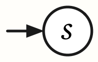
Design ***nondeterministic*** finite-state automata that accept the following languages over the binary alphabet $\{0, 1\}$. Briefly describe how your NFA works.

---

0. $\varnothing$

**Solution (one state):** The following NFA rejects every string, because it has no accepting states:



This NFA $(Q, s, A, \delta)$ is formally defined as follows:

$$\begin{aligned} Q &= \{s\} \\ s &= \text{the only state} \\ A &= \varnothing \\ \delta(s, 0) &= \varnothing \\ \delta(s, 1) &= \varnothing \end{aligned}$$

■

**Solution (no states):** The following NFA with “multiple” start states rejects every string, because it has no accepting states:

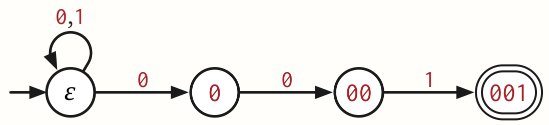
In fact, it has no states at all! This NFA $(Q, S, A, \delta)$ is formally defined as follows:

$$\begin{aligned} Q &= \varnothing \\ S &= \varnothing \\ A &= \varnothing \\ \delta(q, 0) &= \{42\} && \text{for all } q \in Q \\ \delta(q, 1) &= \{\texttt{OOK!}\} && \text{for all } q \in Q \end{aligned}$$

Yes, the definition of $\delta$ is correct, *because there are no states $q \in Q$*. It is also formally redundant, because there is only one function from $\varnothing$ to $\varnothing$. ■

1. Binary strings that end with the suffix 001.

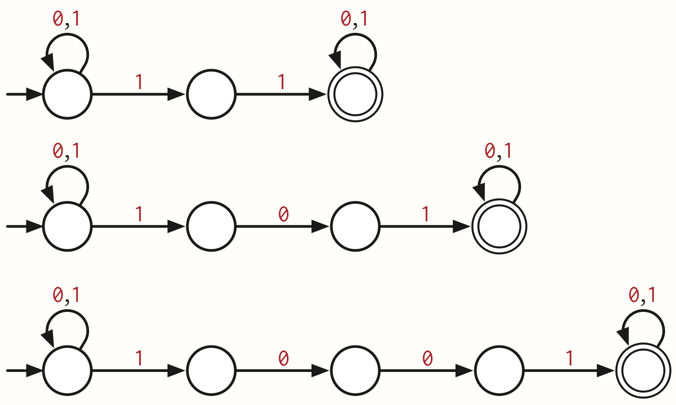
**Solution:**



The NFA nondeterministically guesses which 0 begins the suffix 001. The states are named by the longest prefix of the suffix 001 that the NFA has read so far (assuming it guesses correctly). ■

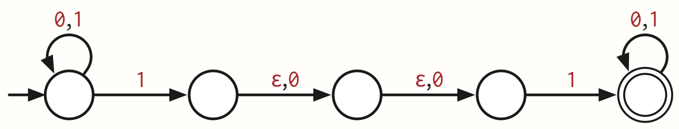
2. Binary strings that contain at least one of the substrings 11, 101, and 1001.

**Solution:** The following NFA has multiple start states:



The NFA guesses at the start which substrings its input contains, and then later guesses when that chosen substring begins. ■

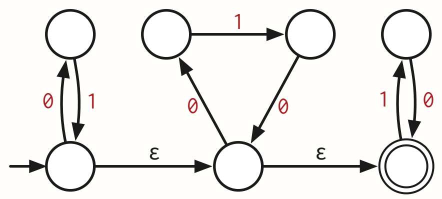
**Solution:** The following NFA has $\varepsilon$-transitions:



The NFA guesses when the required substring begins, and then guesses whether that substring contains zero, one, or two 0s. ■

3. $(01)^*(010)^*(10)^*$

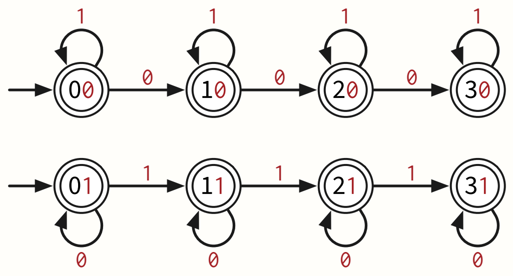
**Solution:**



The NFA guesses the boundaries between the prefix matching $(01)^*$, the substring matching $(010)^*$, and the suffix matching $(10)^*$. ■

4. $\{w \in \{0,1\}^* \mid \min\{\#(0,w), \#(1,w)\} \leq 3\}$

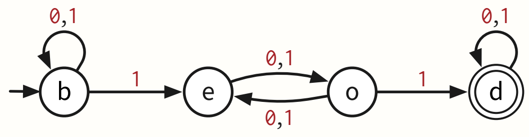
**Solution:** The following NFA with multiple start states guesses one of the symbols, and then accepts any string that contains at most three of that symbol. Each state is labeled with how many of the target symbol it has read so far; for example, state 20 should be read "two 0s".



