1. (20 points). Let \( L \) be any language, let \( L_R \) be the set of reversals of strings in \( L \) so that
\[
L_R = \{ x \mid \text{for some } y \in L, |y| = |x| = n \text{ and } x_1x_2 \ldots x_n = y_ny_{n-1} \ldots y_1 \}
\]
Show that, if \( L \) is regular, so is \( L_R \).

**Solution.**

**Intuition**

Consider a language \( L \) that is represented using a DFA \( M \). In order to show that \( L_R \) is regular, it is enough to construct a NFA that accepts \( L_R \). Given any string that is accepted by the DFA, we want our new NFA to accept the reverse of this string. The key idea is the following: we know that any string accepted by the DFA would start at the initial state and end up in an accepting state. If we look at this path in reverse, i.e. by first starting at the accept state and ending up at the initial state, we are essentially parsing the input in reverse. Therefore, in this reversed path, if we can make our original start state as an accept state and original accept state as the start state, we would be accepting the reversed string.

Therefore, let’s construct the NFA by reversing all the transitions of \( M \) and swapping the initial states and final states. Since this may yield multiple initial states, for the sake of convenience, let’s introduce a new initial state which has \( \varepsilon \)-transitions to each of these initial states. On any input symbol, our new initial state just goes to a dead state and hence doesn’t affect the computation in any way.

**Construction**

Let \( L \) be a regular language and \( M = (Q, \Sigma, \delta, q_0, F) \) be a DFA for \( L \). Then consider the NFA \( M^R = (Q \cup \{q_0^R\}, \Sigma, \delta^R, q_0^R, \{q_0\}) \) where
\[
\delta^R(q, a) = \begin{cases} 
F, & \text{if } q = q_0^R \text{ and } a = \varepsilon \\
\emptyset, & \text{if } q = q_0^R \text{ and } a \neq \varepsilon \\
\{p \in Q \mid \delta(p, a) = q\}, & \text{otherwise}
\end{cases}
\]

Let us now show why this works. Say \( w \) is accepted by \( M \). Then \( M \) starts from \( q_0 \), follows some state sequence and ends in a final state \( q_f \). In \( M^R \), if we read \( w \) backwards (i.e. read \( w^R \)), then an accepting run would jump first to \( q_f \) (by performing an \( \varepsilon \)-transition from \( q_0^R \)), then follow the same state sequence backwards and wind up in \( q_0 \) (which is \( M^R \)'s accepting state). Thus if \( w \in L(M) \) then \( w^R \in L(M^R) \). We can use the same argument to show that if \( w \in L(M^R) \) then \( w^R \in L(M) \). Hence, this proves that \( L(M^R) = L_R \).
2. **(40 points)**. Let \( L \) be any language, let \( L_{\frac{1}{2}} \) be the set of all the first halves of strings in \( L \) so that
\[
L_{\frac{1}{2}} = \{ x \mid \text{for some } y, |y| = |x|, xy \in L \}
\]
For any regular language \( L \) show that \( L_{\frac{1}{2}} \) is regular. (*Hint: Think about the way we implemented two machines in “parallel” by using the Cartesian product. This idea maybe useful for this problem.*)

**Solution.**

**Intuition**

Let \( L \) be a regular language represented using a DFA \( M \). Consider any input \( x \). Let’s denote it’s length by \( n \). Our goal is to check if \( x \) is part of the language \( L_{\frac{1}{2}} \). Suppose at each step \( i \), we keep track of the state \( q \) that \( M \) is in and also the set \( S_i \) of states that can reach an accepting state in exactly \( i \) transitions. That is, at step 1, after processing the first symbol, we keep track of the current state of the machine and the set of states that can reach an accepting state in exactly one transition. Similarly, after seeing 2 input symbols (which denotes step 2), we keep track of the current state of the machine and the set of states that can reach an accepting state in exactly two transitions. Therefore, if the input finishes after \( n \) steps, then we can accept if \( M \)’s state appears in \( S_n \) because that would mean that the machine \( M \) is currently at a state from which we can reach an accepting state using \( n \) transitions. That means that, there exists a string \( y \) such that the length of \( y \) is equal to the length of \( x \) (which is equal to \( n \)) and the string \( xy \) is accepted by machine \( M \). How do we capture all this information? Suppose after \( i \) steps, we are in state \( q \) and have knowledge of \( S_i \). Then when the next input character \( c \) comes in, we advance the state of \( M \) to \( \delta(q, c) \) accordingly and find \( S_{i+1} \) by taking \( S_i \) and following transitions backwards.

