4.2 Solution:

\[ \text{EQDFA}_\text{REX} = \{ \langle A, R \rangle \mid A \text{ is a DFA, } R \text{ is a regular expression, and } L(A) = L(R) \}. \]

We design a TM \( M \) that decides \( \text{EQDFA}_\text{REX} \):

\[ M = \text{On input } \langle A, R \rangle, \text{ where } A \text{ is a DFA and } R \text{ is a regular expression:} \]

1. Convert \( R \) to an equivalent DFA \( B \) by using Theorem 1.54. Then we have \( L(B) = L(R) \).
2. Run TM \( F \) in Theorem 4.5 on input \( \langle A, B \rangle \).
3. If \( F \) accepts, accept. If \( F \) rejects, reject.”

4.3 Solution:

We design a TM \( T' \) to decide \( \text{ALL}_{\text{DFA}} \):

\[ T' = \text{On input } \langle A \rangle, \text{ where } A \text{ is a DFA:} \]

1. Construct a DFA \( B \) that can accept the complement of \( L(A) \).
2. Run TM \( T \) in Theorem 4.4 on input \( \langle B \rangle \).
3. If \( T \) accepts, accept; otherwise reject.

4.4 Solution:

We could apply Theorem 4.7, and design a TM \( S \) as follows:

\[ S = \text{On input } \langle G \rangle, \text{ where } G \text{ is a CFG:} \]

1. Convert \( G \) to an equivalent grammar in Chomsky normal form.
2. List all derivations with 1 step.
3. If any of these derivations generate \( \varepsilon \), accept. If not, reject.”

4.6 Solution:

Since \( B \) is the set of all infinite sequences over \( \{0, 1\} \), then each element in \( B \) is an infinite sequence such as \( (a_1, a_2, a_3, \ldots) \), where \( a_i = 0 \) or \( 1 \). We prove the claim by contradiction. Suppose \( B \) is countable. Then we define a correspondence \( f \) between \( N = \{1, 2, 3, \ldots\} \) and \( B \).

For an arbitrary \( k \in N \), let \( f(k) = (a_{k1}, a_{k2}, \ldots) \), where \( a_{ki} \) is the ith bit in the kth sequence.
Now we define a new sequence \( c = (c_2, c_2, \ldots) \in B \), where \( c_i \) is the opposite of the \( i \)th bit in the \( i \)th sequence. Thus, \( c \in B \) is different from any sequence by at least one bit, so \( c \) does not equal to \( f(k) \) for any \( k \), which is a contradiction, which means \( B \) is uncountable.

4.10 Solution:

There are many possible solutions. But each uses the idea of the pumping lemma for context-free languages in some way. In particular, if \( L \) is a CFL that has a string that can be pumped, \( L \) is infinite. If no string of \( L \) can be pumped but \( L \) is context-free, then \( L \) is finite. So the question is how to find whether there is a string that can be pumped. The solution given here uses material that you learned in CSE 310 as well as 355.

First, we can construct a CFG \( G \) in Chomsky Normal Form whose language is equivalent to that of PDA \( M \). Now mark all terminals of the grammar, and make all variables initially unmarked. Then as long as there is a rule in which the variable on the left hand side is unmarked but everything on the right hand side is marked, mark the variable on the left hand side of the rule. Once this cannot be done any more, delete all unmarked variables and all rules in which an unmarked variable occurs. Every variable that remains can now generate at least one string containing only terminals, no variables. (We saw this in class before, when looking at \( E_{CFG} \).)

Now we form a directed graph \( D \) as follows. The nodes or vertices of the graph are the variables of the grammar. There is a directed edge from A to B whenever the grammar has a rule with A on the left hand side and B on the right hand side. A directed path is a sequence of distinct nodes \((v_1, \ldots, v_k)\) so that there is a directed edge from \( v_i \) to \( v_{i+1} \) for each \( 1 \leq i < k \). There is an algorithm for deciding whether there is a directed path from one node to another; use breadth-first search, for example. We remove each node \( v \) for which there is no path from \( S \) (the node for the start variable) to \( v \). Call the resulting directed graph \( D' \).

