1. Let $M_1$ be the Turing machine whose description is given in Example 3.9. Give the sequence of configurations that $M_1$ enters when started on the following input strings.

(a) 010#010.

\begin{align*}
q_1010#010 & \rightarrow xq_710\#x10 & x\#0q_5x0 & x\#xq_4x0 & x\#xq_1\#xx \\
xq_210#010 & \rightarrow xq_7x10\#x10 & xx0q_6\#xx0 & xxx\#xxq_40 & xxx\#xxq_8xx \\
x1q_20#010 & \rightarrow xq_110\#x10 & xxq_70\#xx0 & xxx\#xxq_6xx & xxx\#xxq_8xx \\
x10q_2\#010 & \rightarrow xxq_30\#x10 & xq_7x0\#xx0 & xxx\#xxq_6xx & xxx\#xxq_8x \\
x10q_4\#010 & \rightarrow xxq_10\#x10 & xxq_0\#xx0 & xxxxxq_6xx & xxxxxq_8 \\
x10q_6#x10 & \rightarrow xx0\#q_5x1 & xxxq_2\#xx0 & xxq_7\#xxx & xxxxx \sqcup q_{accept} \\
x1q_0\#x10 & \rightarrow xx0\#xq_510 & xxxxq_4\#xx0 & \\
(b) 010#01. 

\begin{align*}
q_1010#01 & \rightarrow x10q_6\#x1 & xxq_30\#x1 & xx0q_6\#xx & xxx\#xxq_4x \\
xq_10#01 & \rightarrow x1q_7\#x1 & xx0q_5\#x1 & xxq_70\#xx & xxxxxq_4x \\
x1q_0#01 & \rightarrow xxq_10\#x1 & xx0\#q_5x1 & xxq_70\#xx & xxxxxq_4 \\
x10q_2#01 & \rightarrow xxq_10\#x1 & xx0\#xq_51 & xxq_10\#xx & xxxxx \sqcup q_{reject} \\
x10q_4#01 & \rightarrow xxq_10\#x1 & xx0\#q_6xx & xxxq_2\#xx \\
(c) 01#01. 

\begin{align*}
q_101#01 & \rightarrow x1q_6\#x1 & xxq_3\#x1 & xx0q_6\#xx & xx\#xxq_5x \\
xq_21#01 & \rightarrow xq_7\#x1 & xxq_5\#x1 & xxq_7\#xx & xx\#xxq_5x \\
x1q_2#01 & \rightarrow q_7x1\#x1 & xx\#xxq_51 & xxq_1\#xx & xx\#xxq_5 \sqcup q_{reject} \\
x1#q_401 & \rightarrow q_1x1\#x1 & xx\#q_5xx & xx\#q_8xx \\

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. (a) Give an implementation-level description of a one-tape Turing machine that decides the language

$$\{w_1cw_2cw_3|w_1, w_2, w_3 \in \{a, b\}^*\}.$$ 

(b) Give a more efficient two-tape Turing machine to decide the language from Part (a).
Solution: This problem doesn’t make sense because the given language is regular, so it can be decided by a one-tape Turing machine with the instructions “On input \( w \), scan the input and accept if the input contains exactly two \( c \)'s and otherwise reject.” I had intended to ask you to give Turing machines to decide this language:
\[
\{w_1cw_2cw_3|w_1, w_2, w_3 \in \{a, b\}^* \text{ and } w_1 = w_3\}.
\]
Here is the solution to the problem I meant to ask.

(a) \( M \) = “On input string \( w \):

1. Scan the input, and reject if it does not contain exactly two \( c \)'s.
2. Repeat the following as long as there are uncrossed off symbols to the left of the first \( c \):
   3. Scan the input. Cross off the first uncrossed off symbol and remember it. Move right to the first uncrossed off symbol to the right of the second \( c \) (which could be a blank). If the remembered symbol does not match the uncrossed of symbol, reject, else cross off the uncrossed off symbol and return to the left end of the tape.
4. Scan the input. If there are no uncrossed off symbols to the right of the second \( c \) accept, else reject.”

(b) We define a two-tape Turing machine \( N \) to decide the same language.

\( N \) = “On input string \( w \):

1. Scan the input, and reject if it does not contain exactly two \( c \)'s.
2. Scan the input up to the first \( c \) and copy that part to tape 2.
3. Move right to the second \( c \) on tape 1 and move left to the first cell on tape 2.
4. Move right on both tapes. If all symbols match and a blank is reached on both tapes at the same time, accept, else reject.

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 $\varepsilon$ 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 head on tape 2 is not reading $\#$, then the move is simulated on tape 2, but a left move for $N$ causes the tape head to move right on the second tape of $N'$ and a right move for $N$ causes $N'$ to move left on tape 2.

6. A *write-on-right-move-only* Turing machine is a Turing machine that has to write the same symbol that it is reading whenever it moves left. A new symbol can be written only when the Turing machine moves right. Show that any ordinary Turing machine can be simulated by a write-on-right-move-only Turing machine.

**Solution:**

Given an ordinary TM $M$, we define an equivalent write-on-right-move-only TM $M'$ as follows. Starting with $M$, each right move remains unchanged as does each left move which prints the same symbol as the one being read, but each left move in $M$ that prints a different symbol than the one being read is replaced with three moves in $M'$. If the left move in $M$ goes from state $p$ to state $q$ reading an $a$ and printing a $b$, then in $M'$, two new states $r_1$ and $r_2$ are added. In state $p$ reading an $a$, $M'$ prints a $b$, moves right and goes to state $r_1$. In state $r_1$, reading any symbol $x$, $M'$
prints $x$, moves left, and goes to state $r_2$. In state $r_2$, reading any symbol $x$, $M'$ prints $x$, moves left and goes to state $q$.

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

Let $M_1$ and $M_2$ be Turing machines that decide the languages $A_1$ and $A_2$, respectively. Then, a nondeterministic, 3-tape TM $N$ that decides $A_1A_2$ works as follows.

$N =$ “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 and $M_2$ on $v$ using tape 3.
3. If both $M_1$ and $M_2$ accept, then accept; else reject.”

$N$ is a decider since both $M_1$ and $M_2$ are deciders. If the input $w$ is in $A_1A_2$, then there will be some computation of $N$ (the one that makes the right guess where to break up $w$ into $u$ and $v$) that accepts. If $w$ is not in $A_1A_2$, then all computations of $N$ reject. Thus, $N$ decides $A_1A_2$ and $A_1A_2$ is decidable.

(b) **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.

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

Let $M$ be a (deterministic) Turing machine that recognizes the language $A$. We describe a nondeterministic Turing machine $N$ that recognizes $A^*$. By Corollary 3.19, this shows that $A^*$ is Turing recognizable.

$N =$ “On input $w$

1. If $w = \varepsilon$, then accept.
2. If $w \neq \varepsilon$, then guess a sequence $w_1, \ldots, w_k$ of nonnull strings such that $w = w_1 \cdots w_k$.
3. Set $i = 1$.
4. Run $M$ on $w_i$. If $M$ rejects, then reject. If $M$ accepts and $i = k$, then accept. If $M$ accepts and $i < k$, then increment $i$ by 1 and repeat step 4.

If $w \in A^*$, then there is some computation of $N$ that guesses correctly and accepts. If $w \notin A^*$, then every computation of $N$ on $w$ either rejects or loops. Thus, $N$ recognizes $A^*$.

(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.

$M =$ “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.