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

$$\mathit{flipOdds}(0000111101010101) = 1010010111111111$$

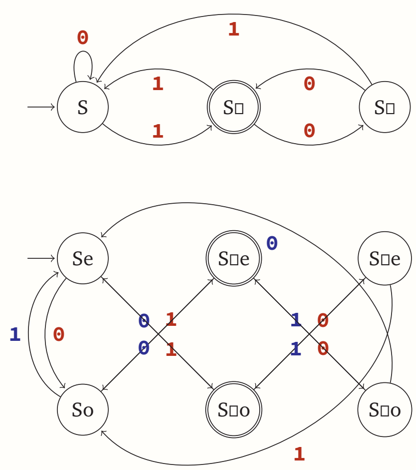
Solution: Let $M = (Q, s, A, \delta)$ be a DFA that accepts L. We construct a new DFA $M' = (Q', s', A', \delta')$ that accepts $\textsc{FlipOdds}(L)$ as follows.

To keep track of if the index is even/odd, we cross the original states Q with the set $\{\textsc{even}, \textsc{odd}\}$. Then every time an input is processed we flip this second coordinate. The starts state is $(s, \textsc{even})$. Effectively this is a flag determining if it is even or odd.

To flip the bits on odd indexes, we define the transition of odd indexed bits (i.e. $(q, \textsc{odd})$) as the transition of the original DFA with a flipped input and the even indexed bits (i.e. $(q, \textsc{even})$) as the transition of the original DFA with the same input.

$$Q' = Q \times \{\textsc{even}, \textsc{odd}\}$$
$$s' = (s, \textsc{even})$$
$$A' = A \times \{\textsc{even}, \textsc{odd}\}$$
$$\delta'((q, \textsc{odd}), 0) = \big(\delta(q, 1), \textsc{even}\big)$$
$$\delta'((q, \textsc{even}), 0) = \big(\delta(q, 0), \textsc{odd}\big)$$
$$\delta'((q, \textsc{odd}), 1) = \big(\delta(q, 0), \textsc{even}\big)$$
$$\delta'((q, \textsc{even}), 1) = \big(\delta(q, 1), \textsc{odd}\big)$$

Example: (Blue odd transitions, Red even transitions)



■

2. UnflipOdd1s(L) := $\{w \in \Sigma^* \mid flipOdd1s(w) \in L\}$, where the function *flipOdd1* inverts every other 1 bit of its input string, starting with the first 1. For example:

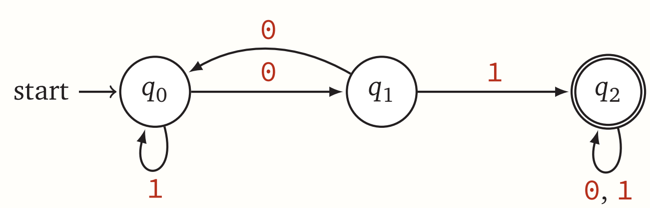
$$flipOdd1s(\texttt{0000111101010101}) = \texttt{0000010100010001}$$

Solution: Let $M = (Q, s, A, \delta)$ be a DFA that accepts L. We need to construct a new DFA/NFA $M' = (Q', s', A', \delta')$ that accepts UnflipOdd1s(L) as follows.

Intuition: By the definition of the language, the new DFA M' should have the same corresponding states at M, given that we unflip every alternating occurrence of 1 (i.e. 1 becomes 0). This can be done by including a flag at every state to know whether the next incoming 1 bit is to be flipped or not.

