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

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

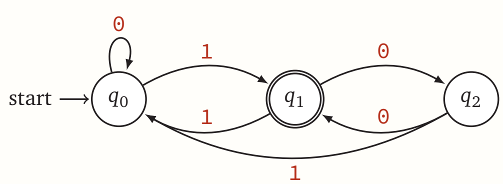
$$flipOdds(\texttt{00000111101010101}) = \texttt{01010010111111111}$$

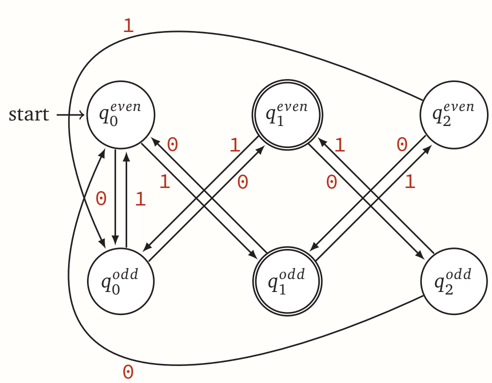
Solution: Let $M = (Q, s, A, \delta)$ be a DFA that accepts L. We construct a new DFA $M' = (Q', s', A', \delta')$ that accepts $\textsc{FlipOdds}(L)$ as follows.

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.

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





■

2. FlipOdd1s(L) := $\{flipOdd1s(w) \mid w \in L\}$, where the function *flipOdd1* inverts every other 1 bit of its input string, starting with the second 1 (which would have a index of 1 in a 0-indexing scheme). For example:

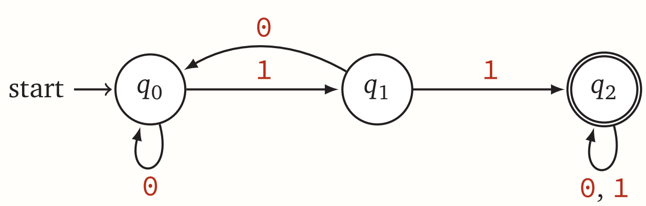
$$flipOdd1s(\mathtt{00001\underline{1}1\underline{1}10\underline{1}010\underline{1}01}) = \mathtt{00001\underline{0}1\underline{0}10\underline{0}010\underline{0}01}$$

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: All we need to do is keep track of if we're on a odd-indexed 1 and if we are, instead of accepting the 1, we accept a zero. So it's similar to the first problem but in this case we keep the zero transitions on the same level but only change between even/odd when we see a 1. And if we're on the odd level, the 1-transition becomes a 0-transition.

Strategy: We need to add Even, Odd 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 $flip = \text{True}$ we should kill the execution thread as it indicates that we waited too long to flip a 0 to a 1.

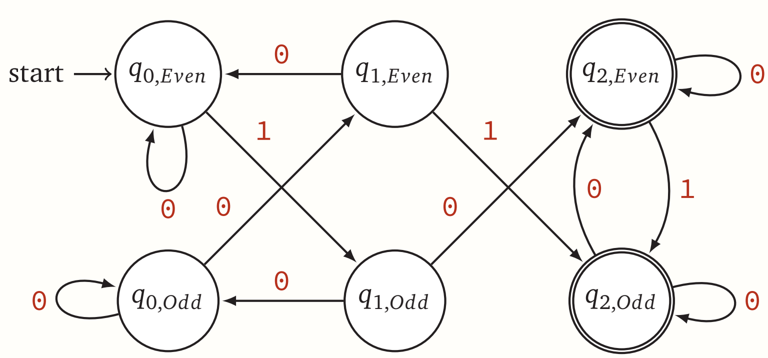
Example: Let M be the DFA of a language which accepts all strings containing the substring 11 .So M will be as follows:



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 $\mathit{flip} = \textsc{True}$.

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

■

3. 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 second 1 (which would have a index of 1 in a 0-indexing scheme). For example:

$$flipOdd1s(00001111101010101) = 00001010100010001$$

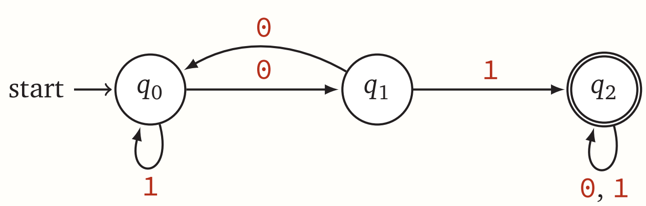
Solution: Let $M = (Q, s, A, \delta)$ be a DFA that accepts L. We need to construct a new DFA/NFA $M' = (Q', s', A', \delta')$ that accepts UnflipOdd1s(L) as follows.

