1. (4.3 from textbook) Let \( ALL_{DFA} = \{ \langle A \rangle \mid A \text{ is a DFA that recognizes } \Sigma^* \} \). Show that \( ALL_{DFA} \) is decidable.

**Solution** We will create a Turing Machine \( M \) to decide \( ALL_{DFA} \). Hard-wired into \( M \) is the specification \( \langle B \rangle \) for a DFA \( B \) that recognizes \( \Sigma^* \); this is easy to build by making the start state an accept state with self-loops corresponding to each symbol in \( \Sigma \). On input \( \langle A \rangle \), \( M \) will feed \( \langle A, B \rangle \) as input to \( TM_{EQ_{DFA}} \), a TM that decides whether the languages of two DFA’s are equal. We know this TM exists by Theorem 4.5. If \( TM_{EQ_{DFA}} \) accepts, then \( L(A) = L(B) = \Sigma^* \), so \( M \) must accept. If \( TM_{EQ_{DFA}} \) rejects, then \( L(A) \neq L(B) \), so \( L(A) \neq \Sigma^* \), hence \( M \) must reject. Because \( M \) decides \( ALL_{DFA} \), we have proven that \( ALL_{DFA} \) is decidable.
2. (4.5 from textbook)
Let $\text{INFINITE}_{DFA} = \{ \langle A \rangle \mid A \text{ is a DFA and } L(A) \text{ is an infinite language} \}$. Show that $\text{INFINITE}_{DFA}$ is decidable.

**Solution** We will create a Turing Machine $M$ to decide $\text{INFINITE}_{DFA}$. A DFA that recognizes an infinite language must contain a loop in its state transitions. This loop can contain just a single state, in the case of a self-loop. It follows that $M$ must consider each state $q$ and determine whether a loop contains $q$. If so, then $M$ must determine whether there is a path from any state in the loop to an accept state. Finally, $M$ must determine whether there is a path from the start state to $q$. These steps are performed by the pseudocode below for $M$:

```plaintext
for (each state q in A) {
    clear marks; // initialization
    mark each state that has a transition from q;
    while (new states can be marked) {
        mark each state that has a transition from a marked state;
        if (state q is marked) {
            // this means we have found a loop containing q
            if (an accept state is marked) {
                // this means an accept state is reachable from
                // a state in the loop, so we must now check
                // whether q is reachable from the start state.
                clear marks; // initialize for second search
                mark the start state;
                while (new states can be marked) {
                    mark each state that has a transition
                    from a marked state;
                    if (state q is marked) {
                        accept; // all conditions are satisfied.
                    }
                }
            }
        }
    }
    reject;
}
```

Because $M$ decides $\text{INFINITE}_{DFA}$, we have proven that $\text{INFINITE}_{DFA}$ is decidable.
3. (5.2 from textbook) Show that $EQ_{CFG}$ is co-Turing-recognizable.

**Solution** We will create a Turing Machine $M$ to recognize the complement of $EQ_{CFG}$, denoted $\overline{EQ_{CFG}}$. The definition of $\overline{EQ_{CFG}}$ is

$$\overline{EQ_{CFG}} = \{\text{strings not of the form } \langle G_1, G_2 \rangle, \text{ where } G_1, G_2 \text{ are CFG's}\} \cup \{\langle G_1, G_2 \rangle \mid G_1, G_2 \text{ are CFG's and } L(G_1) \neq L(G_2)\}.$$ 

Our Turing Machine $M$ must first accept all “junk” strings, which are in the first set. Then it considers strings in the second set, which are of the form $\langle G_1, G_2 \rangle$. By Theorem 4.6, $A_{CFG}$ is decidable; in other words, a Turing Machine can decide whether a given CFG generates a given string. Let $TM_{A_{CFG}}$ be a TM that decides $A_{CFG}$.

For each string $w$ in $\Sigma^*$, where $\Sigma = \Sigma_{G_1} \cup \Sigma_{G_2}$ is the union of the terminals of $G_1$ and $G_2$, $M$ will feed $\langle G_1, w \rangle$ as input to $TM_{A_{CFG}}$; let the result be a boolean variable $g_1$ that is TRUE if $TM_{A_{CFG}}$ accepts $\langle G_1, w \rangle$ and FALSE otherwise. Next $M$ will feed $\langle G_2, w \rangle$ as input to $TM_{A_{CFG}}$; let the result be a boolean variable $g_2$ that is TRUE if $TM_{A_{CFG}}$ accepts $\langle G_2, w \rangle$ and FALSE otherwise.

(Recall that the symbol $\oplus$ denotes the XOR operation.) If $g_1 \oplus g_2 = \text{TRUE}$, then $w$ is in either $L(G_1)$ or $L(G_2)$ but not both; therefore $L(G_1) \neq L(G_2)$, and $M$ accepts $\langle G_1, G_2 \rangle$. If $g_1 \oplus g_2 = \text{FALSE}$, then $M$ considers the next $w$.

Note that $M$ does not need to reject, because it is a recognizer, not a decider. Because $M$ recognizes $\overline{EQ_{CFG}}$, we have proven that $\overline{EQ_{CFG}}$ is Turing-recognizable. Therefore $EQ_{CFG}$ is co-Turing-recognizable.
4. (5.3 from textbook) Find a match in the following instance of the PCP:

\[
\left\{ \begin{array}{c}
[ab] \\
[abab] \\
[b] \\
[aba] \\
[aa]
\end{array} \right\}
\]

Solution

\[
\left\{ \begin{array}{c}
[ab] \\
[abab] \\
[b] \\
[aba] \\
[aa]
\end{array} \right\}
\]

5. (5.10 from textbook) Let \( J = \{ w \mid w = 0x \text{ for some } x \in A_{TM} \text{ or } w = 1y \text{ for some } y \in \overline{A_{TM}} \} \). Show that neither \( J \) nor \( \overline{J} \) is Turing-recognizable.