Strategy: So, every state is represented as $(q, flip)$ with $flip \in \{$ True,False $\}$, where $flip$ = True indicates that the next 1 bit in the input string is to be flipped. We start with $(s,$ True$)$ to ensure that the first 1 bit in the string would be flipped. When that happens, we also reset the flag to be False until the next 1 bit is read from the string at which point of time we just switch the flag back to be True and repeat the process. We can look at an example of this process with an arbitrary regular language input:

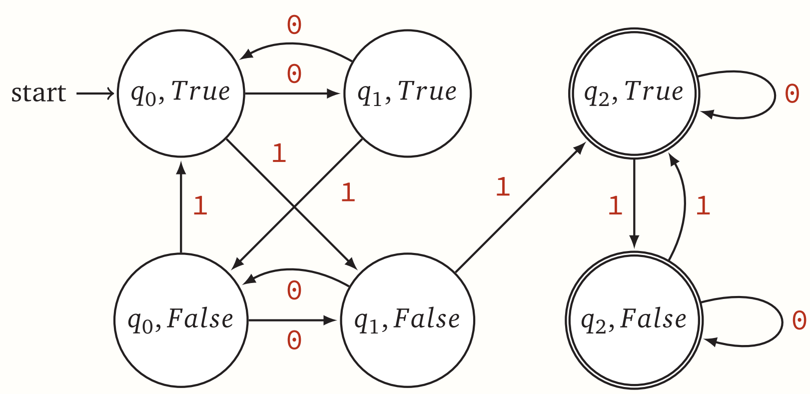
Example: For M' to accept the string 11, we and feed the flipped string 01 to M. DFA M:



$$\delta(s, 0) = q1$$
$$\delta(q2, 0) = q2;$$

q2 is the accepting state for M.

DFA M':



$$\begin{aligned}\delta'((q0,\textsc{True}),1) &= (\delta(q0,0),\textsc{False})\\ &= (q1,\textsc{False})\\ \delta'((q1,\textsc{False}),1) &= (\delta(q1,1),\textsc{True})\\ &= (q1,\textsc{False})\end{aligned}$$

The string is accepted as { q2,TRUE } is an accepting state in M'.

Solution: Essentially, when M' receives some string w as input, we flip every other bit starting from the first bit and stimulate this transformed string on M. This would mean that every state (q, flip) in M' indicates that M is in state q, with the difference that the next 1 bit would have to be flipped only if $\mathit{flip} = \textsc{True}$.

$$\begin{aligned}Q' &= Q \times \{\textsc{True}, \textsc{False}\}\\ s' &= (s, \textsc{True})\\ A' &= A \times \{\textsc{True}, \textsc{False}\}\\ \delta'((q,\textsc{False}),0) &= \big(\delta(q,0),\textsc{False}\big)\\ \delta'((q,\textsc{True}),0) &= \big(\delta(q,0),\textsc{True}\big)\\ \delta'((q,\textsc{False}),1) &= \big(\delta(q,1),\textsc{True}\big)\\ \delta'((q,\textsc{True}),1) &= \big(\delta(q,0),\textsc{False}\big)\end{aligned}$$

Once again, by treating 1 and 0 as synonyms for TRUE and FALSE, respectively, we can rewrite δ' more compactly as

$$\delta'((q,\mathit{flip}),a) = \big(\delta(q,\neg \mathit{flip} \wedge a),\ \mathit{flip} \oplus a\big)$$

■

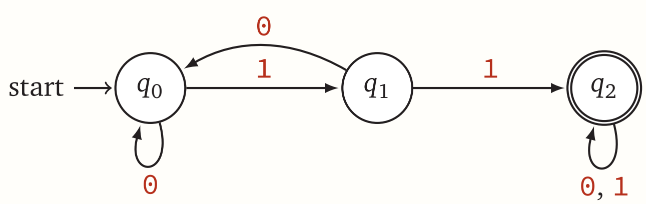
3. FLIPODD1S$(L) := \{flipOdd1s(w) \mid w \in L\}$, where the function *flipOdd1* is defined as in the previous problem.

Solution: Let $M = (Q, s, A, \delta)$ be a DFA that accepts L. We need to construct a new **NFA** $M' = (Q', s', A', \delta')$ that accepts FLIPODD1S(L).

