For $i \in \{1, 0, 13\}$, $q_i = \#(x) - \#_0(x) = i$

for consumed input $x$

$b_{bad} = \text{existence of a prefix } y \text{ (in consumed input)}$

with $|\#(y) - \#_0(y)| > 1$

(violating the desired condition)

$q_1$ and $q_2$: for the machine to nondeterministically enter the "having encountered first 0" and "having encountered second 0" states, respectively — with a separation of length $5k$ for some $k \geq 0$.

$p_1, p_2, \ldots, p_5$: account for the length factor of 5.
(c) Consider no following stepwise approach to the solution.

1. We show that finite automata can be used to perform "left-to-right" computation.

   For example, we can construct an FA accepting the language
   \[ \{ x \in \{a,b,c\}^* \mid \overrightarrow{x} = b \} \]
   left-to-right multiplication
   yielding \( b \)

   We can use the states to remember the left-to-right multiplication yield on the consumed input so far:

   \[
   \begin{align*}
   \text{start} & \rightarrow (\text{start}) \rightarrow (a) \rightarrow (a) \rightarrow (b) \rightarrow (c)
   \end{align*}
   \]

   state \( (b) \) is an accepting state
   Since we desire \( \overrightarrow{x} = b \).

   \( H \) is set of \( H \) de \( \{a,b,c\} \) \( \delta(q, a) = q \cdot d \)

2. Now we show that finite automata can be used to perform "right-to-left" computation.

   For an example similar to 1:
   \[ \{ x \in \{a,b,c\}^* \mid \overleftarrow{x} = b \} \]
   right-to-left multiplication
   yielding \( b \)

   We "reverse" our computation, instead of following the multiplication table (for \( \cdot \)) "deterministically," we start with our initial guess for the overall right-to-left multiplication on the entire input, and for each symbol scanned, we refine/adjust our guess for the right-to-left multiplication on the unconsumed input.
\[
\text{guess } b \text{ is the start state since we desire } x = b
\]

From the state \( \text{guess } b \):

- Transition(s) on \( a \):
  - Solving \( b = a \cdot \theta \) for \( \theta \)
    - (no solution)

- Transition(s) on \( b \):
  - Solving \( b = b \cdot \theta \) for \( \theta \)
    - (one solution, \( \theta = a \))
    - Adjust the guess to \( \text{guess } a \)

- Transition(s) on \( c \):
  - Solving \( b = c \cdot \theta \) for \( \theta \)
    - (one solution, \( \theta = c \))

Similarly, we can determine transitions from other guess states.

But that is not all. The transitions above are for normal processing for right-to-left multiplications. How should we halt when there is no more multiplication (that is, exactly one symbol left)?

\[
\text{start} \rightarrow \text{guess } b \rightarrow \text{accept}
\]

- Transition(s) on \( a \):
  - \( \text{guess } a \)
  - \( \text{guess } c \)

To make sure "exactly one symbol left", before entering \( \text{accept} \), there should be no transition out of \( \text{accept} \).
3. Finally, to construct an FA to accept the given language, we superimpose the ideas in 1 and 2.

Not attempting to optimize the state structure, we can:

\[ \text{structure 1:} \]

\[
\begin{align*}
\text{start} & \xrightarrow{\varepsilon} \text{guess} = a \\
& \xrightarrow{\varepsilon} \text{guess} = b \\
& \xrightarrow{\varepsilon} \text{guess} = c
\end{align*}
\]

\[ \text{structure 2:} \]

\[
\begin{align*}
\text{guess (and remember) the left-to-right (guess) and right-to-left (guess) multiplication yields}
\end{align*}
\]

Structure 1 - Using 1 and 2, to compute the current left-to-right multiplication, yield and update the current guess on the right-to-left multiplication yields

Structures 2 and 3 are similar.
4. We prove the statement by contradiction. Suppose that there exists a DFA $M$ accepting $L$, say $M = (Q, \Sigma, \delta, q_0, F)$ with $L(M) = L$.

Assume that $M$ is an $n$-state machine for some $n \geq 1$.

Consider the following $n+1$ strings over $\Sigma = \{0, 1, \lambda\}$, each of which is an input to the DFA $M$ (the $n+1$ "copies" of $M$):

$$x_1 = 0^i, \quad x_2 = 0^i, \quad \ldots, \quad x_{n+1} = 0^i$$

We question the ending states of these $n+1$ DFA $M$ after consuming their individual inputs.

Since $|Q| = n$ and we have $n+1$ strings $x_1, x_2, \ldots, x_{n+1}$, by the Pigeonhole Principle, there exist $i, j \leq 1, 2, \ldots, n+1$ with $i \neq j$ such that $x_i = 0^i$ and $x_j = 0^i$.

"drive" the DFA $M$ from $q_0$ to the same state, say $p$.

Now, consider the scenario for the two input strings $x_i = 0^i10^i$ and $x_j = 0^j10^j$.

To the DFA $M$, these two strings "drive" the DFA $M$ from $q_0$ to $p$, and from $p$ to the same state, say $p'$ — since $M$ is deterministic, the two transitions (on the two copies of $M$) are identical on the same suffix $0^i$.

But, $0^i10^i \in L$, so $p' \in F$.

while $0^j10^j \notin L$, so $p' \notin F$ { a contradiction.}
5. (c) \[ M \to M_1 : \]

Follow the lecture notes: Proving the equivalence between NFA-\(E\) and NFA:

- Compute \(E\)-closures for states \(s\) in \(M\) and \(E\)-closures for subsets \(\mathcal{S}\) of states \(s\) in \(M\).

- Relate the transition function \(s_1 : M_1 \to M_1\) to the transition function \(s_2 : M_2 \to M_2\) of \(M_2\) and \(E\)-closures involved.

6. (a) Assume that \(L\) is a regular language, that is, \(L = \text{DFA} M = (Q, \Sigma, \delta, q_0, F)\) with \(L(M) = L\).

Consider the V-NFA \(N\) defined by \(N = (Q, \Sigma, \delta', q_0, F)\) where \(\delta'(q, a) = \{ \delta(q, a) \}\) for all \(q \in Q\) and \(a \in \Sigma\).

Note that for each string \(w \in \Sigma^*\), there is only one possible computation path in the DFA. Thus, we have \(L(N) = L\) as desired.

(b) Let \(N = (Q, \Sigma, \delta, q_0, F)\) be an \(V\)-NFA. It suffices to show that we can convert \(N\) to a DFA \(M = (Q, \Sigma, \delta_M, q_0, F_M)\) such that \(L(N) = L(M)\). To do this, we use the Subset Construction for converting an NFA to a DFA. All the steps are essentially the same, except that now we have \(F_M\) is the set of non-empty subsets \(S \neq \emptyset\) such that \(S \subseteq F\).