1. Let $M_2$ be the Turing machine whose description is given in Example 3.7. Give the sequence of configurations that $M_1$ enters when started on the following input strings.

(a) 0000.

\[
\begin{array}{llllll}
q_10000 & \sqcup xq_3 & \sqcup xq_20x & \sqcup q_5xxx & \sqcup xxxq_2 \\
\sqcup q_2000 & \sqcup xq_50x & \sqcup xxq_3x & q_5q_xe & \sqcup xxx & \sqcup q_{accept} \\
\sqcup xq_300 & \sqcup q_5x0x & \sqcup xxq_3x & \sqcup q_2xxx \\
\sqcup x0q_40 & q_5q_xe & \sqcup xxq_5x & \sqcup xq_2xx \\
\sqcup x0q_3 & \sqcup q_2x0x & \sqcup xxq_5x & \sqcup xxq_2x \\
\end{array}
\]

(b) 00000.

\[
\begin{array}{ll}
q_100000 & \sqcup x0q_3 \\
\sqcup q_20000 & \sqcup x0xq_4 \\
\sqcup x0q_300 & \sqcup x0x0 \sqcup q_{reject} \\
\sqcup x0q_40 \\
\end{array}
\]

2. Exercise 3.7

The description is not legitimate because there are infinitely many possible settings of the variables $x_1, \ldots, x_k$ to integral values, so a Turing machine cannot test them all and reject if none of them are roots. (It is legitimate for a Turing machine to test the possible integral values for $x_1, \ldots, x_k$ and accept if any of them are roots of $p$, but if $p$ has no integral roots, the Turing machine will loop, not reject.)

3. Give an implementation-level description of a Turing machine that decides the language

\[\{0^m1^p2^r | m > p \text{ and } r > p\}\].

Solution:

$M = \text{“On input string } w:”$

1. Scan the input, and reject if it contains a 0 after a 1 or a 0 after a 2 or a 1 after a 2.

2. Repeat the following as long as there are 1’s left on the tape:
   3. Scan the input. If there are no 0’s left or no 2’s left, reject. Otherwise, cross off one 0, one 1 and one 2.
   4. Scan the input. If there are at least one 0 left and at least one 2 left, accept, else reject.”
Note: The problem does not state if the Turing machine can have more than one tape. The solution given assumes one tape, but a more efficient solution can be given with three tapes.

4. Problem 3.9

(a) This part follows from Part (b), but we give a separate proof. We know that 1-PDAs are just PDAs, so since $A = \{0^n1^n2^n|n \geq 0\}$ is not a CFL, there is no 1-PDA that recognizes $A$. We will describe a 2-PDA $M$ that recognizes $A$. This will show 2-PDAs are more powerful than 1-PDAs.

$M$ will stop in a nonaccept state if it sees a 0 after it sees 1s or 2s or if it sees a 1 after it sees 2s. Initially, $M$ marks the bottom of both its stacks. While it is reading 0s, it pushes the 0s onto both of its stacks. While it reads 1s, it matches the 1s against the 0s on the first stack. If there are more 0s than 1s, it stops in a nonaccept state. Once the first stack is empty, $M$ does not read any more 1s, but instead matches 2s from the tape with the 0s on the second stack. When the second stack is empty, $M$ goes into an accept state and doesn’t read any more symbols.

(b) First, we argue that any 3-PDA can be simulated by a TM. Let $M$ be a 3-PDA. A nondeterministic TM $M'$ to simulate $M$ will have 4 tapes. The first tape will hold the input and the other three tapes will hold the three stacks of $M$, one per tape, with the bottom of the stack at the left end of the tape. $M'$ nondeterministically simulates one possible computation of $M$ on its input step-by-step, simulating the changes to the stack that $M$ makes by writing on its tape. If this computation of $M$ reaches an accepting state after reading its whole input, then $M'$ accepts. If the computation of $M$ gets stuck before reading the whole input or halts in a nonaccepting state after reading the whole input and without reaching an accepting state after reading the whole input, then $M'$ rejects. If the computation of $M$ goes into an infinite sequence of ε moves, then $M'$ loops.

Now, we argue that every TM can be simulated by a 2-PDA. Let $M$ be a TM. A 2-PDA $P$ to simulate $M$ works as follows. $P$ first marks the bottom of its two stacks. Then it reads its input, pushing it all onto its first stack. Then $P$ pops all but the first symbol from the first stack and pushes these symbols onto the second stack. At this point, $P$ has the first symbol of the input on its first stack and all the rest of the input, in left-to-right order, on its second stack, with the right end of the input at the bottom of the stack.

