

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

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

**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  $\text{FLIPODDS}(L)$  as follows.

To keep track of if the index is even/odd, we cross the original states  $Q$  with the set  $\{\text{EVEN}, \text{ODD}\}$ . Then every time an input is processed we flip this second coordinate. The starts state is  $(s, \text{EVEN})$ . 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.  $(q, \text{ODD})$ ) as the transition of the original DFA with a flipped input and the even indexed bits (i.e.  $(q, \text{EVEN})$ ) as the transition of the original DFA with the same input.

$$Q' = Q \times \{\text{EVEN}, \text{ODD}\}$$

$$s' = (s, \text{EVEN})$$

$$A' = A \times \{\text{EVEN}, \text{ODD}\}$$

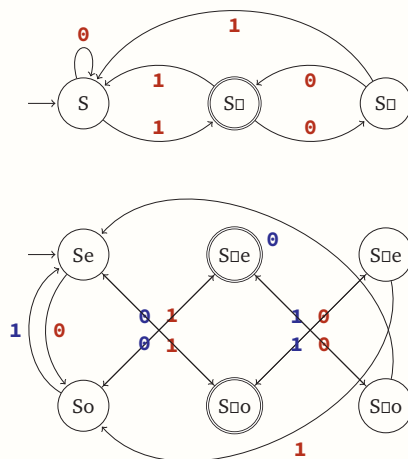
$$\delta'((q, \text{ODD}), 0) = (\delta(q, 1), \text{EVEN})$$

$$\delta'((q, \text{EVEN}), 0) = (\delta(q, 0), \text{ODD})$$

$$\delta'((q, \text{ODD}), 1) = (\delta(q, 0), \text{EVEN})$$

$$\delta'((q, \text{EVEN}), 1) = (\delta(q, 1), \text{ODD})$$

Example: (Blue odd transitions, Red even transitions)



2.  $\text{UNFLIPODD1s}(L) := \{w \in \Sigma^* \mid \text{flipOdd1s}(w) \in L\}$ , where the function  $\text{flipOdd1s}$  inverts every other **1** bit of its input string, starting with the first **1**. For example:

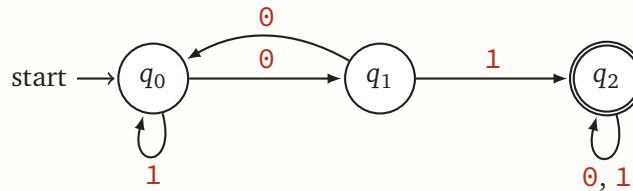
$$\text{flipOdd1s}(0000\underline{1}11\underline{1}010\underline{1}010\underline{1}) = 0000\underline{0}1\underline{1}0\underline{1}00\underline{1}00\underline{0}1$$

**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  $\text{UNFLIPODD1s}(L)$  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  $(q, \text{flip})$  with  $\text{flip} \in \{\text{TRUE}, \text{FALSE}\}$ , where  $\text{flip} = \text{TRUE}$  indicates that the next **1** bit in the input string is to be flipped. We start with  $(s, \text{TRUE})$  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:

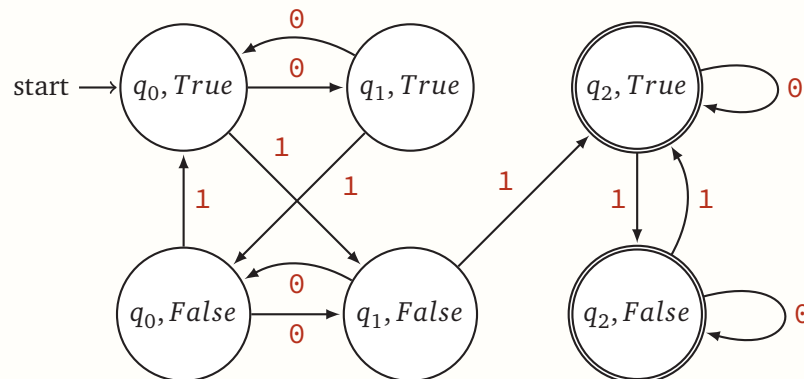
Example: For  $M'$  to accept the string **11**, we and feed the flipped string **01** to  $M$ . DFA  $M$ :



$$\begin{aligned}\delta(s, 0) &= q1 \\ \delta(q2, 0) &= q2;\end{aligned}$$

$q2$  is the accepting state for  $M$ .

