Let L be an arbitrary regular language over the alphabet $\Sigma = \{0, 1\}$. Prove that the following languages are also regular.

1. $\text{Superstrings}(L) := \{xyz \mid y \in L \text{ and } x, z \in \Sigma^*\}$. This language contains all superstrings of strings in L. For example:

$$\text{Superstrings}(\{10010\}) = \left\{\underline{10010},\ 010\underline{10010},\ \underline{10010}11,\ 100\underline{10010}010,\ \cdots\right\}$$

[Hint: This is much easier than it looks.]

Solution: $(0+1)^*L(0+1)^*$. ■

2. $\text{Substrings}(L) := \{y \mid x, y, z \in \Sigma^* \text{ and } xyz \in L\}$. This language contains all substrings of strings in L.

Solution: Let $M = (Q, s, A, \delta)$ be an arbitrary DFA that accepts the regular language L. Without loss of generality, we assume that M satisfies the following conditions:

- Every state in Q is reachable in M from the start state s. (Otherwise, we can remove any unreachable states.)
- Every state in Q can reach an accepting state, except for a single dump state called fail. (Otherwise, we can merge all states in Q that can't reach an accepting state, and call the new merged state fail.)

We construct a new **NFA with multiple start states** $M' = (Q', S', A', \delta')$ that accepts $\text{Substrings}(L)$ as follows.

$$\begin{aligned} Q' &= Q \\ S' &= Q \\ A' &= Q \setminus \{\text{fail}\} \\ \delta'(q, a) &= \{\delta(q, a)\} \quad \text{for all } q \in Q \text{ and } a \in \Sigma \end{aligned}$$

M' first guesses the state $\delta^*(s, x)$ of M reached by the unknown prefix x, then reads the input string y and passes it to M, and finally guesses a suffix z that leads M to an accepting state. Because every state in M is reachable from s, guessing $\delta^*(s, x)$ is equivalent to guessing a state in Q. Similarly, because every state of M except fail can reach an accepting state, we know that there is an appropriate suffix z for every state except fail. ■

Solution: Let $M = (Q, s, A, \delta)$ be an arbitrary DFA that accepts the regular language L. We construct a new **NFA with ε-transitions** $M' = (Q', s', A', \delta')$ that accepts $\textsc{Substrings}(L)$ as follows.

$$\begin{aligned} Q' &= Q \cup \{\text{before}, \text{during}, \text{after}\} \\ s' &= (s, \text{before}) \\ A' &= A \times \{\text{after}\} \end{aligned}$$

$$\begin{aligned} \delta'((q, \text{before}), \varepsilon) &= \{(\delta(q, 0), \text{before}),\ (\delta(q, 1), \text{before}),\ (q, \text{during})\} \\ \delta'((q, \text{before}), 0) &= \varnothing \\ \delta'((q, \text{before}), 1) &= \varnothing \end{aligned}$$

$$\begin{aligned} \delta'((q, \text{during}), \varepsilon) &= \{(q, \text{after})\} \\ \delta'((q, \text{during}), 0) &= \{(\delta(q, 0), \text{during})\} \\ \delta'((q, \text{during}), 1) &= \{(\delta(q, 1), \text{during})\} \end{aligned}$$

$$\begin{aligned} \delta'((q, \text{after}), \varepsilon) &= \{(\delta(q, 0), \text{after}),\ (\delta(q, 1), \text{after})\} \\ \delta'((q, \text{after}), 0) &= \varnothing \\ \delta'((q, \text{after}), 1) &= \varnothing \end{aligned}$$

M' first guesses the symbols in x that were deleted **before** the input string y and passes them to M, then passes the input string y to M, and finally guesses the symbols in z that were deleted after the input string y and passes them to M. ■

3. $\textsc{Cycle}(L) := \{xy \mid x, y \in \Sigma^* \text{ and } yx \in L\}$. This language contains all strings that can be obtained by splitting a string in L into a prefix and a suffix and concatenating them in the wrong order. For example:

$$\textsc{Cycle}(\{\texttt{OOK!}, \texttt{OOKOOK}\}) = \left\{\texttt{OOK!}, \texttt{OK!O}, \texttt{K!OO}, \texttt{!OOK}, \texttt{OOKOOK}, \texttt{OKOOKO}, \texttt{KOOKOO}\right\}$$

Solution: Let $M = (Q, s, A, \delta)$ be an arbitrary DFA that accepts the regular language L. Without loss of generality, assume that M satisfies the following conditions:

- Every state in Q is reachable in M from the start state s. (Otherwise, we can remove any unreachable states.)
- Every state in Q can reach an accepting state, except for a single dump state called fail. (Otherwise, we can merge all states in Q that can't reach an accepting state, and call the new merged state fail.)

