Let $L$ be an arbitrary regular language.

1. Prove that the language $\text{insert} 1(L) = \{ x1y \mid xy \in L \}$ is regular.

Intuitively, $\text{insert} 1(L)$ is the set of all strings that can be obtained from strings in $L$ by inserting exactly one $1$. For example, if $L = \{ \epsilon, \text{00K!} \}$, then $\text{insert} 1(L) = \{ 1, \text{100K!}, \text{010K!}, \text{001K!}, \text{00K1!}, \text{00K!1} \}$.

**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 $\text{insert} 1(L)$ as follows:

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

$M'$ 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 $1$. □
2. Prove that the language $\text{delete} 1(L) := \{xy \mid x1y \in L\}$ 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 $L = \{10101, 00, \varepsilon\}$, 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.

■
3. Consider the following recursively defined function on strings:

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

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

- \(stutter(\text{PRESTO}) = \text{PPRREESSTTOO}\)
- \(stutter(\text{HOCUS\POCUS}) = \text{HHOOCCUUSS\PPOOCCUUSS}\)

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

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

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

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

\[
Q' = Q \\
 s' = s \\
 A' = A \\
\delta'(q, a) = \delta(\delta(q, a), a)
\]

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

\[\blacksquare\]
(b) Prove that the language $stutter(L) := \{stutter(w) \mid w \in L\}$ is regular.

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

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

- $Q' = Q \times (\{\bullet\} \cup \Sigma) \cup \{\text{fail}\}$ for some $\bullet \not\in \Sigma$
- $s' = (s, \bullet)$
- $A' = \{(q, \bullet) \mid q \in A\}$
- $\delta'((q, \bullet), a) = (q, a)$
- $\delta'((q, a), b) = \begin{cases} (\delta(q, a), \bullet) & \text{if } a = b \\ \text{fail} & \text{if } a \neq b \end{cases}$
- $\delta'(\text{fail}, a) = \text{fail}$

$M'$ reads the input string $stutter(w)$ and simulates $M$ running on input $w$.

- State $(q, \bullet)$ means $M'$ has just read an even-indexed symbol in $stutter(w)$, 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-indexed symbol in $stutter(w)$, 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 $stutter(w)$ for any string $w$.

---

\[\text{The first symbol in the input string has index 1; the second symbol has index 2, and so on.}\]
Solution (via regular expressions): Let $R$ be an arbitrary regular expression. We recursively construct a regular expression $\text{stutter}(R)$ as follows:

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

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

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

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

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

$$
\text{stutter}\left((1 + \epsilon)(01)^*(\emptyset + \epsilon) + \emptyset^*\right) = (11 + \epsilon)(\emptyset011)^*(\emptyset0 + \epsilon) + (\emptyset0)^*
$$

Although this may look simpler, actually proving that it works requires the same induction arguments. ■
4. Consider the following recursively defined function on strings:

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

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

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

Once again, let \(L\) be an arbitrary regular language.

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

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

\[
\begin{align*}
Q' &= Q \\
s' &= s \\
A' &= A \cup \{ q \in Q \mid \delta(q, a) \cap A \neq \emptyset \text{ for some } a \in \Sigma \} \\
\delta'(q, a) &= \bigcup_{b \in \Sigma} \{ \delta(\delta(q, b), a) \}
\end{align*}
\]

\( M' \) 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 symbol in \( w \).