Problems for Week Nine

Context-Free Grammars
Here’s some practice problems to help you get comfortable designing CFGs. There are a number of patterns that come up over the course of these problems, and we hope that by the time you’ve finished working through them you have a deeper understanding of how CFGs work!

i. Let $\Sigma = \{a, b\}$ and let $L = \{ w \in \Sigma^* \mid w$ has no $a$’s or has no $b$’s $\}$. Write a CFG for $L$.

ii. Let $\Sigma = \{a, b\}$ and let $L = \{ w \in \Sigma^* \mid w$ has at least one a and at least one b $\}$. Write a CFG for $L$.

iii. Let $\Sigma = \{a, b\}$ and let $L = \{ a^n b a^n \mid n \in \mathbb{N} \}$. Write a CFG for $L$.

iv. Let $\Sigma = \{a, b\}$ and let $L = \{ a^n b^{2n} \mid n \in \mathbb{N} \}$. Write a CFG for $L$.

v. Let $\Sigma = \{a, b\}$ and let $L = \{ a^n b^n m \mid n, m \in \mathbb{N}$ and $n \leq m \leq 5n \}$. Write a CFG for $L$.

vi. Let $\Sigma = \{a, b\}$ and let $L = \{ a^n b^n \mid n, m \in \mathbb{N}$ and $n \neq m \}$. Write a CFG for $L$.

vii. Let $\Sigma = \{a, b, c\}$ and let $L = \{ a^n b^n c^n \mid n \in \mathbb{N}$ and $n = m$ or $n = p \}$. Write a CFG for $L$.

viii. Let $\Sigma = \{a, b\}$ and let $L = \{ a^n b^n \mid n \in \mathbb{N} \}$. Write a CFG for $L^*$, the Kleene closure of $L$.

ix. Let $\Sigma = \{a, b\}$ and let $L = \{ a^n b^n \mid n, m \in \mathbb{N}$ and either $n=2m$ or $m=2n \}$. Write a CFG for $L$.

x. Let $\Sigma = \{a, b\}$ and let $L = \{ a^n b^n \mid n \in \mathbb{N} \}$. Write a CFG for $\overline{L}$, the complement of $L$.

Turing Machines
Although much of our discussion of Turing machines takes place at a high level, it’s still instructive to try to design Turing machines at the level of individual states.

i. Let $\Sigma = \{\emptyset, 1\}$ and let $L = \{ w \in \Sigma^* \mid w$ is a palindrome $\}$ (recall that a palindrome is a string that’s the same when read forwards and backwards). Draw a state-transition diagram of a TM for $L$.

ii. Draw the state-transition diagram for a TM whose language is $\{ a^n b^n c^n \mid n \in \mathbb{N} \}$.
The Story So Far
From the “lava diagram” in lecture, you probably noticed that
\[ \text{REG} \subset \text{R} \subset \text{RE} \]
Here, \( \text{REG} \) is the class of all regular languages, \( \text{R} \) is the class of all decidable languages, and \( \text{RE} \) is the class of all recognizable languages.
On Problem Set Eight, you’ll show that \( \text{REG} \subseteq \text{R} \).

i. Show that \( \text{REG} \neq \text{R} \).
ii. Show that \( \text{R} \subseteq \text{RE} \). (Hint: What’s the definition of \( \text{R} \)? What’s the definition of \( \text{RE} \)? Expand out the requisite terms and see what you find.)

Closure Properties of \( \text{R} \)
This question explores various closure properties of \( \text{R} \). Because \( \text{R} \) corresponds to decidable problems, languages in \( \text{R} \) are precisely the languages for which you can write a method
\[ \text{bool inL(string w)} \]
such that
- for any string \( w \in L \), calling \( \text{inL}(w) \) returns true.
- for any string \( w \not\in L \), calling \( \text{inL}(w) \) returns false.
This means that we can reason about closure properties of the decidable languages by writing actual pieces of code.

i. Let \( L_1 \) and \( L_2 \) be decidable languages over the same alphabet \( \Sigma \). Prove that \( L_1 \cup L_2 \) is also decidable. To do so, suppose that you have methods \( \text{inL1} \) and \( \text{inL2} \) matching the above conditions, then show how to write a method \( \text{inL1uL2} \) with the appropriate properties. Then, briefly justify why your construction is correct.

ii. Repeat problem (i), except proving that the \( \text{R} \) languages are closed under concatenation.

Decidable Languages
All regular languages are decidable, but below is a purported proof that the regular language described by the regular expression \( a^*b \) is undecidable:

**Theorem:** \( a^*b \) is undecidable.

**Proof:** By contradiction; assume \( a^*b \) is decidable. Let \( D \) be a decider for it. Consider what happens when we run \( D \) on a string of infinitely many \( a \)'s followed by a \( b \) and on a string of infinitely many \( a \)'s. Let’s call this first string \( x \) and the second string \( y \). Since \( D \) is a decider, it halts on all inputs, and therefore cannot run for an infinitely long time. Therefore, \( D \) must halt before reading the last character of \( x \) and the last character of \( y \). Because \( x \) and \( y \) are the same except for their last character, we see that \( D \) must have the same behavior when run on \( x \) and when run on \( y \). If \( D \) accepts \( x \), then \( D \) also accepts \( y \), but \( y \) is not in the language \( a^*b \). Otherwise, \( D \) rejects \( x \), but \( x \) is in the language \( a^*b \). Both cases contradict the fact that \( D \) is a decider for \( a^*b \). We have reached a contradiction, so our assumption must have been wrong. Thus \( a^*b \) is undecidable. ■

What’s wrong with this proof?