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 during the lab session.)

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:

$$\text{flipOdds}(00011110101000) = 101001011111110$$

**Solution:** Let $M = (Q, s, A, \delta)$ be an arbitrary 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 $\text{flip} = \text{TRUE}$.

- $Q’ = Q \times \{\text{TRUE}, \text{FALSE}\}$
- $s’ = (s, \text{TRUE})$
- $A’ = A \times \{\text{TRUE}, \text{FALSE}\}$
- $\delta’((q, \text{flip}), a) = (\delta(q, a \oplus \text{flip}), \neg\text{flip})$

Here I am treating 1 and 0 as synonyms for TRUE and FALSE, respectively, and $\oplus$ denotes exclusive-or.
2. \text{UNFLIPODD1s}(L) := \{w \in \Sigma^* \mid \text{flipOdd1s}(w) \in L\}$, where the function \text{flipOdd1} inverts every other 1 bit of its input string, starting with the first 1. For example:

\[
\text{flipOdd1s}(0000111010101) = 00000101001001
\]

**Solution:** Let $M = (Q, s, A, \delta)$ be an arbitrary DFA that accepts $L$. We construct a new DFA $M' = (Q', s', A', \delta')$ that accepts $\text{UNFLIPODD1s}(L)$ as follows.

Intuitively, $M'$ receives some string $w$ as input, flips every other 1 bit, and then 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 if and only if $\text{flip} = \text{True}$.

\[
Q' = Q \times \{\text{True, False}\}
\]
\[
s' = (s, \text{True})
\]
\[
A' = A \times \{\text{True, False}\}
\]
\[
\delta'((q, \text{flip}), a) = (\delta(q, \text{flip} \oplus a), \text{flip} \oplus a)
\]

Again, I am treating 1 and 0 as synonyms for True and False, respectively, and $\oplus$ denotes exclusive-or.
3. \( \text{FlipOdd}_1s(L) := \{ \text{flipOdd}_1s(w) \mid w \in L \} \), where the function \( \text{flipOdd}_1 \) 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 \( \text{FlipOdd}_1s(L) \) as follows.

Intuitively, \( M' \) receives some string \( \text{flipOdd}_1s(w) \) as input, guesses which 0 bits to restore to 1s, and simulates \( M \) on the restored string \( w \). No string in \( \text{FlipOdd}_1s(L) \) has two 1s in a row, so if \( M' \) ever sees 11, it must reject.

Each state \((q, \text{flip})\) of \( M' \) indicates that \( M \) is in state \( q \), and we need to flip some 0 bit before the next 1 bit if and only if \( \text{flip} = \text{True} \).

\[
Q' = Q \times \{ \text{True, False} \} \\
s' = (s, \text{True}) \\
A' = A \times \{ \text{True, False} \} \\
\delta'((q, \text{False}), 0) = \{ (\delta(q, 0), \text{False}) \} \\
\delta'((q, \text{True}), 0) = \{ (\delta(q, 0), \text{True}), (\delta(q, 1), \text{False}) \} \\
\delta'((q, \text{False}), 1) = \{ (\delta(q, 1), \text{True}) \} \\
\delta'((q, \text{True}), 1) = \emptyset
\]

The last transition indicates that we waited too long to flip a 0 to a 1, so we should kill the current execution thread. ■
4. \( \text{Faro}(L) := \{ \text{faro}(w, x) \mid w, x \in L \text{ and } |w| = |x| \} \), where the function \( \text{faro} \) is defined recursively as follows:

\[
\text{faro}(w, x) := \begin{cases} 
  x & \text{if } w = \epsilon \\
  a \cdot \text{faro}(x, y) & \text{if } w = ay \text{ for some } a \in \Sigma \text{ and some } y \in \Sigma^* 
\end{cases}
\]

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

Intuitively, \( M' \) reads the string \( \text{faro}(w, x) \) as input, splits the string into the subsequences \( w \) and \( x \), and passes those strings to independent copies of \( M \). Let \( M_1 \) denote the copy that processes the first string \( w \), and let \( M_2 \) denote the copy that processes the second string \( x \).

Each state \((q_1, q_2, \text{next})\) indicates that machine \( M_1 \) is in state \( q_1 \), machine \( M_2 \) is in state \( q_2 \), and \( \text{next} \) indicates whether \( M_1 \) or \( M_2 \) receives the next input bit. Because of the constraint \( |w| = |x| \), machine \( M' \) can accept only if \( \text{next} = 1 \).

\[
\begin{align*}
Q' &= Q \times Q \times \{1, 2\} \\
s' &= (s, s, 1) \\
A' &= \{(q_1, q_2, 1) \mid q_1, q_2 \in A\} \\
\delta'((q_1, q_2, \text{next}), a) &= \begin{cases} 
  (\delta(q_1, a), q_2, 2) & \text{if } \text{next} = 1 \\
  (q_1, \delta(q_2, a), 1) & \text{if } \text{next} = 2 
\end{cases}
\end{align*}
\]