G) = L(H)\}$

Recall: $\text{EQ}_{\text{DFA}} = \{(A, B) \mid A \text{ and } B \text{ are DFAs and } L(A) = L(B)\}$

- Used Symmetric Difference

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

- where $C = \text{complement, union, intersection of machines } A \text{ and } B$
- 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:

\[ A = \{ a^m b^n c^n | m, n \geq 0 \} \]

\[ B = \{ a^n b^n c^m | m, n \geq 0 \} \]

• But \( A \cap B = \{ a^n b^n c^n | n \geq 0 \} \)

• ... which is not a CFL! (So we have a contradiction)
Complement of a CFL is not Closed!

- If CFLs closed under complement:

  \[
  \begin{align*}
  \text{if } G_1 \text{ and } G_2 \text{ context-free} & \Rightarrow L(G_1) \text{ and } L(G_2) \text{ context-free} \\
  L(G_1) \cup L(G_1) \text{ context-free} & \Rightarrow \text{Union of CFLs is closed} \\
  L(G_1) \cap L(G_2) \text{ context-free} & \Rightarrow \text{DeMorgan’s Law!} \\
  \text{But intersection is not closed for CFLS (prev slide)}
  \end{align*}
  \]
Thm: $E_{\text{CFG}}^Q$ is a decidable language?

$$E_{\text{CFG}}^Q = \{(G, H) | G \text{ and } H \text{ are CFGs and } L(G) = L(H)\}$$

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

- It’s not recognizable either!
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 can’t 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

• So what is not computable? i.e., what’s here?

• A way to test the limit of a computational model is to see what it can compute about computational models ...
  • Thought: Is there an decider (algorithm) to determine whether a TM is an decider?

Hmmm
Next time: Is $A_{\text{TM}}$ decidable?

$$A_{\text{TM}} = \{\langle M, w \rangle | M \text{ is a TM and } M \text{ accepts } w \}$$
Check-in Quiz 3/21
On gradescope