Problems for Week Nine

Here are some problems you can use to get a better feel for TMs, decidability, and self-reference.

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 $\Sigma = \{0, 1\}$ and let $L = \{ w \in \Sigma^* \mid w \text{ is a palindrome } \}$ (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.

i. Show that $\text{REG} \subseteq \text{R}$. To do so, explain how to directly turn a DFA for a language $L$ into a decider for $L$. There's a rather simple construction you can use here to do this.

ii. Show that $\text{R} \subseteq \text{RE}$ in two different ways: first, show how to convert a decider for a language $L$ into a recognizer for a language $L$, then show how to convert a decider for a language $L$ into a verifier for a language $L$. 
Closure Properties of R

This question explores various closure properties of \( R \). Because \( R \) corresponds to decidable problems, languages in \( R \) 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 \( w \notin L \), 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 \( \text{inL}_1 \) and \( \text{inL}_2 \) matching the above conditions, then show how to write a method \( \text{inL}_{1\cup2} \) with the appropriate properties. Then, briefly justify why your construction is correct.

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

iii. Take a look at the argument you came up with for part (i) of this problem. If \( \text{inL}_1 \) and \( \text{inL}_2 \) are recognizers rather than deciders, they're allowed to loop infinitely in the case where the input string doesn't belong to \( L_1 \) or \( L_2 \), respectively. Does the construction you came up with work to show that \( L_1 \cup L_2 \) is recognizable in this case? Why or why not?

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?
Self-Reference and Decidability

Consider the language \( L = \{ \langle M \rangle \mid M \text{ is a TM that accepts at least one string} \} \).

i. As we've seen in class, there's a close correspondence between languages and decision problems. What problem does the above language correspond to?

This language is undecidable, meaning that the problem you've described in part (i) cannot be solved by any algorithm. Let's now go see why this is.

ii. Suppose for the sake of contradiction that \( L \in R \). This means that we could write a function

\[
\text{bool acceptsAtLeastOneString(string program)}
\]

that accepts as input the source code of a program, then returns true if the program accepts at least one string and returns false otherwise. Write a self-referential program that uses this function to obtain a contradiction. As a hint, recall the general template for these sorts of programs: have the program ask whether it accepts at least one string, then have it do the opposite of whatever it determines it's supposed to do.

iii. Formalize your reasoning from part (ii) by writing a formal proof that \( L \notin R \). To do so, follow the proof template from lecture: assume that \( L \in R \), describe what that assumption entails, describe (at a high level) a self-referential TM that causes a contradiction, then explain why you get a contradiction in all cases.