Solution First we will demonstrate a reduction \( f : \Sigma^* \to \Sigma^* \) of \( A_{TM} \) to \( J \). Given a string \( z \in \Sigma^* \), let \( f(z) = 1z \). By definition of \( J \), \( z \in \overline{A_{TM}} \) iff \( 1z \in J \). Thus \( f \) is a reduction of \( \overline{A_{TM}} \) to \( J \), so \( \overline{A_{TM}} \leq_m J \). Because \( A_{TM} \) is not Turing-recognizable, by Corollary 5.23 \( J \) is not Turing-recognizable.

Now we will demonstrate a reduction \( g : \Sigma^* \to \Sigma^* \) of \( A_{TM} \) to \( J \). Given a string \( t \in \Sigma^* \), let \( g(t) = 0t \). By definition of \( J \), \( t \in A_{TM} \) iff \( 0t \in J \). Thus \( g \) is a reduction of \( A_{TM} \) to \( J \), so \( A_{TM} \leq_m J \). A function that reduces language \( L_1 \) to language \( L_2 \) also reduces \( \overline{L_1} \) to \( \overline{L_2} \); hence \( g \) is also a reduction from \( \overline{A_{TM}} \) to \( J \), so \( \overline{A_{TM}} \leq_m J \). Because \( \overline{A_{TM}} \) is not Turing-recognizable, by Corollary 5.23 \( \overline{J} \) is not Turing-recognizable.

Therefore we have proven that neither \( J \) nor \( \overline{J} \) is Turing-recognizable.

6. (5.11 from textbook) Give an example of an undecidable language \( B \), where \( B \leq_m \overline{B} \).

Solution As it happens, the language \( J \) from problem 5.10 will work. We must prove that \( J \leq_m \overline{J} \). Define the function \( h : \Sigma^* \to \Sigma^* \) as follows:

Given a string \( u \in J \), if \( u = 0x \) where \( x \in A_{TM} \), let \( h(u) = 1x \). Then \( h(u) \in \overline{J} \) (because \( 1x \) cannot be in \( J \)).

On the other hand, if \( u = 1y \) where \( y \in \overline{A_{TM}} \), let \( h(u) = 0y \). Then \( h(u) \in J \) (because \( 0y \) cannot be in \( J \)).

Finally, we need to handle the case where \( u = \epsilon \). (We can assume without loss of generality that \( \Sigma = \{0, 1\} \), so all other strings in \( \Sigma^* \) begin with 0 or 1.) Let \( h(\epsilon) = 0c \), where \( c \) is a (fixed) element of \( A_{TM} \). Notice that \( \epsilon \notin J \) and \( h(\epsilon) \notin \overline{J} \), so this definition of \( h(\epsilon) \) does not violate the requirement that \( u \in J \) iff \( h(u) \in \overline{J} \).

The function \( h \) is a reduction of \( J \) to \( \overline{J} \). Therefore \( J \leq_m \overline{J} \).
7. Consider the Turing machine $M_2$ of figure 3.4. The tape alphabet $\Gamma$ is $\{0, x, \sqcup\}$. Give rules $R$ of a Semi-Thue system such that a word $w \in \{0\}^+$ is in $L(M_2)$ iff $w\sqcup \Rightarrow q_{\text{accept}}$. For simplicity, please use the rules $\gamma q_{\text{accept}} \rightarrow q_{\text{accept}}$ and $q_{\text{accept}}\gamma \rightarrow q_{\text{accept}}$ for all $\gamma \in \Gamma$.

**Solution (by Rene)** The rules in $R$ are given below. Note that the last six rules are the erasing rules, used after the $q_{\text{accept}}$ state appears in the string.

\[
\begin{align*}
q_1 0 & \rightarrow \sqcup q_2 \\
q_2 x & \rightarrow x q_2 \\
q_2 0 & \rightarrow x q_3 \\
q_2 \sqcup & \rightarrow \sqcup q_{\text{accept}} \\
q_3 x & \rightarrow x q_3 \\
q_3 0 & \rightarrow 0 q_4 \\
0 q_3 \sqcup & \rightarrow q_5 0 \sqcup \\
x q_3 \sqcup & \rightarrow q_5 x \sqcup \\
\sqcup q_3 \sqcup & \rightarrow q_5 \sqcup \sqcup \\
q_4 x & \rightarrow x q_4 \\
q_4 0 & \rightarrow x q_3 \\
q_5 \sqcup & \rightarrow \sqcup q_2 \\
0 q_5 0 & \rightarrow q_5 0 0 \\
x q_5 0 & \rightarrow q_5 x 0 \\
\sqcup q_5 0 & \rightarrow q_5 \sqcup 0 \\
0 q_5 x & \rightarrow q_5 0 x \\
x q_5 x & \rightarrow q_5 x x \\
\sqcup q_5 x & \rightarrow q_5 \sqcup x \\
0 q_{\text{accept}} & \rightarrow q_{\text{accept}} \\
q_{\text{accept}} 0 & \rightarrow q_{\text{accept}} \\
\sqcup q_{\text{accept}} & \rightarrow q_{\text{accept}} \\
q_{\text{accept}} \sqcup & \rightarrow \sqcup q_{\text{accept}} \\
x q_{\text{accept}} & \rightarrow q_{\text{accept}} \\
q_{\text{accept}} x & \rightarrow q_{\text{accept}}
\end{align*}
\]