$P$ now simulates $M$ one step at a time, using moves that do not read any symbols from $P$’s tape. The current symbol of $M$ is always at the top of the first stack. If $M$ moves right, then $P$ replaces the top of the first stack with the symbol written by $M$ and pops the top symbol off the second stack and pushes it onto the first stack. If the
marker is at the top of the second stack, then $P$ leaves this symbol alone instead of popping it and pushes a blank onto the first stack. If $M$ moves left, then $P$ pops a symbol off the first stack and pushes onto the second stack the symbol that $M$ writes in its move. If this brings the marker to the top of the first stack, then this means that $M$ has tried to move left from the left end of the tape, so $P$ pops the symbol it just put onto the second stack and puts it back on the first stack.

If $M$ reaches its accept state, then $P$ goes into an accept state. If $M$ goes into its reject state, then $P$ stops simulating $M$, but does not go into an accept state. If $M$ loops, then $P$ has an infinite sequence of $\varepsilon$ moves after reading its whole input. $M$ and $P$ recognize the same language.

Now we have

- Every 2-PDA can be simulated by a 3-PDA. (Since a 3-PDA can just ignore its third stack.)
- Every 3-PDA can be simulated by a TM.
- Every TM can be simulated by a 2-PDA.

Thus, 2-PDAs, 3-PDAs and TMs are all equally powerful.

5. Problem 3.11

We must show that every ordinary Turing machine can be simulated by a Turing machine with doubly infinite tape and that every Turing machine with doubly infinite tape can be simulated by an ordinary Turing machine.

First suppose that $M$ is an ordinary Turing machine. We define a Turing machine $M'$ with doubly infinite tape to simulate $M$ as follows. $M'$ begins by moving left one cell and printing a special symbol $. Then, $M'$ moves right and goes into the start state of $M$. From then on, $M'$ just simulates $M$, except that whenever $M'$ reads $\$, it moves right and stays in the same state.

Now suppose that $N$ is a Turing machine with doubly infinite tape. We define an ordinary Turing machine $N'$ that simulates $N$. $N'$ will be a two-tape Turing machine. The book shows that a two-tape TM can be simulated by a one-tape TM. The first tape of $N'$ contains the cells from the first symbol of the original input to the right. The second tape contains the cells on $N$’s tape to the left of the original input, in reverse order. Initially, $N'$ transforms its input $w_1 \cdots w_n$ on the first tape into $\# w_1 \cdots w_n$ and puts a $\#$ in the first cell of tape 2. The tape head on tape one sits on $w_1$ and the tape head on tape two sits on $\#$. $N'$ then starts simulating moves of $N$ one at a time. At the start of each move simulation, exactly one of the two tape heads will be sitting on $\#$. If the first tape head is not on $\#$, then a move is simulated on tape 1. If after the move is simulated, the tape head on tape 1 reads $\#$, then the tape head on tape 2 is moved one cell to the right. If at the beginning of a move simulation the tape
6. A deterministic JFLAP Turing machine differs from our definition in several ways.

- A deterministic JFLAP TM can have more than one accept state.
  (No further moves are allowed once an accept state is reached.)
- A JFLAP TM does not have any reject states.
- A deterministic JFLAP TM can get stuck (i.e., reach a state and tape symbol combination where there is no next move even though the state is not an accept state).

For a JFLAP TM, an accepting computation is one that reaches an accept state, while a rejecting computation is one that gets stuck (in a non-accept state).

Show how to transform a deterministic JFLAP TM into an equivalent deterministic TM as we have defined this concept.

**Solution:**

Given a deterministic JFLAP TM $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$, we define $M'$ which is a TM according to our definition and recognizes the same language as $M$. We obtain $M'$ by adding two new states $q_a$ and $q_r$ to $M$. $q_a$ is the accept state of $M'$ and $q_r$ is the reject state of $M'$. For each accept state $q$ of $M$ and each tape symbol $a$, we add a transition $a \rightarrow R$ from $q$ to $q_a$. For each non-accept state $q$ of $M$ and each tape symbol $a$ such that $M$ gets stuck in state $q$ reading $a$ (which means that $M$ rejects the input), we add a transition $a \rightarrow R$ from $q$ to $q_r$.

7. (a) **Problem 3.15d**

Let $M$ be a Turing machine that decides a language $A$. Then a Turing machine $M'$ that decides the complement of the language $A$ is the same as $M$ except that the accepting and rejecting states are reversed. Thus, the complement of $A$ is decidable.

(b) **Problem 3.15e**

Let $M_1$ and $M_2$ be Turing machines that decide the languages $A_1$ and $A_2$, respectively. Then, a 2-tape TM $M$ that decides $A_1 \cap A_2$ works as follows.