We construct a new NFA $M' = (Q', S', A', \delta')$ **with ε-transitions and multiple start states** that accepts $\textsc{Cycle}(L)$ as follows.

$$Q' = Q \times Q \times \{\mathsf{suffix}, \mathsf{prefix}\}$$
$$S' = \{(q, q, \mathsf{suffix}) \mid q \in Q \setminus \{\mathsf{fail}\}\}$$
$$A' = \{(q, q, \mathsf{prefix}) \mid q \in Q \setminus \{\mathsf{fail}\}\}$$

$$\delta'((p, q, \mathsf{suffix}), a) = \{(p, \delta(q, a), \mathsf{suffix})\} \qquad \text{for all } p, q \in Q \text{ and } a \in \Sigma$$
$$\delta'((p, q, \mathsf{prefix}), a) = \{(p, \delta(q, a), \mathsf{prefix})\} \qquad \text{for all } p, q \in Q \text{ and } a \in \Sigma$$

$$\delta'((p, q, \mathsf{suffix}), \varepsilon) = \{(p, s, \mathsf{prefix})\} \qquad \text{for all } p \in Q \text{ and } q \in A$$
$$\delta'((p, q, \mathsf{suffix}), \varepsilon) = \varnothing \qquad \text{for all } p \in Q \text{ and } q \in Q \setminus A$$
$$\delta'((p, q, \mathsf{prefix}), \varepsilon) = \varnothing$$

Intuitively, M' reads the input string yx and simulates M running on the original string xy. M' guesses and remembers the state $p = \delta^*(s, x)$, simulates M reading y starting from p, guesses the boundary between y and x via ε-transitions, and finally simulates M reading x starting from s.

- State (p, q, suffix) means that our simulation of M started in state p, the simulation is currently in state q, and M is reading the suffix x.
- State (p, q, prefix) means that our simulation of M started in state p, the simulation is currently in state q, and M is reading the prefix y.
- Whenever M is in an accepting state q, we can guess that we are done reading the suffix x, reset M to its start state, and start reading the prefix y.
- Finally, M' accepts if the simulation of M is in the same state after reading the prefix y where is started reading the suffix x.

■

Work on these later.

4. $\textsc{Subsequences}(L)$ is the set of all *subsequences* of strings in L. A subsequence of a string w is the result of deleting zero or more symbols from w, leaving the remaining symbols in order. For example:

$$\textsc{Subsequences}(\{10010\}) = \{\varepsilon, 0, 1, 00, 01, 10, 11, 000, 001, 010, 100, 101, 110, 0010, 1000, 1001, 10010\}$$

Solution: Let $M = (Q, s, A, \delta)$ be an arbitrary DFA that accepts the regular language L. We construct a new **NFA with ε-transitions** $M' = (Q', s', A', \delta')$ that accepts $\textsc{Subsequences}(L)$ as follows.

$$\begin{aligned} Q' &= Q \\ s' &= s \\ A' &= A \\ \delta'(q, 0) &= \{\delta(q, 0)\} \\ \delta'(q, 1) &= \{\delta(q, 1)\} \\ \delta'(q, \varepsilon) &= \{\delta(q, 0),\ \delta(q, 1)\} \end{aligned}$$

Intuitively, M' guesses which symbols have been deleted from its input string. ■

5. $\textsc{FlipOdds}(L) := \{\mathit{flipOdds}(w) \mid w \in L\}$, where the function *flipOdds* inverts every other bit in w, starting with the first bit. For example:

$$\mathit{flipOdds}(\texttt{0000111101010100}) = \texttt{1010010111111110}$$

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

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

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

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

$$\delta'((q, \textsc{False}), \texttt{0}) = \big(\delta(q, \texttt{0}),\ \textsc{True}\big)$$
$$\delta'((q, \textsc{True}), \texttt{0}) = \big(\delta(q, \texttt{1}),\ \textsc{False}\big)$$
$$\delta'((q, \textsc{False}), \texttt{1}) = \big(\delta(q, \texttt{1}),\ \textsc{True}\big)$$
$$\delta'((q, \textsc{True}), \texttt{1}) = \big(\delta(q, \texttt{0}),\ \textsc{False}\big)$$

