Problem 1

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

a. Every context-free language can also be decided by a Turing machine. 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 the state-transition diagram for a Turing machine whose language is \( L \).

b. We’ve shown that the language \( L = \{ a^n b^n c^n \mid n \in \mathbb{N} \} \) is not context-free and therefore cannot be recognized by a pushdown automaton (nor by a finite automaton). However, it can be recognized by a Turing machine. Draw a state-transition diagram for a Turing machine whose language is \( L \).

Problem 2

Design a Turing machine that decides the language \( \{ w \mid w \text{ contains twice as many 0s as 1s} \} \) over the alphabet \( \{0, 1\} \). Give an implementation-level description in the style of Example 3.7 on page 171 (not a formal description or a state diagram).

Problem 3

In class, we alluded to the fact that \( \text{REG} \) (the class of all regular languages) is a subset of \( \text{R} \) (the class of all Turing-decidable languages). Describe a construction that, given a \( \text{DFA} \) \( D \) produces a decider \( D' \) where \( L(D) = L(D') \). Briefly justify why the TM \( D' \) you construct is a decider and why it accepts precisely the strings that \( D \) accepts.

Problem 4

What does it mean to solve a problem? If \( L \) is a language over \( \Sigma \) and \( M \) is a Turing machine with input alphabet \( \Sigma \), any of these properties may hold:

The 27th is Wednesday, but you can turn in your work in class on Tuesday if you’re done!
1. If \( M \) is a decider (i.e., halts on all inputs).
2. For any string \( w \in \Sigma^* \), if \( M \) accepts \( w \), then \( w \in L \).
3. For any string \( w \in \Sigma^* \), if \( M \) rejects \( w \), then \( w \notin L \).

To claim that a Turing machine solves a problem, it seems like it should have at least some of these properties. But we can show that having just two of these properties doesn’t say much.

a. Prove that if \( L \) is any language over \( \Sigma \), then there is a Turing machine \( M \) that satisfies properties 1 and 2 with respect to \( L \).

b. Prove that if \( L \) is any language over \( \Sigma \), then there is a Turing machine \( M \) that satisfies properties 1 and 3 with respect to \( L \).

c. Prove that if \( L \) is any language over \( \Sigma \), then there is a Turing machine \( M \) that satisfies properties 2 and 3 with respect to \( L \).

d. Suppose \( L \) is a language over \( \Sigma \) for which there is a Turing machine \( M \) that satisfies properties 1, 2, and 3. What can you say about \( L \)?

Problem 5

Give a high-level description of a Turing machine \( M \) such that \( L(M) \in R \), but \( M \) is not a decider. (This shows that just because a Turing machine's language is decidable, it's not necessarily the case that the TM itself must be a decider.)

This doesn't need to be a specific language. How could you modify a decider for some language \( L \) to be only a recognizer?