Intuition: This is seems like a complex language but let's break it down. First thing to realize is that the language is not a one-to-one relationship. For instance, let's say $flipOdd1s(w) = 010$. In this case, w could be a number of things. possible solutions for w include: $w = 010$ or $w = 011$. In both cases, $flipOdd1s(w) = 010$. Also observe that runs of 1's cannot be a part of the language of L because $flipOdd1s(w)$ always results in atleast one 0 in between every pair of 1's

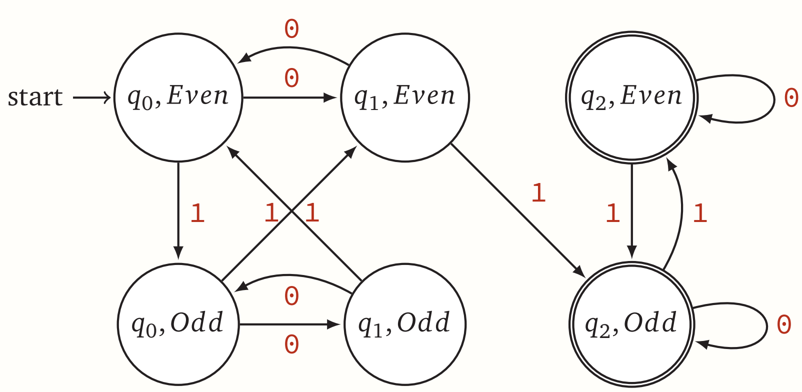
So what does the NFA for UnflipOdd1s(L) do? Well when you see the first 1, you don't want to do anything. But once you get that first 1, you need to unflip a zero (accept a 1 instead of a 0) before you get the the next 1.

Strategy: So, every state is represented as $(q, flip)$ with $flip \in \{$ Even, Odd $\}$, where $flip$ = Odd indicates that we need to accept a 1 where L would have accepted a 0. We start with $(s,$ Even$)$ to ensure that the first 1 bit in the string would not be flipped. When that happens, we also reset the flag to be Odd until the next 1 bit is read from the string at which point of time we just switch the flag back to be Even 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 111, we and feed the flipped string 101 to M. DFA M:



DFA M':



Solution: So now let's geenralize what we did constructing the NFA for L' above to any arbitrary version of L.

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

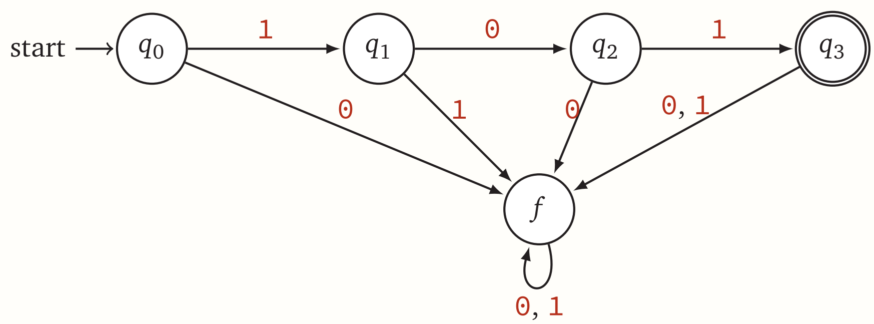
Once again, by treating `1` and `0` as synonyms for Even and Odd, respectively, we can rewrite δ' more compactly as

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

■

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: The given language $\text{cycle}(L)$ is a set of strings that can be obtained by spliting a string $w \in L$ into two parts and swapping the order of the parts. As an example, if $L = \{101\}$, then $\text{cycle}(L) = \{101, 011, 110\}$. To get the idea, consider the following DFA $M = (\Sigma, Q, s, A, \delta)$ for the langauge L.



Suppose we start from the state q_2 instead of q_0, traverse through the DFA to reach q_3, take an ϵ-transition to q_0, then continue traversal until reaching back to q_2. This traversal would represent the string 110, which is in $\text{cycle}(L)$. Therefore, if we could start from an arbitrary state $q \in Q$ and traverse the DFA in a similar way as presented above, the traversals would represent the language $\text{cycle}(L)$.

At a high-level, we construct an NFA with $|Q|$ different copies of a pair of M(therefore, it would be the total of $2|Q|$ copies of M). Each pair would correspond to a certain starting state, among all states in Q. For each pair, one copy of M corresponds to pre-cycle, and the other corresponds to post-cycle. We also add a pseudo start state s' that can ϵ-transition to one of the copies. Then, we modify the transition function so it allows the traversal explained above.

