Let $L$ be an arbitrary regular language over the alphabet $\Sigma = \{0, 1\}$. Prove that the following languages are also regular. (You probably won’t get to all of these.)

1. $\text{flipOdds}(L) := \{\text{flipOdds}(w) \mid w \in L\}$, where the function $\text{flipOdds}$ inverts every odd-indexed bit in $w$. For example:

$$\text{flipOdds}(000111101010101) = 1010010111111111$$

**Solution:** Let $M = (Q, s, A, \delta)$ be a DFA that accepts $L$. We construct a new DFA $M’ = (Q’, s’, A’, \delta’)$ that accepts $\text{flipOdds}(L)$ as follows.

Intuitively, $M’$ receives some string $\text{flipOdds}(w)$ as input, restores every other bit to obtain $w$, and simulates $M$ on the restored string $w$.

Each state $(q, \text{flip})$ of $M’$ indicates that $M$ is in state $q$, and we need to flip the next input bit if $\text{flip} = \text{True}$.

- $Q’ = Q \times \{\text{True}, \text{False}\}$
- $s’ = (s, \text{True})$
- $A’ = A \times \{\text{True}, \text{False}\}$

$$\begin{align*}
\delta’((q, \text{False}), 0) &= (\delta(q, 0), \text{True}) \\
\delta’((q, \text{True}), 0) &= (\delta(q, 1), \text{False}) \\
\delta’((q, \text{False}), 1) &= (\delta(q, 1), \text{True}) \\
\delta’((q, \text{True}), 1) &= (\delta(q, 0), \text{False})
\end{align*}$$

By treating $1$ and $0$ as synonyms for $\text{True}$ and $\text{False}$, respectively, we can rewrite $\delta’$ more compactly as

$$\delta’((q, \text{flip}), a) = (\delta(q, a \oplus \text{flip}), \neg \text{flip})$$
2. \textsc{UNflipOdd1s}(L) := \{w \in \Sigma^* \mid \text{flipOdd1s}(w) \in L\}, where the function \textit{flipOdd1} inverts every other 1 bit of its input string, starting with the first 1. For example:

\[
\text{flipOdd1s}(0000111101010101) = 00001010001001
\]

\textbf{Solution:} Let \(M = (Q, s, A, \delta)\) be a DFA that accepts \(L\). We construct a new DFA \(M' = (Q', s', A', \delta')\) that accepts \textsc{UNflipOdd1s}(L) as follows.

Intuitively, \(M'\) receives some string \(w\) as input, flips every other 1 bit, and simulates \(M\) on the transformed string.

Each state \((q, \text{flip})\) of \(M'\) indicates that \(M\) is in state \(q\), and we need to flip the next 1 bit of and only if \(\text{flip} = \text{TRUE}\).

\[
\begin{align*}
Q' &= Q \times \{\text{TRUE}, \text{FALSE}\} \\
S' &= (s, \text{TRUE}) \\
A' &= A \times \{\text{TRUE}, \text{FALSE}\} \\
\delta'((q, \text{FALSE}), 0) &= (\delta(q, 0), \text{FALSE}) \\
\delta'((q, \text{TRUE}), 0) &= (\delta(q, 0), \text{TRUE}) \\
\delta'((q, \text{FALSE}), 1) &= (\delta(q, 1), \text{TRUE}) \\
\delta'((q, \text{TRUE}), 1) &= (\delta(q, 0), \text{FALSE})
\end{align*}
\]

Once again, by treating 1 and 0 as synonyms for \text{TRUE} and \text{FALSE}, respectively, we can rewrite \(\delta'\) more compactly as

\[
\delta'((q, \text{flip}), a) = (\delta(q, \neg \text{flip} \land a), \text{flip} \oplus a)
\]
3. \textsc{flipOdd1s}(L) := \{\text{flipOdd1s}(w) \mid w \in L\}, where the function \text{flipOdd1} is defined as in the previous problem.

Solution: Let \( M = (Q, s, A, \delta) \) be a DFA that accepts \( L \). We construct a new NFA \( M' = (Q', s', A', \delta') \) that accepts \text{flipOdd1s}(L) as follows.

Intuitively, \( M' \) receives some string \text{flipOdd1s}(w) as input, guesses which \( \emptyset \) bits to restore to \( 1s \), and simulates \( M \) on the restored string \( w \). No string in \text{flipOdd1s}(L) has two \( 1s \) in a row, so if \( M' \) ever sees \( 11 \), it rejects.