Keeping track of the state of \( M \) is easy, but let us find a simple way to keep track of \( S_i \). For this, we use \( M_R \) which was the machine from Problem 2 that accepts the reverse of all strings accepted by \( M \). Remember that \( M_R \) is a NFA. However, instead of executing \( M_R \) on each symbol, we make every non-\( \varepsilon \)-move of \( M_R \) over *any* possible alphabet symbol (call this new machine \( \hat{M}_R \)). Then if we feed \( i \) characters into \( \hat{M}_R \), then the set of possible states it could be in is the set of states from which we can reach an accepting state of \( M \) in exactly \( i \) steps and this is precisely \( S_i \)!

Now, we use a cross-product construction to simultaneously run \( M \) and \( \hat{M}_R \).

**Construction**

Let’s define NFA \( N \) with state set \( Q \times 2^Q \) (here, we ignore \( q_0^R \) from \( M_R \) and just assume that \( M_R \) had multiple start states). The initial state of \( N \) will be \((q_0, F)\) since \( M \) starts in \( q_0 \) and \( \hat{M}_R \) starts in \( F \). When we are in state \((q, S)\) and we read input symbol \( c \), we move to state \((\delta(q, c), \{ p \mid \delta(p, c') \in S, c' \in \Sigma \}) \). Now, our accepting states are \( \{(q, S) \mid q \in S \} \).

Let’s see why this construction works. Consider a string \( x \in L_{\frac{1}{2}} \) and let \( y \) be its second half such that \( xy \in L \). After processing input string \( x \), if we are in state \((q, S)\) it means that \( S \) contains the set of states from which we can reach an accept state of \( M \) in \( |x| \) steps. Since we know that \( xy \in L \) and that \(|y| = |x|\), it means that \( q \) must be
a part of the set $S$. Thus, $N$ indeed accepts $x$. Similarly, we can show that any string accepted by $N$ lies in $L_{\frac{1}{2}}$. Hence, this proves that $L_{\frac{1}{2}}$ is regular.
3. (40 points). A DFA $M$ reads its input $x$ once from left to right. What if $M$ can read $x$ again? That is, $M$ reads $x$ from left to right then goes back to the start and reads $x$ from left to right again. Call this a two-pass DFA. Does re-reading the input help a DFA overcome its limited memory? Show that any language accepted by a two-pass DFA is also accepted by a normal DFA.

Formally, a two-pass DFA $M_{2p}$ for a language $L \subseteq \Sigma^*$ can be thought of as a normal DFA $M$ over the alphabet $\Sigma \cup \{\$\}$, where $\$ is a special symbol that represents reaching the end of the first-pass of the input. We say that a two-pass DFA accepts a string $x \in \Sigma^*$ if when viewed as a normal DFA over alphabet $\Sigma \cup \{\$\}$, it accepts the string $x\$x$. Therefore,

$$L(M_{2p}) = \{x \in \Sigma^* | M \text{ accepts } x\$x\}$$

In other words, show that for any two-pass DFA $M_{2p}$, $L(M_{2p})$ is regular.

Note that a two-pass DFA is trivially as strong as a normal DFA. This is because a two-pass DFA can "ignore" one reading of the input.

Hint: Think about the state that the two-pass DFA is in when it reads the $\$ symbol.

Solution.