Formally, we construct NFA $M' := (\Sigma, Q', s', A', \delta')$, where

- $Q' := (Q \times Q \times \{pre, post\}) \cup \{s'\}$
- $A' := \{(q, q, post) \mid q \in Q\}$
- The transition function δ' is defined as follows,

$$\delta'(s', \epsilon) = \{(q, q, pre) \mid q \in Q\}$$

$$\delta'((q_i, q_j, pre), x) = \begin{cases} (q_i, s, post) & \text{if } q_j \in A, x = \epsilon \\ (q_i, \delta(q_j, x), pre) & otherwise \end{cases}$$

$$\delta'((q_i, q_j, post), x) = (q_i, \delta(q_j, x), post)$$

A state $q' = (q_i, q_j, pre)$, for an example, represents that the traversal started from q_i, so far the input string led to q_j, and we haven't cycled yet. Once we reach one of the original accepting states within a pre-cycle copy, we can take an ϵ-transition to the original starting state s of the corresponding post-cycle copy, and then continue traversal. We accept when we reach the state from which we started the traversal within the post-cycle copy. ■

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

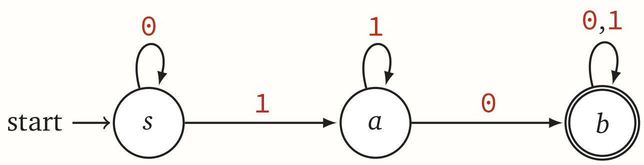
   Intuitively, $insert1(L)$ is the set of all strings that can be obtained from strings in L by inserting exactly one 1. For example, if $L = \{\varepsilon, \texttt{OOK!}\}$, then $insert1(L) = \{\texttt{1}, \texttt{1OOK!}, \texttt{O1OK!}, \texttt{OO1K!}, \texttt{OOK1!}, \texttt{OOK!1}\}$.

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 $insert1(L)$.

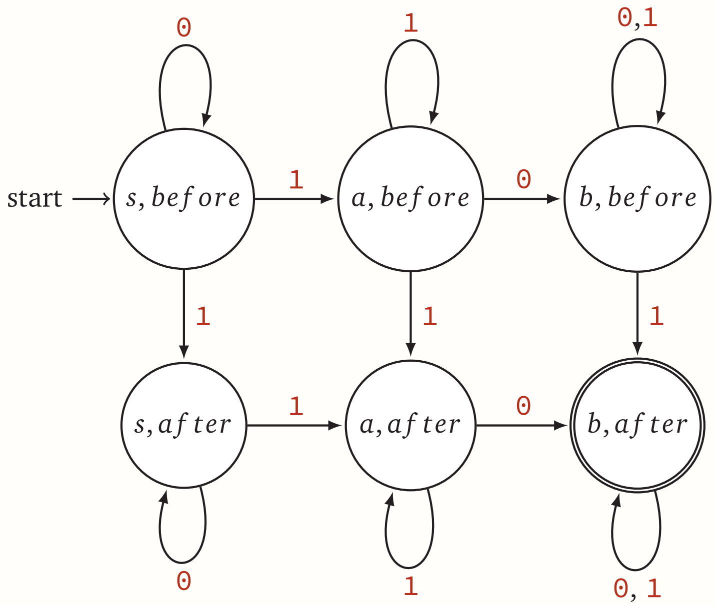
Intuition: Since the string in the language is represented as x1y, where x represents all the possible prefixes of a string in L and y represents all the suffixes. We can use two states - *before* and *after*. A state change can occur from *before* to *after* when we see a 1. If the machine is in the *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.

$$Q' := Q \times \{\mathit{before}, \mathit{after}\}$$
$$s' := (s, \mathit{before})$$
$$A' := \big\{(q, \mathit{after}) \;\big|\; q \in A\big\}$$

$$\delta'((q, \mathit{before}), a) = \begin{cases} \big\{(\delta(q,a), \mathit{before}),\ (q, \mathit{after})\big\} & \text{if } a = 1 \\ \big\{(\delta(q,a), \mathit{before})\big\} & \text{otherwise} \end{cases}$$
$$\delta'((q, \mathit{after}), a) = \big\{(\delta(q,a), \mathit{after})\big\}$$

■

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

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

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 $delete\mathtt{1}(L)$ as follows.

Intuitively, M' simulates M, but inserts a single $\mathtt{1}$ 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 $\mathtt{1}$.
- The state $(q, after)$ means (the simulation of) M is in state q and M' has already inserted a $\mathtt{1}$.

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

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

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

■

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.

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

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

■

[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:

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

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:

- $evens(\texttt{EXPELLIARMUS}) = \texttt{XELAMS}$
- $evens(\texttt{AVADA}\diamond\texttt{KEDAVRA}) = \texttt{VD}\diamond\texttt{EAR}$.

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.

$$\begin{aligned}
Q' &= Q \\
s' &= s \\
A' &= A \cup \left\{ q \in Q \;\middle|\; \delta(q,a) \cap A \neq \varnothing \text{ for some } a \in \Sigma \right\} \\
\delta'(q,a) &= \bigcup_{b \in \Sigma} \left\{ \delta\big(\delta(q,b),a\big) \right\}
\end{aligned}$$

■