$M =$ “On input string $w$

1. Copy $w$ onto tape 2.
2. Simulate $M_1$ on $w$ using tape 1 and $M_2$ on $w$ using tape 2.
3. If both $M_1$ and $M_2$ accept, then accept; else reject.”
$M$ is a decider since both $M_1$ and $M_2$ are deciders. If the input $w$ is in $A_1 \cap A_2$, then both $M_1$ and $M_2$ will accept $w$, so $M$ accepts $w$. If $w$ is not in $A_1 \cap A_2$, then either $M_1$ or $M_2$ (or both) will reject $w$, so $M$ rejects $w$. Thus $M$ decides $A_1 \cap A_2$ and $A_1 \cap A_2$ is decidable.

8. (a) **Problem 3.16b**

Let $M_1$ and $M_2$ be Turing machines that recognize the languages $A_1$ and $A_2$, respectively. Then, a nondeterministic, 3-tape TM $N$ that recognizes $A_1 \cap A_2$ works as follows.

$N = \text{"On input string } w$

1. Nondeterministically divide $w$ into $w = uv$. Copy $u$ onto tape 2 and $v$ onto tape 3.
2. Simulate $M_1$ on $u$ using tape 2. If $M_1$ rejects, then reject. If $M_1$ accepts, then go to Step 3.
3. Simulate $M_2$ on $v$ using tape 3. If $M_2$ accepts, then accept. If $M_2$ rejects, then reject.”

(Note that if either $M_1$ loops on $u$ or $M_2$ loops on $v$, then the given computation of $N$ loops.) If the input $w$ is in $A_1 \cap A_2$, then there will be some computation of $N$ on $w$ (the one that makes the right guess where to break up $w$ into $u$ and $v$) that accepts. If $w$ is not in $A_1 \cap A_2$, then all computations of $N$ on $w$ either reject or loop. Thus, $N$ recognizes $A_1 A_2$ and $A_1 A_2$ is Turing-recognizable.

(b) **Problem 3.16d**

Let $M_1$ and $M_2$ be Turing machines that recognize the languages $A_1$ and $A_2$, respectively. Then, a 2-tape TM $M$ that recognizes $A_1 \cap A_2$ works as follows.

$M = \text{"On input string } w$

1. Copy $w$ onto tape 2.
2. Simulate $M_1$ on $w$ using tape 1. If $M_1$ rejects, then reject. If $M_1$ accepts, then go to Step 3.
3. Simulate $M_2$ on $w$ using tape 2. If $M_2$ accepts, then accept. If $M_2$ rejects, then reject.”

(Note that if either $M_1$ or $M_2$ loops on $w$, then $M$ loops on $w.$) If the input $w$ is in $A_1 \cap A_2$, then both $M_1$ and $M_2$ will accept $w$, so $M$ accepts $w$. If $w$ is not in $A_1 \cap A_2$, then either $M_1$ or $M_2$ (or both) will reject or loop on $w$, so $M$ either rejects or loops on $w$. Thus $M$ recognizes $A_1 \cap A_2$ and $A_1 \cap A_2$ is Turing-recognizable.

9. If $A$ is a language, then $\text{PREFIX}(A)$ is the language

$$\{ u | uv \in A \text{ for some string } v \}$$

(a) Prove that if $A$ is decidable, then $\text{PREFIX}(A)$ is Turing recognizable.
(b) Prove that if $A$ is Turing recognizable, then $\text{PREFIX}(A)$ is Turing recognizable.
[Since every decidable language is Turing recognizable, this part implies the first part, but since the proof is harder, I made it a separate part.]

**Solution:**

a) Let $A$ be a language over an alphabet $\Sigma$ and let $M$ be a Turing machine that decides $A$. Since $\Sigma^*$ is countably infinite, we may list the strings in $\Sigma^*$ as $s_1, s_2, s_3, \ldots$. A Turing machine $N$ that recognizes $\text{PREFIX}(A)$ is given by

$$N = \text{"On input } w \text{"
1. For } i = 1, 2, 3, \ldots
2. \text{Run } M \text{ on } ws_i. \text{ If } M \text{ accepts, accept. If } M \text{ rejects, next } i."}$$

b) Now assume that $M$ only recognizes $A$. Then, $M$ may go into an infinite loop on some inputs, and we have to modify our definition of $N$ to take account of this fact. Instead of just running $M$ on $ws_i$ and waiting for $M$ to halt, which may not happen, $N$ runs $M$ in parallel on several inputs, for a fixed number of steps.

$$N = \text{"On input } w \text{"n}
1. For } i = 1, 2, 3, \ldots
2. \text{Run } M \text{ on } ws_1, ws_2, \ldots, ws_i \text{ for } i \text{ steps each. If } M \text{ accepts any of the strings in } i \text{ steps accept, else, next } i."}$$