Decidable Problems (i.e., Algorithms) about Context-Free Languages (CFLs)

Monday March 29, 2021
Announcements

• HW 6 due date past

• HW 7 due Sun 4/4 11:59pm EST
  • Remember to use your “library” of theorems

• HW 8 out soon
  • due Sun 4/11 11:59pm EST
  • Covers Ch 4-5 material (starting Wed)
Last time: Decidable DFA Langs (i.e., algorithms)

- $A_{DFA} = \{ \langle B, w \rangle | B \text{ is a DFA that accepts input string } w \}$

- $A_{NFA} = \{ \langle B, w \rangle | B \text{ is an NFA that accepts input string } w \}$

- $A_{REX} = \{ \langle R, w \rangle | R \text{ is a regular expression that generates string } w \}$

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

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

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:
  - Is there an algorithm to parse 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., if there is a rule like: $S \rightarrow 0S$ 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 an upper bound on the number of derivation steps?
Chomsky Normal Form
Noam Chomsky
A context-free grammar is in \textit{Chomsky normal form} if every rule is of the form
\[ A \rightarrow BC, \quad A \rightarrow \alpha \]
where \( \alpha \) 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
  - $+n$ steps: to turn each variable into a terminal
  - **Total**: $2n - 1$ steps

\[
\begin{align*}
A & \rightarrow BC \\
A & \rightarrow a
\end{align*}
\]
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

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

\[
\begin{align*}
S_0 & \rightarrow S \\
S & \rightarrow ASA | aB \\
A & \rightarrow B | S \\
B & \rightarrow b | \varepsilon
\end{align*}
\]
**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 $A \rightarrow \epsilon$
   - $A$ 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:** $A_{\text{CFG}}$ 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."
Thm: $E_{CFG}$ is a decidable language.

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

Recall:

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

$T =$ “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
Thm: $E_{\text{CFG}}$ is a decidable language.

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

- Create decider that calculates reachability for grammar $G$
  - Except go backwards, start from terminals, to avoid 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, accept; otherwise, reject.”
Thm: $E_{\text{CFG}}$ is a decidable language?

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

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

- 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!

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

• Not a CFL!
  • See textbook example 2.36
Complement of a CFL is not Closed!

• If CFLs closed under complement:

  \[
  \text{if } G_1 \text{ and } G_2 \text{ context-free} \\
  \overline{L(G_1)} \text{ and } \overline{L(G_2)} \text{ context-free} \\
  \overline{L(G_1)} \cup \overline{L(G_1)} \text{ context-free} \\
  \overline{L(G_1)} \cap \overline{L(G_2)} \text{ context-free}
  \]

DeMorgan’s Law!
Thm: $EQ_{CFG}$ is a decidable language?

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

• No!
  • You cannot decide whether two grammars represent the same lang!

• It’s not recognizable either!
  • (But we won’t learn how to prove this until Chapter 5)
Decidability of CFGs Recap

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

- $E_{CFG} = \{ \langle G \rangle \mid G \text{ is a CFG and } L(G) = \emptyset \}$
  - Compute “reachability” of start variable from terminals

- $EQ_{CFG} = \{ \langle G, H \rangle \mid G \text{ and } H \text{ are CFGs and } L(G) = L(H) \}$
  - We couldn’t prove that this is decidable!
  - (So you can’t use this theorem when creating another decider)
The Limits of Turing Machines?

• So TMs can express any “computation”
  • I.e., any (Python, Java, Racket, ...) program you write is a Turing Machine

• So why do we focus on TMs that process other machines?

• Because in CS420, we also want to study the limits of computation
  • And a good way to test the limit of a computational model is to see what it can compute about other computational models ...

• So what are the limits of TMs? I.e., what’s here?
  • Or out here?
Next time: $A_{TM}$ is undecidable

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

$A_{TM}$ = the problem of computers simulating other computers, e.g.:

$U = \text{“On input } \langle M, w \rangle, \text{ where } M \text{ is a TM and } w \text{ is a string:}$$
\begin{enumerate}
    \item Simulate $M$ on input $w$.
    \item If $M$ ever enters its accept state, accept; if $M$ ever enters its reject state, reject.”
\end{enumerate}$

I.e., will machines take over the world?
Kinds of Functions (a fn maps Domain -> Range)

- **Injective**
  - A.k.a., “one-to-one”
  - Every element in Domain has a unique mapping
  - How to remember:
    - Domain is mapped “in” to the Range

- **Surjective**
  - A.k.a., “onto”
  - Every element in Range is mapped to
  - How to remember:
    - “Sur” = “over” (eg, survey); Domain is mapped “over” the Range

- **Bijective**
  - A.k.a., “correspondence” or “one-to-one correspondence”
  - Is both injective and surjective
  - Unique pairing of every element in Domain and Range
Countability

• A set is “countable” if it is:
  • Finite
  • Or, there exists a bijection between the set and the natural numbers
    • This set is then considered to have the same size as the set of natural numbers
    • This is called “countably infinite”
Exercise: Which set is larger?

• The set of:
  • Natural numbers, or
  • Even numbers?

• They are the **same** size! Both are **countably infinite**
  • Bijection:

<table>
<thead>
<tr>
<th>$n$</th>
<th>$f(n) = 2n$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>:</td>
<td>:</td>
</tr>
</tbody>
</table>
Exercise: Which set is larger?

• The set of:
  • Natural numbers $\mathbb{N}$, or
  • Positive rational numbers? $\mathbb{Q} = \{ \frac{m}{n} \mid m, n \in \mathbb{N} \}$
• They are the same size! Both are countably infinite
Exercise: Which set is larger?

• The set of:
  • Natural numbers $\mathcal{N}$, or
  • Positive rational numbers? $Q = \left\{ \frac{m}{n} \mid m, n \in \mathcal{N} \right\}$

• They are the same size! Both are countably infinite.
Exercise: Which set is larger?

• The set of: \[ \mathbb{N}, \mathbb{R} \]
  • Natural numbers, or
  • Real numbers?

• There are **more** real numbers. It is **uncountably infinite**.

Proof: next time!
Check-in Quiz 3/29

On gradescope