Let $M_{2p}$ be a 2-pass DFA. $M_{2p}$ is represented as a DFA $M = (Q, \Sigma \cup \{\$\}, \delta, q_1, F)$ and $L(M_{2p})$ consists of all inputs $x \in \Sigma^*$ such that $x\$x$ is accepted by $M$.

Our goal is to construct a DFA $M' = (Q', \Sigma, \delta', q', F')$ such that $L(M_{2p}) = \{x | x\$x \in L(M)\} = L(M')$.

Intuition.

(a) Suppose we know the state in $M$ that the machine ends up after parsing the $x\$ part. Let's say that state is $q_x$ (independent of input). In that case it is straightforward to build $M'$. We will let $M'$ be the machine obtained from $M$ where the starting state is set to be $q_x$. Then, if we accept on input $x$, that means that $x\$x$ would have been accepted by $M$.

(b) The problem is that we do not have such a state $q_x$. We can resolve this issue by running $|Q|$ machines in parallel. Let us say that that states in $M$ are labeled by $\{q_1, ..., q_{|Q|}\}$ where $q_1$ is the starting state. Now in each parallel repetition $i \in [|Q|]$, we can treat $q_i$ as the starting state.

(c) We will also have one more machine that take $x$ and will try to find that state that you land up in after running $x\$. Then we can stitch these two parts. One part guesses where in $M$ the machine lands up in after parsing $x\$, say it is $q_i$. The second part simulates the path where each state is treated as the starting
Let Machine $M$ on input string $x$, if we are in state $q$, let’s see why this construction works. Consider a string $x$ if $M$ started at $q$, $M$ goes from the starting state to $q_{accept}$ when $M$ would go from the starting state to $q$ set of accept states. Formally, for $0 \leq i \leq n$, alphabet $\Sigma$, and changing the set of accept states to a single state for each new DFA.

Let $M'$ be a two-pass DFA and let $M = (Q, \Sigma \cup \{\$\}, \delta, q_0, F)$. Let $|Q| = n$. Define, Machine $M' = (Q', \Sigma, \delta', q_1', F')$.

- $Q' = (Q_1 \times Q_2 \times \ldots \times Q_n, (Q_{n+1} \times Q_{n+2}))$. Each $Q_i = Q$. Roughly, each $Q_i$ keeps track of the state in the machine where the state $q_i \in Q$ is the starting state for $i \in [n]$. $Q_{n+1}$ keeps track of the current state after the first pass of $x$. $Q_{n+2}$ keeps track of the state reachable by a dollar sign from $Q_{n+1}$’s state.
- We define starting state $q_{start}' = (q_1, \ldots, q_n, (q_1, \delta(q_1, \$)))$ where $q_i$ through $q_n$ are all the states of $Q$.
- We define $\delta'$. On input $(q_1', \ldots, q_n', (q_{n+1}', q_{n+2}'))$ where each $q_i'$ is in $Q$, and a character $a \in \Sigma$, do the following.
  Output $(\delta(q_1', a), \ldots, \delta(q_n', a), (\delta(q_{n+1}', a), \delta(\delta(q_{n+1}', a), \$)))$
- Set $F'$ as the states $(q_1', \ldots, q_n', (q_{n+1}', q_{n+2}'))$ with each $q_i' \in Q$ such that $q_{n+2}' = q_i$ for some $i \in [n]$, and $q_i' \in F$.

