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:

$\text{flipOdds}(0001111010100) = 10100101111110$

**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. \textbf{UnflipOdd}s(L) := \{w \in \Sigma^* \mid \text{flipOdd}s(w) \in L\}, where the function \text{flipOdd} inverts every other 1 bit of its input string, starting with the first 1. For example:

\[
\text{flipOdd}s(\text{0000111010101}) = \text{0000010100100100101}
\]

\textbf{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{UnflipOdd}s(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 \(\text{flip} = \text{True}\).

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

Again, I am treating 1 and 0 as synonyms for \text{True} and \text{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 $\text{flip} = \text{True}$.

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

$$\delta'((q,\text{False}),0) = \{\delta(q,0), \text{False}\}$$
$$\delta'((q,\text{True}),0) = \{\delta(q,0), \text{True}\}, (\delta(q,1), \text{False})\}$$
$$\delta'((q,\text{False}),1) = \{\delta(q,1), \text{True}\}$$
$$\delta'((q,\text{True}),1) = \emptyset$$

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. \textsc{Shuffle}(L) := \{\text{shuff}(w, x) \mid w, x \in L \text{ and } |w| = |x|\}$, where the function $\text{shuff}$ is defined recursively as follows:

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

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

Intuitively, $M'$ reads the string $\text{shuff}(w, x)$ as input, splits the string into the subsequences $w$ and $x$, and passes those strings to two 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. This is \textbf{not} a standard product construction!

The constraint $|w| = |x|$ implies that every string in $\text{Shuffle}(L)$ has even length, so $M'$ can accept only if $\text{next} = 1$.

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