DFA  $M'$ :



$$\begin{aligned}
\delta'((q0, \text{TRUE}), 1) &= (\delta(q0, 0), \text{FALSE}) \\
&= (q1, \text{FALSE}) \\
\delta'((q1, \text{FALSE}), 1) &= (\delta(q1, 1), \text{TRUE}) \\
&= (q1, \text{FALSE})
\end{aligned}$$

The string is accepted as  $\{ q2, \text{TRUE} \}$  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  $(q, \text{flip})$  in  $M'$  indicates that  $M$  is in state  $q$ , with the difference that the next 1 bit would have to be flipped only if  $\text{flip} = \text{TRUE}$ .

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

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

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

■

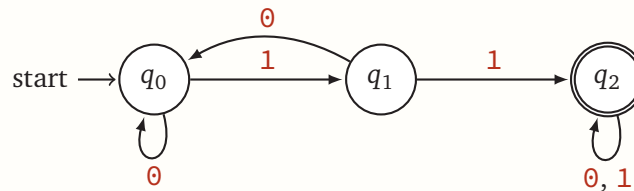
3.  $\text{FLIPODD1s}(L) := \{\text{flipOdd1s}(w) \mid w \in L\}$ , where the function  $\text{flipOdd1s}$  is defined as in the previous problem.

**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  $\text{FLIPODD1s}(L)$ .

**Intuition:**  $M'$  receives some string  $\text{flipOdd1s}(w)$  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  $\text{TRUE}(\text{flip} = \text{TRUE})$  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  $\text{FLIPODD1s}$  will flip every other 1 bit, so if  $M'$  ever sees 11, it rejects. Also, when we see a 1 and  $\text{flip} = \text{TRUE}$  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:

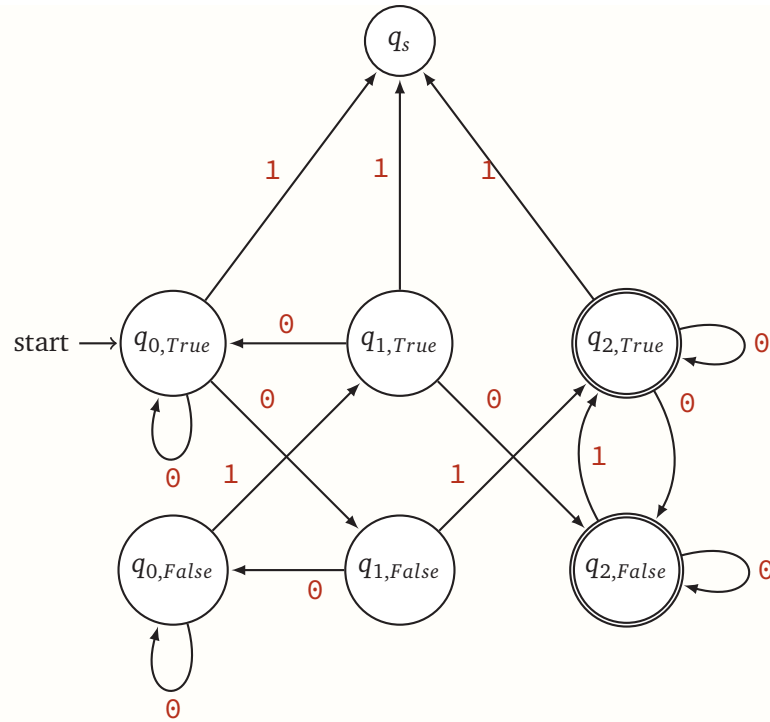


$q_2$  is the accepting state of  $M$ . So for the string 11:

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

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

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  $(q_2, \text{TRUE})$  which is an accepting state of  $M'$ .