Let’s see why this construction works. Consider a string $x \in L(M')$. After processing input string $x$, if we are in state $(q_1', \ldots, q_n', (q_{n+1}', q_{n+2}'))$, then this means that on input $x$, $M$ goes from the starting state to $q_{n+1}'$, and can consume an additional $\$ to go from $q_{n+1}'$ to $q_{n+2}'$. Additionally, $q_i'$ for $i \in [n]$ is the state that $M$ would be in after input $x$ if $M$ started at $q_i$. Therefore, since we accept when $q_{n+2}' = q_i$ and $q_i' \in F$, then we accept when $M$ would go from the starting state to $q_{n+1}'$ on $x$, from $q_{n+1}'$ to $q_{n+2}'$ on $\$, and from $q_{n+2}' = q_i$ to some state in $F$ on input $x$. Thus, $x\$x \in L(M)$. Similarly, if $x\$x \in L(M)$, then this means that there is some path from the starting state to some state $q_i$ on $x$, a path from $q_i$ to $q_j$ on $\$, and a path from $q_j$ to an accepting state on $x$. Therefore, in $M'$, after input $x$, $q_{n+1}' = q_i$, $q_{n+2}' = q_j$, and $q_i'$ is an accepting state. Therefore, $M'$ will accept.

**Alternative Solution**

Let $M_{2p}$ be a two-pass DFA and let $M = (Q, \Sigma \cup \{\$\}, \delta, q_0, F)$ denote the machine $M_{2p}$ when viewed as a normal DFA. Let $n = |Q|$ and denote the states in $Q$ by $q_0, q_1, \ldots, q_{n-1}$. We can create $n$ DFA’s $M_0, \ldots, M_{n-1}$ by taking $M$, restricting it to the alphabet $\Sigma$, and changing the set of accept states to a single state for each new DFA. $M_0$ will be $M$ with the accept state changed to $q_0$, $M_1$ will have $q_1$ as the accept state, etc. Formally, for $0 \leq i \leq n-1$,

$$M_i = (Q, \Sigma, \delta, q_0, \{q_i\}),$$
where $\delta$ is the restriction of $\delta_s$ to input space $Q \times \Sigma$. It follows that the set of all strings $s \in \Sigma^*$ can be decomposed into $L(M_0) \cup L(M_1) \cup \ldots L(M_{n-1})$. Now, we have to consider the second reading of the string. We can now create $n$ DFA’s $N_0 \ldots N_{n-1}$ by taking $M$, restricting it to the alphabet $\Sigma$, and choosing the state reached after reading $\$\$ from each of the $n$ states to be the start state. That is, $N_0$ is the DFA with $\delta_s(q_0, \$\$) as the start state, $N_1$ is the DFA with $\delta_s(q_1, \$\$) as the start state, and so on. Formally, for $0 \leq i \leq n - 1$,

$$N_i = (Q, \Sigma, \delta, \delta_s(q_i, \$\$), F).$$

Define the language

$$L_{\text{reg}} = \bigcup_{0 \leq i \leq n-1} (L(M_i) \cap L(N_i)).$$

Since $L_{\text{reg}}$ is obtained by a finite unions and intersections of regular languages, it is regular. We now claim that

$$L(M_{2p}) = L_{\text{reg}}.$$  

To see this, suppose that $x \in L(M_{2p})$. Then, $M$ accepts $x\$\$x$. Let $q_i$ denote the state that $M$ is in after reading $x$. Then, $x \in L(M_i)$. Furthermore, $x \in L(N_i)$ since $M$ accepts $x\$\$x$, so it accepts the computation $x$ when starting from the state it reaches after reading $x\$\$. Therefore, $x \in L(M_i) \cap L(N_i)$ and therefore $x \in L_{\text{reg}}$. Conversely, suppose that $x \in L_{\text{reg}}$. Then, there exists some $i$ such that $x \in L(M_i) \cap L(N_i)$. Since $x \in L(M_i)$, $M$ is in state $q_i$ after reading $x$. Since $x \in L(N_i)$, $M$ accepts after reading $x$ when starting from $\delta_s(q_i, \$\$)$. Therefore, it follows that $M$ accepts $x\$\$x$ since after reading $x$, it is in state $q_i$, then after reading $\$, it is in state $\delta_s(q_i, \$\$)$, and then after reading the second $x$, it is in an accept state. Therefore, it follows that

$$L(M_{2p}) = L_{\text{reg}}$$

and is therefore regular.