Let $L$ be an arbitrary regular language. Prove that the language $\text{reverse}(L) := \{w^R \mid w \in L\}$ is regular. 

**Hint:** Consider a DFA $M$ that accepts $L$ and construct a NFA that accepts $\text{reverse}(L)$.

**Solution:**

Let $M = (\Sigma, Q, s, A, \delta)$ be a DFA that accepts $L$. We construct an NFA $M' = (\Sigma', Q', s', A', \delta')$ that accepts $\text{reverse}(L)$ as follows.

\[
\begin{align*}
Q' &:= Q \cup \{t\} \quad (\text{here } t \text{ is a new state not in } Q) \\
s' &:= t \\
A' &:= \{s\} \\
\delta'(t, \epsilon) &= A \\
\forall q \in Q, a \in \Sigma \quad \delta'(q, a) &= \{q' \in Q \mid \delta(q', a) = q\}
\end{align*}
\]

$M'$ is obtained from $M$ by reversing all the directions of the edges, adding a new state $t$ that becomes the new start state that is connected via $\epsilon$ edges to all the original accepting states. There is a single accepting state in $M'$ which is the start state of $M$. To see that $M'$ accepts $\text{reverse}(L)$ you need to see that any accepting walk of $M'$ corresponds to an accepting walk of $M$.

Another way to show that $\text{reverse}(L)$ is regular is via regular expressions. For any regular expression $r$ you can construct a regular expression $r'$ such that $L(r') = \text{reverse}(L)$ using the inductive definition of regular languages. We ignore the base cases as exercise and consider the inductive cases.

- If $r_1$ and $r_2$ are regular expressions and $r'_1$ and $r'_2$ are regular expressions for the reverse languages then the reverse for $r_1 + r_2$ is $r'_1 + r'_2$.
- For $r_1 r_2$ we have $r'_2 r'_1$.
- For $(r_1)^*$ we have $(r'_1)^*$.

---

Let $L$ be an arbitrary regular language. Prove that the language $\text{insert1}(L) := \{xy \mid xy \in L\}$ is regular.

Intuitively, $\text{insert1}(L)$ is the set of all strings that can be obtained from strings in $L$ by inserting exactly one $1$. For example, if $L = \{\varepsilon, OOK!\}$, then $\text{insert1}(L) = \{1, 1OOK!, O1OK!, OO1K!, OOK1!, OOK!1, OOK1\}$. 

**Solution:**

Let $M = (\Sigma, Q, s, A, \delta)$ be a DFA that accepts $L$. We construct an NFA $M' = (\Sigma', Q', s', A', \delta')$ that accepts $\text{insert1}(L)$ as follows:

\[
\begin{align*}
Q' &:= Q \times \{\text{before, after}\} \\
s' &:= (s, \text{before}) \\
A' &:= \{(q, \text{after}) \mid q \in A\}
\end{align*}
\]
\[ \delta'(q, \text{before}, a) = \begin{cases} \{ (\delta(q, a), \text{before}), (q, \text{after}) \} & \text{if } a = 1 \\ \{ (\delta(q, a), \text{before}) \} & \text{otherwise} \end{cases} \]

\[ \delta'(q, \text{after}, a) = \{ (\delta(q, a), \text{after}) \} \]

M' nondeterministically chooses a 1 in the input string to ignore, and simulates M running on the rest of the input string.

- The state \((q, \text{before})\) means (the simulation of) M is in state q and M' has not yet skipped over a 1.
- The state \((q, \text{after})\) means (the simulation of) M is in state q and M' has already skipped over a 1.

---

**Solutions for extra problems**

3. Let \(L\) be an arbitrary regular language. Prove that the language \(\text{delete}_1(L) := \{ xy | x1y \in L \}\) is regular. Intuitively, \(\text{delete}_1(L)\) is the set of all strings that can be obtained from strings in \(L\) by deleting exactly one 1. For example, if \(L = \{101101, 00, \varepsilon\}\), then \(\text{delete}_1(L) = \{01101, 10101, 10110\}\).