**Solution:** Each state  $(q, \text{flip})$  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  $\text{flip} = \text{TRUE}$ .

$$Q' = Q \times \{\text{TRUE}, \text{FALSE}\}$$

$$s' = (s, \text{TRUE})$$

$$A' = A \times \{\text{TRUE}, \text{FALSE}\}$$

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

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

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

$$\delta'((q, \text{TRUE}), 1) = \emptyset$$

■

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

**Solution:** HW problem.

■

5. Prove that the language  $\text{insert}1(L) := \{x1y \mid xy \in L\}$  is regular.

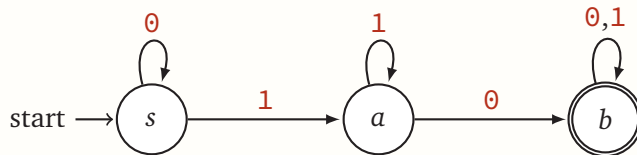
Intuitively,  $\text{insert}1(L)$  is the set of all strings that can be obtained from strings in  $L$  by inserting exactly one **1**. For example, if  $L = \{\varepsilon, 00K!\}$ , then  $\text{insert}1(L) = \{1, 100K!, 010K!, 001K!, 00K1!, 00K!1\}$ .

**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  $\text{insert}1(L)$ .

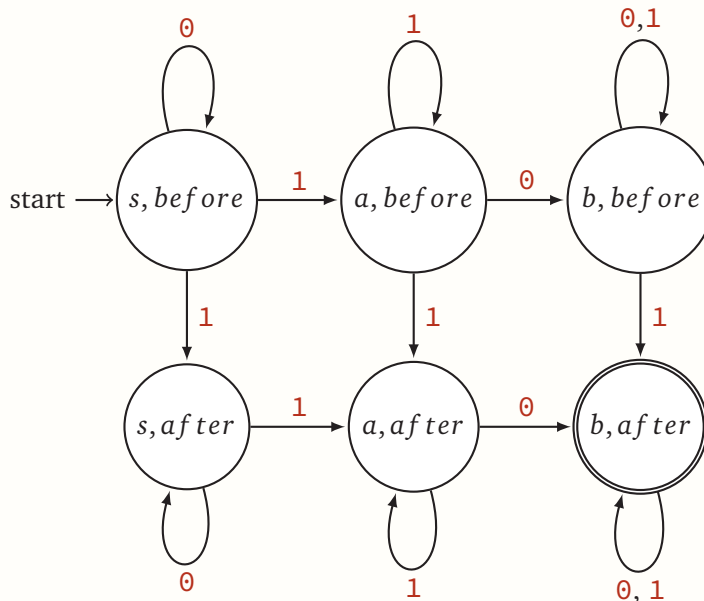
**Intuition:** Since the string in the language is represented as  $x1y$ , where  $x$  represents all the possible prefixes of a string in  $L$  and  $y$  represents all the suffixes. We can use two states - *before* and *after*. A state change can occur from *before* to *after* when we see a **1**. 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.  $\Sigma$  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  $(q, \text{before})$  means (the simulation of)  $M$  is in state  $q$  and  $M'$  has not yet skipped over a **1**.
- The state  $(q, \text{after})$  means (the simulation of)  $M$  is in state  $q$  and  $M'$  has already skipped over a **1**.

$$Q' := Q \times \{\text{before}, \text{after}\}$$

$$s' := (s, \text{before})$$

$$A' := \{(q, \text{after}) \mid q \in A\}$$

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

$$\delta'((q, \text{after}), a) = \{(\delta(q, a), \text{after})\}$$

■

6. Prove that the language  $\text{delete1}(L) := \{xy \mid x1y \in L\}$  is regular.

Intuitively,  $\text{delete1}(L)$  is the set of all strings that can be obtained from strings in  $L$  by deleting exactly one **1**. For example, if  $L = \{101101, 00, \epsilon\}$ , then  $\text{delete1}(L) = \{01101, 10101, 10110\}$ .