(There is no missing dump state; $\delta(30,0) = \delta(31,1) = \varnothing$.) ■

5. Binary strings that contain a substring of the form $1x1$, where $|x|$ is odd.

**Solution:** The following NFA guesses when the substring $1x1$ begins and ends. The state labels are mnemonic for **b**efore the first 1, **e**ven number of symbols after the first 1, **o**dd number of symbols after the first 1, and **d**one.

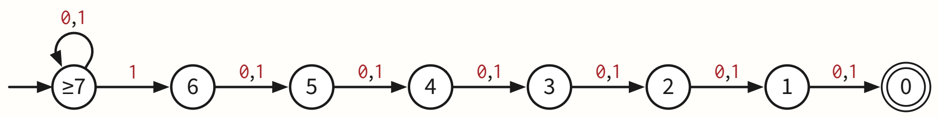


■

**Problems to think about later:**

6. Binary strings with length at least 7 whose 7th symbol from the end is 1.

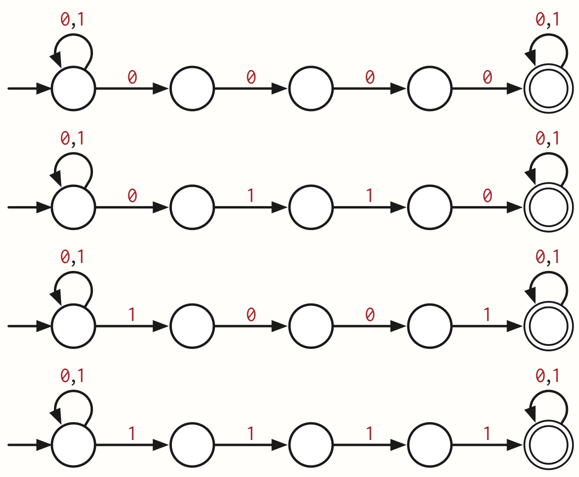
**Solution:** The following NFA guesses which 1 begins the last seven symbols in the input string. Each state is labeled with the NFA's guess for the number of symbols left in the input.



The equivalent DFA has 255 states, one for each non-empty subset of NFA states. ■

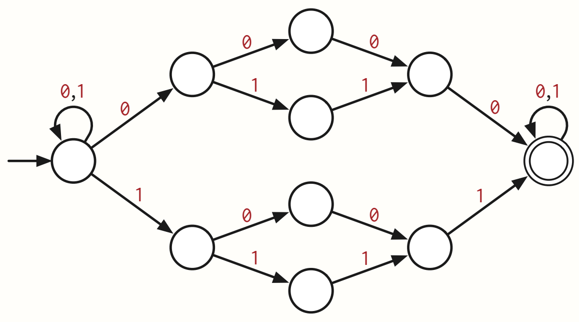
7. Binary strings that contain at least one palindrome substring of length 4. *[Hint: What are the palindromes of length 4?]*

**Solution:** The following NFA uses multiple start states, one for each binary palindrome of length 4. The NFA guesses which palindrome of length 4 the input string contains, and then guesses when that palindrome substring begins.



■

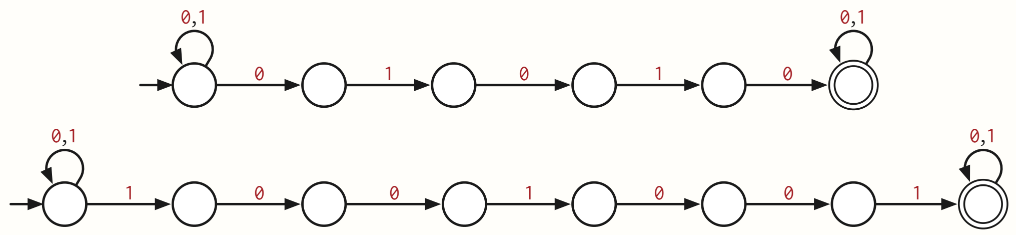
**Solution:** The following NFA guesses when the palindrome substring of length 4 begins, and then accepts if the next four symbols are a palindrome.