Each state \((q, \text{flip})\) of \( M' \) indicates that \( M \) is in state \( q \), and we need to flip a \( \emptyset \) bit before the next \( 1 \) bit if and only if \( \text{flip} = \text{True} \).

\[
Q' = Q \times \{\text{True, False}\}
\]
\[
s' = (s, \text{True})
\]
\[
A' = A \times \{\text{True, False}\}
\]
\[
\delta'((q, \text{False}), \emptyset) = \{\delta(q, \emptyset), \text{False}\}
\]
\[
\delta'((q, \text{True}), \emptyset) = \{\delta(q, \emptyset), \text{True}, \delta(q, 1), \text{False}\}
\]
\[
\delta'((q, \text{False}), 1) = \{\delta(q, 1), \text{True}\}
\]
\[
\delta'((q, \text{True}), 1) = \emptyset
\]

The last transition indicates that we waited too long to flip a \( \emptyset \) to a \( 1 \), so we should kill the current execution thread.

4. Prove that the language \( \text{insert1}(L) := \{x1y \mid x,y \in L\} \) 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 \( L = \{\epsilon, \text{OOK}!\} \), then \( \text{insert1}(L) = \{1, 100K!, 010K!, 001K!, 00K1!, \text{OOK}1!, \text{OOK}11\} \).

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

Intuitively, \( M' \) 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 \( 1 \).

\[
Q' := Q \times \{\text{before, after}\}
\]
\[
s' := (s, \text{before})
\]
\[
A' := \{(q, \text{after}) \mid q \in A\}
\]
\[
\delta'((q, \text{before}), a) = \begin{cases} 
\{\delta(q, a), \text{before}\}, (q, \text{after}) & \text{if } a = 1 \\
\{\delta(q, a), \text{before}\} & \text{otherwise}
\end{cases}
\]
\[
\delta'((q, \text{after}), a) = \{\delta(q, a), \text{after}\}
\]
5. 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 \( L = \{101101,00,\varepsilon \} \), then \( \text{delete}_1(L) = \{01101,10101,10110\} \).

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

Intuitively, \( M' \) 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.

\[
Q' := Q \times \{ \text{before, after} \} \\
S' := (s, \text{before}) \\
A' := \{(q, \text{after}) \mid q \in A \} \\
\delta'(q, \text{before}, \varepsilon) = \{(\delta(q, 1), \text{after})\} \\
\delta'(q, \text{after}, \varepsilon) = \emptyset \\
\delta'(q, \text{before}, a) = \{(\delta(q, a), \text{before})\} \\
\delta'(q, \text{after}, a) = \{(\delta(q, a), \text{after})\}
\]
Consider the following recursively defined function on strings:

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

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

- \(\text{stutter(PRESTO)} = \text{PPRREESSTT00}\)
- \(\text{stutter(HOCUS•POCUS)} = \text{HH00CUUSS••PP00CUUSS}\)

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

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

Intuitively, \(M'\) 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.

\[
\begin{align*}
Q' &= Q \\
s' &= s \\
A' &= A \\
\delta'(q, a) &= \delta(\delta(q, a), a)
\end{align*}
\]
(b) Prove that the language $stutter(L) := \{stutter(w) \mid w \in L\}$ is regular.

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

$M'$ 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 $fail$ 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 $w$.

$$ Q' = Q \times (\{\bullet\} \cup \Sigma) \cup \{fail\} \quad \text{for some new symbol } \bullet \notin \Sigma $$

$$ s' = (s,\bullet) $$

$$ A' = \{(q,\bullet) \mid q \in A\} $$

$$ \delta'((q,\bullet),a) = (q,a) \quad \text{for all } q \in Q \text{ and } a \in \Sigma $$

$$ \delta'((q,a),b) = \begin{cases} 
(q,a),\bullet 
& \text{if } a = b \\
fail 
& \text{if } a \neq b 
\end{cases} \quad \text{for all } q \in Q \text{ and } a, b \in \Sigma $$

$$ \delta'(fail,a) = fail \quad \text{for all } a \in \Sigma $$

---

$^a$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:

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

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 an 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:

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

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

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

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

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

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

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

**Solution:** Let \( M = (Q,s,A,\delta) \) be a DFA that accepts \( L \). We construct a DFA \( M' = (Q',s',A',\delta') \) that accepts \( \text{evens}^{-1}(L) \) 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 \( \text{evens}(w) \).

- 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 \( \text{evens}(L) := \{ \text{evens}(w) \mid w \in L \} \) is regular.

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

Intuitively, \( M' \) reads the input string \( \text{evens}(w) \) and simulates \( M \) running on string \( w \), while nondeterministically guessing the missing symbols in \( w \).

- When \( M' \) reads the symbol \( a \) from \( \text{evens}(w) \), it guesses a symbol \( b \in \Sigma \) and simulates \( M \) reading \( ba \) from \( w \).

- When \( M' \) finishes \( \text{evens}(w) \), it guesses whether \( w \) has even or odd length, and in the odd case, it guesses the last symbol in \( w \).

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