**Solution:**

Let \(M = (\Sigma, Q, s, A, \delta)\) be a DFA that accepts \(L\). We construct an NFA \(M' = (\Sigma, Q', s', A', \delta')\) with \(\varepsilon\)-transitions that accepts \(\text{delete}_1(L)\) as follows:

\[ Q' := Q \times \{ \text{before, after} \} \]
\[ s' := (s, \text{before}) \]
\[ A' := \{ (q, \text{after}) \mid q \in A \} \]

\[ \delta'((q, \text{before}), \varepsilon) = \{ (\delta(q, 1), \text{after}) \} \]
\[ \delta'((q, \text{after}), \varepsilon) = \emptyset \]
\[ \delta'((q, \text{before}), a) = \{ (\delta(q, a), \text{before}) \} \]
\[ \delta'((q, \text{after}), a) = \{ (\delta(q, a), \text{after}) \} \]

\(M'\) simulates \(M\), but inserts a single 1 into \(M\)'s input string at a nondeterministically chosen location.

- The state \((q, \text{before})\) means (the simulation of) \(M\) is in state q and \(M'\) has not yet inserted a 1.
- The state \((q, \text{after})\) means (the simulation of) \(M\) is in state q and \(M'\) has already inserted a 1.
Consider the following recursively defined function on strings:

\[
\text{stutter}(w) := \begin{cases} 
\varepsilon & \text{if } w = \varepsilon \\
\alpha \cdot \text{stutter}(x) & \text{if } w = \alpha x \text{ for some symbol } \alpha \text{ and some string } x 
\end{cases}
\]

Intuitively, \( \text{stutter}(w) \) doubles every symbol in \( w \). For example:

- \( \text{stutter}('PRESTO') = 'PPREESSTTOO' \)
- \( \text{stutter}('HOCUS\square\text{POCUS}') = 'HHOCCUUSS\square\square\text{POOCCUUSS}' \)

Let \( L \) be an arbitrary regular language.

4.A. Prove that the language \( \text{stutter}^{-1}(L) := \{ w \mid \text{stutter}(w) \in L \} \) is regular.

**Solution:**
Let \( M = (\Sigma, Q, s, A, \delta) \) be a DFA that accepts \( L \).

We construct an DFA \( M' = (\Sigma', s', A', \delta') \) that accepts \( \text{stutter}^{-1}(L) \) as follows:

\[
\begin{align*}
Q' &= Q \\
s' &= s \\
A' &= A \\
\delta'(q, a) &= \delta(\delta(q, a), a)
\end{align*}
\]

\( M' \) reads its input string \( w \) and simulates \( M \) running on \( \text{stutter}(w) \). Each time \( M' \) reads a symbol, the simulation of \( M \) reads two copies of that symbol.

4.B. Prove that the language \( \text{stutter}(L) := \{ \text{stutter}(w) \mid w \in L \} \) is regular.

**Solution:**
Let \( M = (\Sigma, Q, s, A, \delta) \) be a DFA that accepts \( L \).

We construct an DFA \( M' = (\Sigma', s', A', \delta') \) that accepts \( \text{stutter}(L) \) as follows:

\[
\begin{align*}
Q' &= Q \times (\{\bullet\} \cup \Sigma) \cup \{\text{fail}\} \quad \text{for some } \bullet \notin \Sigma \\
s' &= (s, \bullet) \\
A' &= \{(q, \bullet)\} \quad q \in A \\
\delta'((q, \bullet), a) &= (q, a) \\
\delta'((q, a), b) &= \begin{cases} 
(\delta(q, a), \bullet) & \text{if } a = b \\
\text{fail} & \text{if } a \neq b
\end{cases} \\
\delta'(\text{fail}, a) &= \text{fail}
\end{align*}
\]

\( M' \) reads the input string \( \text{stutter}(w) \) and simulates \( M \) running on input \( w \).

- State \( (q, \bullet) \) means \( M' \) has just read an even symbol in \( \text{stutter}(w) \), so \( M \) should ignore the next symbol (if any).
- For any symbol \( a \in \Sigma \), state \( (q, a) \) means \( M' \) has just read an odd symbol in \( \text{stutter}(w) \), and that symbol was \( a \). If the next symbol is an \( a \), then \( M \) should transition normally; otherwise, the simulation should fail.
- The state \( \text{fail} \) means \( M' \) has read two successive symbols that should have been equal but were not; the input string is not \( \text{stutter}(w) \) for any string \( w \).
Solution:

Let $R$ be an arbitrary regular expression. We recursively construct a regular expression $\text{stutter}(R)$ as follows:

$$
\text{stutter}(R) := \begin{cases} 
\emptyset & \text{if } R = \emptyset \\
stutter(w) & \text{if } R = w \text{ for some string } w \in \Sigma^* \\
stutter(A) + \text{stutter}(B) & \text{if } R = A + B \text{ for some regular expressions } A \text{ and } B \\
stutter(A) \cdot \text{stutter}(B) & \text{if } R = AB \text{ for some regular expressions } A \text{ and } B \\
(\text{stutter}(A))^* & \text{if } R = A^* \text{ for some regular expression } A
\end{cases}
$$

To prove that $L(\text{stutter}(R)) = \text{stutter}(L(R))$, we need the following identities for arbitrary languages $A$ and $B$:

- $\text{stutter}(A \cup B) = \text{stutter}(A) \cup \text{stutter}(B)$
- $\text{stutter}(A \cdot B) = \text{stutter}(A) \cdot \text{stutter}(B)$
- $\text{stutter}(A^*) = \text{stutter}(A)^*$

These identities can all be proved by inductive definition-chasing, after which the claim $L(\text{stutter}(R)) = \text{stutter}(L(R))$ follows by induction. We leave the details of the induction proofs as an exercise for a future semester exam the reader.

Equivalently, we can directly transform $R$ into $\text{stutter}(R)$ by replacing every explicit string $w \in \Sigma^*$ inside $R$ with $\text{stutter}(w)$ (with additional parentheses if necessary). For example:

$$
\text{stutter}((1 + \varepsilon)(01)^*(0 + \varepsilon) + 0^*) = (11 + \varepsilon)(0011)^*(00 + \varepsilon) + (00)^*
$$

Although this may look simpler, actually proving that it works requires the same induction arguments.

---

5. Consider the following recursively defined function on strings:

$$
evens(w) := \begin{cases} 
\varepsilon & \text{if } w = \varepsilon \\
\varepsilon & \text{if } w = a \text{ for some symbol } a \\
b \cdot evens(x) & \text{if } w = abx \text{ for some symbols } a \text{ and } b \text{ and some string } x
\end{cases}
$$

Intuitively, $\text{evens}(w)$ skips over every other symbol in $w$. For example:

- $\text{evens}(\text{EXPELLIARMUS}) = \text{XELAMS}$
- $\text{evens}(\text{AVADA\Box KEDAVRA}) = \text{VD\Box EAR}$.

Once again, let $L$ be an arbitrary regular language.

5.A. Prove that the language $\text{evens}^{-1}(L) := \{w \mid \text{evens}(w) \in L\}$ is regular.

**Solution:**

Let $M = (\Sigma, Q, s, A, \delta)$ be a DFA that accepts $L$. We construct an DFA $M' = (\Sigma, Q', s', A', \delta')$ that
accepts $evens^{-1}(L)$ as follows:

$$Q' = Q \times \{0, 1\}$$
$$s' = (s, 0)$$
$$A' = A \times \{0, 1\}$$
$$\delta'(q, 0, a) = (q, 1)$$
$$\delta'(q, 1, a) = (\delta(q, a), 0)$$

$M'$ reads its input string $w$ and simulates $M$ running on $evens(w)$.

- State $(q, 0)$ means $M'$ has just read an even symbol in $w$, so $M$ should ignore the next symbol (if any).
- State $(q, 1)$ means $M'$ has just read an odd symbol in $w$, so $M$ should read the next symbol (if any).

5.B. Prove that the language $evens(L) := \{evens(w) \mid w \in L\}$ is regular.

**Solution:**

Let $M = (\Sigma, Q, s, A, \delta)$ be a DFA that accepts $L$. We construct an NFA $M' = (\Sigma, Q', s', A', \delta')$ that accepts $evens(L)$ as follows:

$$Q' = Q$$
$$s' = s$$
$$A' = A \cup \{q \in Q \mid \delta(q, a) \cap A \neq \emptyset \text{ for some } a \in \Sigma\}$$
$$\delta'(q, a) = \bigcup_{b \in \Sigma} \{\delta(\delta(q, b), a)\}$$

$M'$ reads the input string $evens(w)$ and simulates $M$ running on string $w$, while nondeterministically guessing the missing symbols in $w$.

- When $M'$ reads the symbol $a$ from $evens(w)$, it guesses a symbol $b \in \Sigma$ and simulates $M$ reading $ba$ from $w$.
- When $M'$ finishes $evens(w)$, it guesses whether $w$ has even or odd length, and in the odd case, it guesses the last character of $w$. 