■

8. Binary strings that contain both `01010` and `1001001` as substrings.

**Solution:** We use a product of two NFAs, each of which accepts strings containing one of the two substrings.



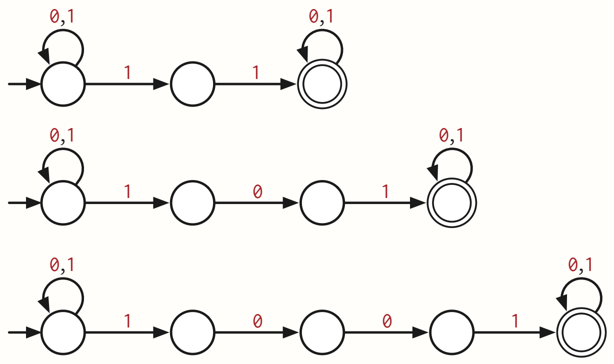
The product of two NFAs $M_1 = (Q_1, s_1, A_1, \delta_1)$ and $M_2 = (Q_2, s_2, A_2, \delta_2)$ is defined identically to the product of two DFAs, except for the transition function. The choice of accepting states is specific to this language.

$$\begin{aligned} Q &= Q_1 \times Q_2 \\ s &= (s_1, s_2) \\ A &= A_1 \times A_2 \\ \delta(q, a) &= \delta_1(q_1, a) \times \delta_2(q_2, a) \end{aligned}$$

■

9. Binary strings that contain at least *two* of the substrings `11`, `101`, and `1001`.

**Solution:** We apply a product construction to three NFAs, each of which accepts strings with one of the three substrings.

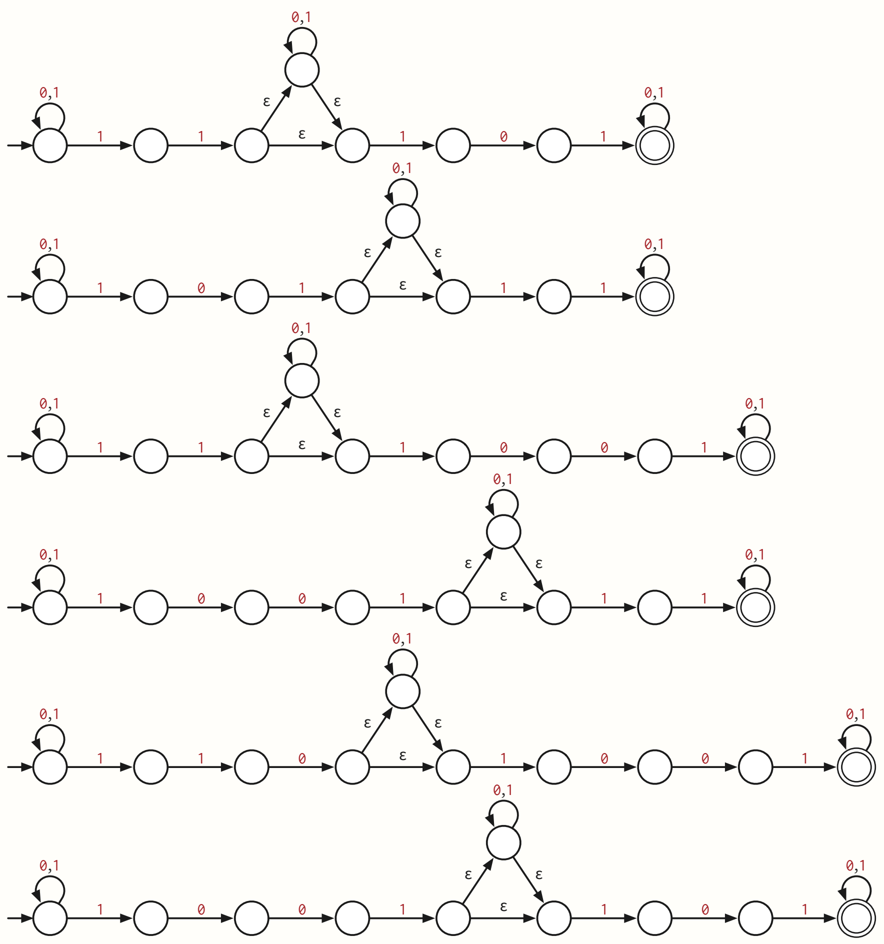


The product of the three NFAs $M_1 = (Q_1, s_1, A_1, \delta_1)$, $M_2 = (Q_2, s_2, A_2, \delta_2)$, and $M_3 = (Q_3, s_3, A_3, \delta_3)$ is defined identically to the product of three DFAs, except for the transition function. The choice of accepting states is specific to this language.

$$\begin{aligned} Q &= Q_1 \times Q_2 \times Q_3 \\ s &= (s_1, s_2, s_3) \\ A &= (A_1 \times A_2 \times Q_3) \cup (A_1 \times Q_2 \times A_3) \cup (Q_1 \times A_2 \times A_3) \\ \delta(q, a) &= \delta_1(q_1, a) \times \delta_2(q_2, a) \times \delta_3(q_3, a) \end{aligned}$$

■

**Solution (the hard way):** The following brute-force NFA with multiple start states guesses which two substrings the input string contains, in which order, and whether the substrings overlap.



Yeeeeeah. Please don't do this, especially on exams. ■