Intuition: M' receives some string $flipOdd1s(w)$ as input, chooses which 0 bits to restore to 1s, and simulates M on the restored string w. We would have two cases when we see a 0 as it can either be just a 0 from the start or can be a 1 which was flipped and we would want to restore it.So while restoring, when the flip bit is TRUE($flip$ = TRUE) and we encounter a 0 ,we would have 2 possible states either we flip the bit or leave it as it is.Therefore,we are going to construct a NFA.

Strategy: We need to add TRUE, FALSE to the states to accommodate the *flip* bit.M' would never accept two consecutive 1s (Eg:11) because FLIPODD1S will flip every other 1 bit, so if M' ever sees 11, it rejects.Also,when we see a 1 and $flip$ = TRUE we should kill the execution thread as it indicates that we waited too long to flip a 0 to a 1.

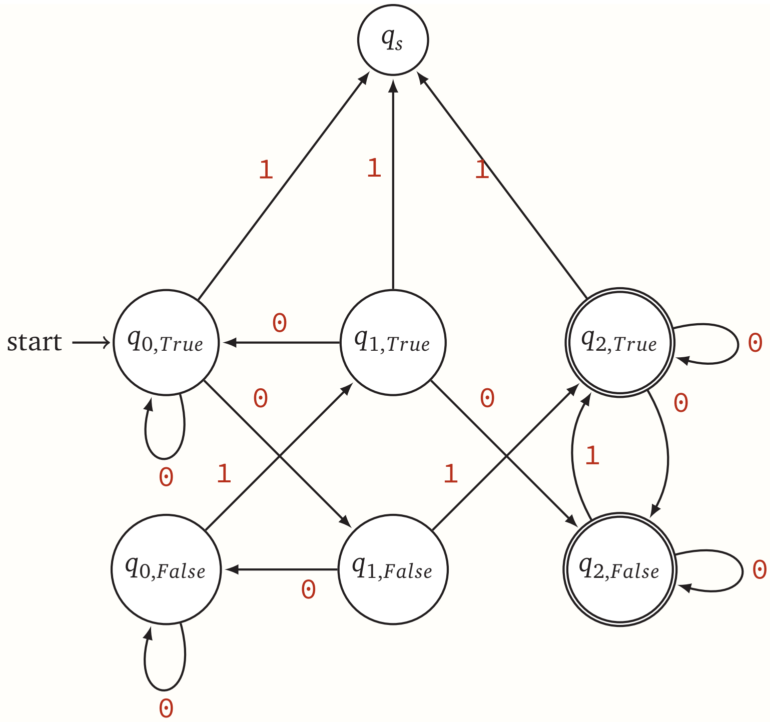
Example: Let M be the DFA of a language which accepts all strings containing the substring 11 .So M will be as follows:



q_2 is the accepting state of M.So for the string 11:

$$\delta(q_0, 1) = q_1$$
$$\delta(q_1, 1) = q_2$$

Since q_2 is the accepting state of M.For M' the accepting states would be $\{(q_2, \text{TRUE}), (q_2, \text{FALSE})\}$.



The input 01 to M' gives the final state $(q_2,\ \textsc{True})$ which is an accepting state of M'.

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

$$
\begin{aligned}
Q' &= Q \times \{\textsc{True}, \textsc{False}\} \\
s' &= (s, \textsc{True}) \\
A' &= A \times \{\textsc{True}, \textsc{False}\} \\
\delta'((q, \textsc{False}), 0) &= \{(\delta(q, 0),\ \textsc{False})\} \\
\delta'((q, \textsc{True}), 0) &= \{(\delta(q, 0),\ \textsc{True}),\ (\delta(q, 1),\ \textsc{False})\} \\
\delta'((q, \textsc{False}), 1) &= \{(\delta(q, 1),\ \textsc{True})\} \\
\delta'((q, \textsc{True}), 1) &= \varnothing
\end{aligned}
$$

■

4. $\text{cycle}(L) := \{xy | x, y \in \Sigma^*, yx \in L\}$, The language that accepts the rotations of string from a regular language.