**Solution:** Let  $M = (Q, s, A, \delta)$  be a DFA that accepts  $L$ . We construct an NFA  $M' = (Q', s', A', \delta')$  with  $\epsilon$ -transitions that accepts  $\text{delete1}(L)$  as follows.

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

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

$$Q' := Q \times \{\text{before}, \text{after}\}$$

$$s' := (s, \text{before})$$

$$A' := \{(q, \text{after}) \mid q \in A\}$$

$$\delta'((q, \text{before}), \epsilon) = \{(\delta(q, \mathbf{1}), \text{after})\}$$

$$\delta'((q, \text{after}), \epsilon) = \emptyset$$

$$\delta'((q, \text{before}), a) = \{(\delta(q, a), \text{before})\}$$

$$\delta'((q, \text{after}), a) = \{(\delta(q, a), \text{after})\}$$

■



7. Consider the following recursively defined function on strings:

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

Intuitively,  $\text{stutter}(w)$  doubles every symbol in  $w$ . For example:

- $\text{stutter}(\text{PRESTO}) = \text{PPRREESSTT00}$
- $\text{stutter}(\text{HOCUS} \diamond \text{POCUS}) = \text{HH00CCUUSS} \diamond \text{PP00CCUUSS}$

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

Intuitively,  $M'$  reads its input string  $w$  and simulates  $M$  running on  $\text{stutter}(w)$ . 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  $\text{stutter}(L) := \{\text{stutter}(w) \mid w \in L\}$  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  $\text{stutter}(L)$  as follows.

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

- State  $(q, \bullet)$  means  $M'$  has just read an even-indexed<sup>a</sup> symbol in  $\text{stutter}(w)$ , 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  $\text{stutter}(w)$ , 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  $\text{stutter}(w)$  for any string  $w$ .

$$Q' = Q \times (\{\bullet\} \cup \Sigma) \cup \{\text{fail}\} \quad \text{for some new symbol } \bullet \notin \Sigma$$

$$s' = (s, \bullet)$$

$$A' = \{(q, \bullet) \mid q \in A\}$$

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

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

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

<sup>a</sup>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  $\text{stutter}(R)$  as follows:

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

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

- $\text{stutter}(A \cup B) = \text{stutter}(A) \cup \text{stutter}(B)$
- $\text{stutter}(A \cdot B) = \text{stutter}(A) \cdot \text{stutter}(B)$
- $\text{stutter}(A^*) = (\text{stutter}(A))^*$

These identities can all be proved by inductive definition-chasing, after which the claim  $L(\text{stutter}(R)) = \text{stutter}(L(R))$  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  $\text{stutter}(R)$  by replacing every explicit string  $w \in \Sigma^*$  inside  $R$  with  $\text{stutter}(w)$  (with additional parentheses if necessary). For example:

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

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

8. Consider the following recursively defined function on strings:

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

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

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

Once again, let  $L$  be an arbitrary regular language.

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

- 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  $\text{evens}(L) := \{\text{evens}(w) \mid w \in L\}$  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  $\text{evens}(L)$  as follows.

Intuitively,  $M'$  reads the input string  $\text{evens}(w)$  and simulates  $M$  running on string  $w$ , while nondeterministically guessing the missing symbols in  $w$ .

- When  $M'$  reads the symbol  $a$  from  $\text{evens}(w)$ , it guesses a symbol  $b \in \Sigma$  and simulates  $M$  reading  $ba$  from  $w$ .
- When  $M'$  finishes  $\text{evens}(w)$ , it guesses whether  $w$  has even or odd length, and in the odd case, it guesses the last symbol in  $w$ .

$$Q' = Q$$

$$s' = s$$

$$A' = A \cup \{q \in Q \mid \delta(q, a) \cap A \neq \emptyset \text{ for some } a \in \Sigma\}$$

$$\delta'(q, a) = \bigcup_{b \in \Sigma} \{\delta(\delta(q, b), a)\}$$

■