When there is a directed path from node \( x \) to node \( y \) and a directed edge from node \( y \) to node \( x \), they form a directed cycle. We claim that the language of \( G \) is infinite if and only if \( D' \) contains a directed cycle (which can be decided using breadth-first search). If there is no cycle in \( D' \), the language is finite because the number of possible parse trees is finite – no variable can repeat on any root to leaf path in a parse tree. If there is a cycle in \( D' \), suppose that \( V \) is a node on the cycle. There must be a path from \( S \) to \( V \) and therefore in the grammar \( S \rightarrow^* uVz \); if \( u \) or \( z \) contains variables then each can be replaced by a string of terminals, because any variable that could not be was deleted earlier. There must be a cycle involving \( V \) by hypothesis, and therefore in the grammar \( V \rightarrow^* vVy \); if \( v \) or \( y \) contains variables then each can be replaced by a string of terminals, because any variable that could not be was deleted earlier. Also it cannot be the case that both \( v \) and \( y \) are empty, because the grammar is in Chomsky normal form. Finally, because
V is a node in the graph it must be that $V \Rightarrow^* x$ for some string $x$ of terminals (or $V$ would have been deleted earlier).

Now let’s put the pieces together. Because $S \Rightarrow^* uVz$ and $V \Rightarrow^* vVy$, we have $S \Rightarrow^* uv^iVy^iZ$ for all $i \geq 0$. But since $V \Rightarrow^* x$, we have $S \Rightarrow^* uv^ixy^iZ$ for all $i \geq 0$. So the language contains infinitely many strings.

Using the directed graph idea, we design the TM T as follows:

$T = \text{“On input <M>, where M is a PDA:}$$$
1. Convert M to equivalent CFG G in Chomsky Normal Form.
2. Eliminate all variables that cannot generate a string consisting only of terminals, along with all rules that contain them.
3. Eliminate all variables that cannot be reached from the start variable, along with all rules that contain them.
4. Form the directed graph D’ with the remaining variables as nodes.
5. Using breadth-first search, determine if D’ contains a directed cycle. If it does, accept. If it does not, reject.”

I did not expect people to use this way of expressing the solution. It could also be done without resorting to directed graphs by just considering all parse trees of height $n$ where $n$ is the number of variables in the CFG in Chomsky normal form that generate at least one string that contains only terminals. In fact you don’t even need to make the whole parse tree – if you can find any downward path from the root that has length $n$, some variable must repeat by the pigeonhole principle, and you can get the same “pieces” that we put together above.

All of these methods are very similar in spirit to the pumping lemma for context-free languages, which should give you a guide for how to attack them, particularly if you approach the problem using parse trees.

4.12 Solution:

Since $L(R)$ and $L(S)$ are both regular languages, and regular languages are closed under complementation and intersection, so $\overline{L(S)} \cap L(R)$ is also regular, so there would be a DFA A that can accept this language. It is easy to see that $L(R) \subseteq L(S)$ if and only if $\overline{L(S)} \cap L(R) = \emptyset$, so we use the TM T’ in Theorem 4.4 as follows:

$T’ = \text{“On input <R,S>, where R and S are regular expressions:}$$$
1. Construct a DFA A that can accept $\overline{L(S)} \cap L(R)$
2. Run TM T in Theorem 4.4 on input <A>.
3. If T accepts, accept; otherwise, reject.”

4.19 Solution:

We already proved in Homework 2(c), that if L is regular, the reversal of L, Rev(L), is also regular. It means we can construct a DFA N to accept Rev(L).

Observe that $< M > \in S$, when $L(M) = Rev(L(M))$. Hence, we design a TM R as follows:

R = “On input <M>, where M is a DFA:

1. Construct a DFA N to accept $Rev(L(M))$.
2. Run TM F in Theorem 4.5 on input <M, N>.
3. If F accepts, accept; otherwise, reject.”