■

6. UNFLIPODD1S(L) := $\{w \in \Sigma^* \mid \mathit{flipOdd1s}(w) \in L\}$, where the function *flipOdd1* inverts every other 1 bit of its input string, starting with the first 1. For example:

$$\mathit{flipOdd1s}(\underline{0000}1\underline{1}1\underline{1}00\underline{1}01\underline{0}1\underline{0}) = 0000\underline{1}0\underline{1}00\underline{0}01\underline{0}00$$

Solution: Let $M = (Q, s, A, \delta)$ be an arbitrary DFA that accepts the regular language L. We construct a new **DFA** $M' = (Q', s', A', \delta')$ that accepts 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, flip) of M' indicates that M is in state q, and we need to flip the next 1 bit if and only if $\mathit{flip} = \text{TRUE}$.

$$Q' = Q \times \{\text{TRUE}, \text{FALSE}\}$$
$$s' = (s, \text{TRUE})$$
$$A' = A \times \{\text{TRUE}, \text{FALSE}\}$$

$$\delta'((q, \text{TRUE}), 0) = \big(\delta(q, 0),\ \text{TRUE}\big)$$
$$\delta'((q, \text{FALSE}), 0) = \big(\delta(q, 0),\ \text{FALSE}\big)$$
$$\delta'((q, \text{TRUE}), 1) = \big(\delta(q, 0),\ \text{FALSE}\big)$$
$$\delta'((q, \text{FALSE}), 1) = \big(\delta(q, 1),\ \text{TRUE}\big)$$

■

7. FlipOdd1s$(L) := \{flipOdd1s(w) \mid w \in L\}$, where the function *flipOdd1s* is defined in the previous problem.

Solution: Let $M = (Q, s, A, \delta)$ be an arbitrary DFA that accepts the regular language L. We construct a new **NFA** $M' = (Q', s', A', \delta')$ that accepts FlipOdd1s(L) as follows.

Intuitively, M' receives some string *flipOdd1s*(w) as input, ***guesses*** which 0 bits to restore to 1s, and simulates M on the restored string w. No string in FlipOdd1s(L) has two 1s in a row, so if M' ever sees 11, it must reject.

Each state $(q, flip)$ 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 $flip = \text{True}$.

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

$$
\begin{aligned}
\delta'((q, \text{False}), 0) &= \big\{\big(\delta(q, 0),\ \text{False}\big)\big\}\\
\delta'((q, \text{True}), 0) &= \big\{\big(\delta(q, 0),\ \text{True}\big),\ \big(\delta(q, 1),\ \text{False}\big)\big\}\\
\delta'((q, \text{False}), 1) &= \big\{\big(\delta(q, 1),\ \text{True}\big)\big\}\\
\delta'((q, \text{True}), 1) &= \varnothing
\end{aligned}
$$

(The last transition indicates that we waited too long to flip a 0 to a 1, so we should kill the current execution thread.) ■

8. STUTTER$(L) = \{stutter(w) \mid w \in L\}$, where the function *stutter* duplicates every symbol in the input string:

$$stutter(w) := \begin{cases} \varepsilon & \text{if } w = \varepsilon \\ aa \bullet stutter(x) & \text{if } w = ax \end{cases}$$

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

$$\begin{aligned} Q' &= Q \times (\{‽\} \cup \Sigma) \cup \{fail\} && \text{for some new symbol } ‽ \notin \Sigma \\ s' &= (s, ‽) \\ A' &= \{(q, ‽) \mid q \in A\} \end{aligned}$$

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

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

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

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

- State $(q, ‽)$ means M' has read an even number of symbols of $stutter(w)$, so M should ignore the next symbol (if any).[a]
- For any symbol $a \in \Sigma$, state (q, a) means M' has read an odd number of symbols of $stutter(w)$, and the last symbol read was a. If the next symbol is an a, then M should transition normally; otherwise, the simulation should fail.
- The dump 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.

■

[a] The symbol ‽ is called an *interrobang*.

9. $\textsc{Unstutter}(L) = \{w \mid \mathit{stutter}(w) \in L\}$, where the function *stutter* is defined in the previous problem.

Solution: Let $M = (Q, s, A, \delta)$ be an arbitrary DFA that accepts the regular language L. We construct a **DFA** $M' = (Q', s', A', \delta')$ that accepts $\textsc{Unstutter}(L)$ as follows:

$$
\begin{aligned}
Q' &= Q \\
s' &= s \\
A' &= A \\
\delta'(q, a) &= \delta(\delta(q, a), a)
\end{aligned}
$$

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