Solution: HW problem. ■

5. Prove that the language $insert1(L) := \{x1y \mid xy \in L\}$ is regular.

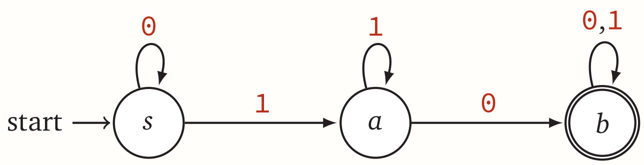
   Intuitively, $insert1(L)$ is the set of all strings that can be obtained from strings in L by inserting exactly one 1. For example, if $L = \{\varepsilon, \texttt{OOK!}\}$, then $insert1(L) = \{\texttt{1}, \texttt{1OOK!}, \texttt{O1OK!}, \texttt{OO1K!}, \texttt{OOK1!}, \texttt{OOK!1}\}$.

Solution: Let $M = (Q, s, A, \delta)$ be a DFA that accepts L. We need to construct an NFA $M' = (Q', s', A', \delta')$ that accepts $insert1(L)$.

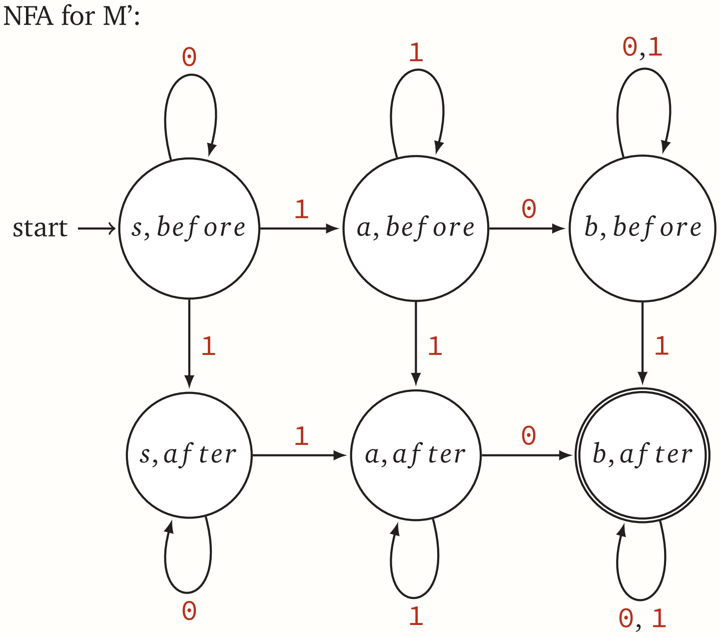
Intuition: Since the string in the language is represented as x1y, where x represents all the possible prefixes of a string in L and y represents all the suffixes. We can use two states - *before* and *after*. A state change can occur from *before* to *after* when we see a 1. If the machine is in the *before* state and it reads a 1, it can choose to either stay in the *before* state or move to the *after* state. If the machine is in the *after* state and reads a 1, it will stay in the *after* state since it had already chosen a 1 to ignore previously. Thus we combine the *before* and *after* states with the states of *M* *(Q)* to form the set of states *Q'* of *M'*.

Strategy: M' nondeterministically simulates M running on a string prefix, then uses a 1 character and then runs M the rest of the input string. The transformation is best shown in the following example:

DFA for M:



NFA for M':



Solution: So we need to simply formalize the transformation above. First we know we need to double the states. Σ stays the same. For the delta functions both sets of DFAs have the same transitions but we need to add a 1 transition from the DFA simulating the prefix to the DFA simulating the suffix.

- The state (q, before) means (the simulation of) M is in state q and M' has not yet skipped over a 1.
- The state (q, after) means (the simulation of) M is in state q and M' has already skipped over a 1.

$$Q' := Q \times \{\mathit{before}, \mathit{after}\}$$
$$s' := (s, \mathit{before})$$
$$A' := \big\{(q, \mathit{after}) \bigm| q \in A\big\}$$
$$\delta'((q, \mathit{before}), a) = \begin{cases} \big\{(\delta(q,a), \mathit{before}),\ (q, \mathit{after})\big\} & \text{if } a = 1 \\ \big\{(\delta(q,a), \mathit{before})\big\} & \text{otherwise} \end{cases}$$
$$\delta'((q, \mathit{after}), a) = \big\{(\delta(q,a), \mathit{after})\big\}$$

■

6. Prove that the language $delete\texttt{1}(L) := \{xy \mid x\texttt{1}y \in L\}$ is regular.

   Intuitively, $delete\texttt{1}(L)$ is the set of all strings that can be obtained from strings in L by deleting exactly one 1. For example, if $L = \{\texttt{101101}, \texttt{00}, \varepsilon\}$, then $delete\texttt{1}(L) = \{\texttt{01101}, \texttt{10101}, \texttt{10110}\}$.

Solution: Let $M = (Q, s, A, \delta)$ be a DFA that accepts L. We construct an NFA $M' = (Q', s', A', \delta')$ with ε-transitions that accepts $delete\texttt{1}(L)$ as follows.

Intuitively, M' simulates M, but inserts a single 1 into M's input string at a nondeterministically chosen location.

- The state (q, before) means (the simulation of) M is in state q and M' has not yet inserted a 1.
- The state (q, after) means (the simulation of) M is in state q and M' has already inserted a 1.

$$\begin{aligned} Q' &:= Q \times \{\mathit{before}, \mathit{after}\} \\ s' &:= (s, \mathit{before}) \\ A' &:= \big\{(q, \mathit{after}) \bigm| q \in A\big\} \end{aligned}$$

$$\begin{aligned} \delta'((q, \mathit{before}), \varepsilon) &= \big\{(\delta(q, \texttt{1}), \mathit{after})\big\} \\ \delta'((q, \mathit{after}), \varepsilon) &= \varnothing \end{aligned}$$

$$\begin{aligned} \delta'((q, \mathit{before}), a) &= \big\{(\delta(q, a), \mathit{before})\big\} \\ \delta'((q, \mathit{after}), a) &= \big\{(\delta(q, a), \mathit{after})\big\} \end{aligned}$$

■

7. Consider the following recursively defined function on strings:

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

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

- $stutter(\texttt{PRESTO}) = \texttt{PPRREESSTTOO}$
- $stutter(\texttt{HOCUS}\diamond\texttt{POCUS}) = \texttt{HHOOCCUUSS}\diamond\diamond\texttt{PPOOCCUUSS}$

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

Intuitively, 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.

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

■

(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. wWe 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[a] 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.

$$Q' = Q \times (\{\bullet\} \cup \Sigma) \cup \{fail\} \qquad \text{for some new symbol } \bullet \notin \Sigma$$
$$s' = (s, \bullet)$$
$$A' = \{(q, \bullet) \mid q \in A\}$$

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

■

[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 $stutter(R)$ as follows:

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

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

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

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:

$$stutter\big((1 + \varepsilon)(01)^*(0 + \varepsilon) + 0^*\big) = (11 + \varepsilon)(0011)^*(00 + \varepsilon) + (00)^*$$

Although this may look simpler, actually *proving* that it works requires the same induction arguments. ■

8. Consider the following recursively defined function on strings:

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

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

- $evens$(EXPELLIARMUS) = XELAMS
- $evens$(AVADA⋄KEDAVRA) = 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 $M = (Q, s, A, \delta)$ be a DFA that accepts L. We construct a **DFA** $M' = (Q', s', A', \delta')$ 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 $M = (Q, s, A, \delta)$ be a DFA that accepts L. We construct an NFA $M' = (Q', s', A', \delta')$ that accepts $evens(L)$ as follows.

Intuitively, 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.

$$Q' = Q$$
$$s' = s$$
$$A' = A \cup \Big\{ q \in Q \;\Big|\; \delta(q, a) \cap A \neq \varnothing \text{ for some } a \in \Sigma \Big\}$$
$$\delta'(q, a) = \bigcup_{b \in \Sigma} \big\{ \delta\big(\delta(q, b), a\big) \big\}$$

■