1. **(Operations on languages)**

   1.1. $\textsc{Cycle}(L)$ consists of strings formed by taking a string in L, breaking it into two pieces, and concatenating them in reverse order. For $A = \{0^n1^n \ : n > 0\}$, we can perform this process on a string 0^n1^n by either breaking it in the left half (in the 0's) or in the right half (in the 1's), giving us $0^k1^n0^{n-k}$ or $1^k0^n1^{n-k}$, respectively. We can thus write

   $$\textsc{Cycle}(A) = \{0^k1^n0^{n-k} : 0 \leq k \leq n\} \cup \{1^k0^n1^{n-k} : 0 \leq k \leq n\}.$$

   1.2. The condition that $\textsc{Min}(L) = \textsc{Max}(L) = L$ is equivalent to the requirement that for any two distinct strings in L, neither is a prefix of the other. Some (infinite) examples of such languages are $\{0^n1 : n \geq 0\}$, $\{1^n0^n : n > 0\}$.

   1.3. No. Suppose, to get a contradiction, that we have a language L over some alphabet Σ such that $\textsc{Min}(L) = \Sigma^*$. Let $a \in \Sigma$. Then the strings $a, aa \in \textsc{Min}(L) \subset L$. But a is a proper prefix of aa, so by definition aa cannot be in $\textsc{Min}(L)$, giving us our contradiction.

   1.4. $\textsc{Half}(L)$ consists of the first halves of all even-length strings in L. For a string of the form $0^p1^q0^r$, where $q = p + r$ so that in particular it has even length $2q$, this is just 0^p1^r. We can therefore write

   $$\textsc{Half}(L) = \{0^p1^r : p, r \geq 0\}$$

   1.5. $\textsc{HalfPalindrome}(L)$ consists of the first halves of all even-length palindromes in L. In other words, $\textsc{HalfPalindrome}(L) = \textsc{Half}(L')$, where $L' \subset L$ is the subset of all palindromes in L (all strings in L already have even length, as we saw above). But L' is just $\{0^p1^q0^p : q = 2p\}$, so we have

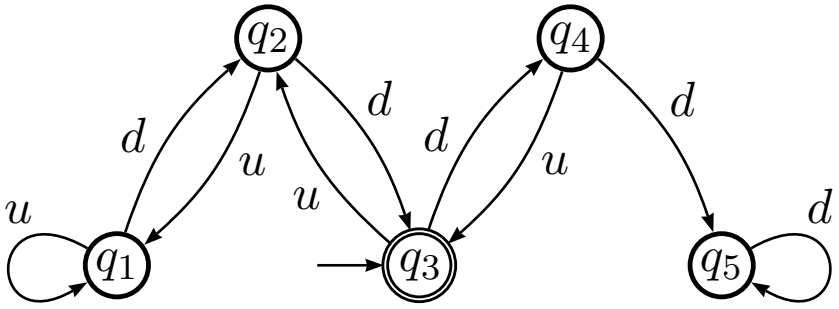
   $$\textsc{HalfPalindrome}(L) = \{0^p1^p : p \geq 0\}$$

2. **(Tables $\leftrightarrow$ Diagrams)**

   2.1. $M_1 = (Q, \Sigma, \delta, q_1, F)$, where
   $Q = \{q_1, q_2, q_3\}$, $\Sigma = \{a, b\}$, $F = \{q_2\}$, and
   $\delta(q_1, a) = q_2, \delta(q_1, b) = q_1$,
   $\delta(q_2, a) = q_3, \delta(q_2, b) = q_3$,
   $\delta(q_3, a) = q_2, \delta(q_3, b) = q_1$.

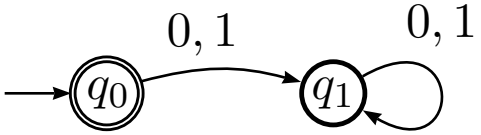
   $M_2 = (Q, \Sigma, \delta, q_1, F)$, where
   $Q = \{q_1, q_2, q_3, q_4\}$, $\Sigma = \{a, b\}$, and
   $\delta(q_1, a) = q_1, \delta(q_1, b) = q_2$,
   $\delta(q_2, a) = q_3, \delta(q_2, b) = q_4$,
   $\delta(q_3, a) = q_2, \delta(q_3, b) = q_1$,
   $\delta(q_4, a) = q_3, \delta(q_4, b) = q_4$.

2.2.

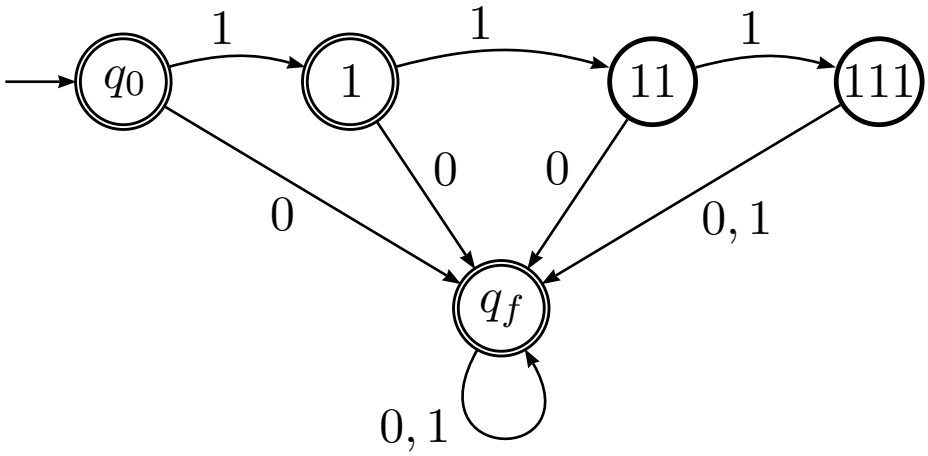


3. **(Designing DFAs)**

3.1. If the empty string is to be accepted, the start state must be an accept state. Moreover, any symbol read should take the machine to a permanent reject state. Both these requirements are captured by the following DFA.

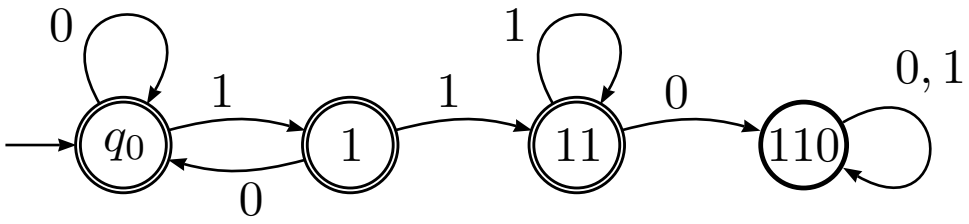


3.2. We want to keep track of the first three symbols on the input string in only so much as there is still a possibility of the string being 11 or 111. Thus, if we see either a 0 or a fourth symbol, we move to a permanent accept state, since the string is certainly neither 11 nor 111. Following this intuition, we only need five states, as follows.



3.3. Essentially, we want to keep track of the last three symbols seen, say, $\ldots xyz$, but again in only so much as they may be part (or all) of the string 110. More precisely, we want to keep

track of any suffix of xyz that is also a prefix of 110. That way, we would never miss a 110 that comes by, and can simply move to a permanent reject state if we see it. In the following DFA, other than the start state q_0, state names indicate the longest suffix of xyz that is a prefix of 110.

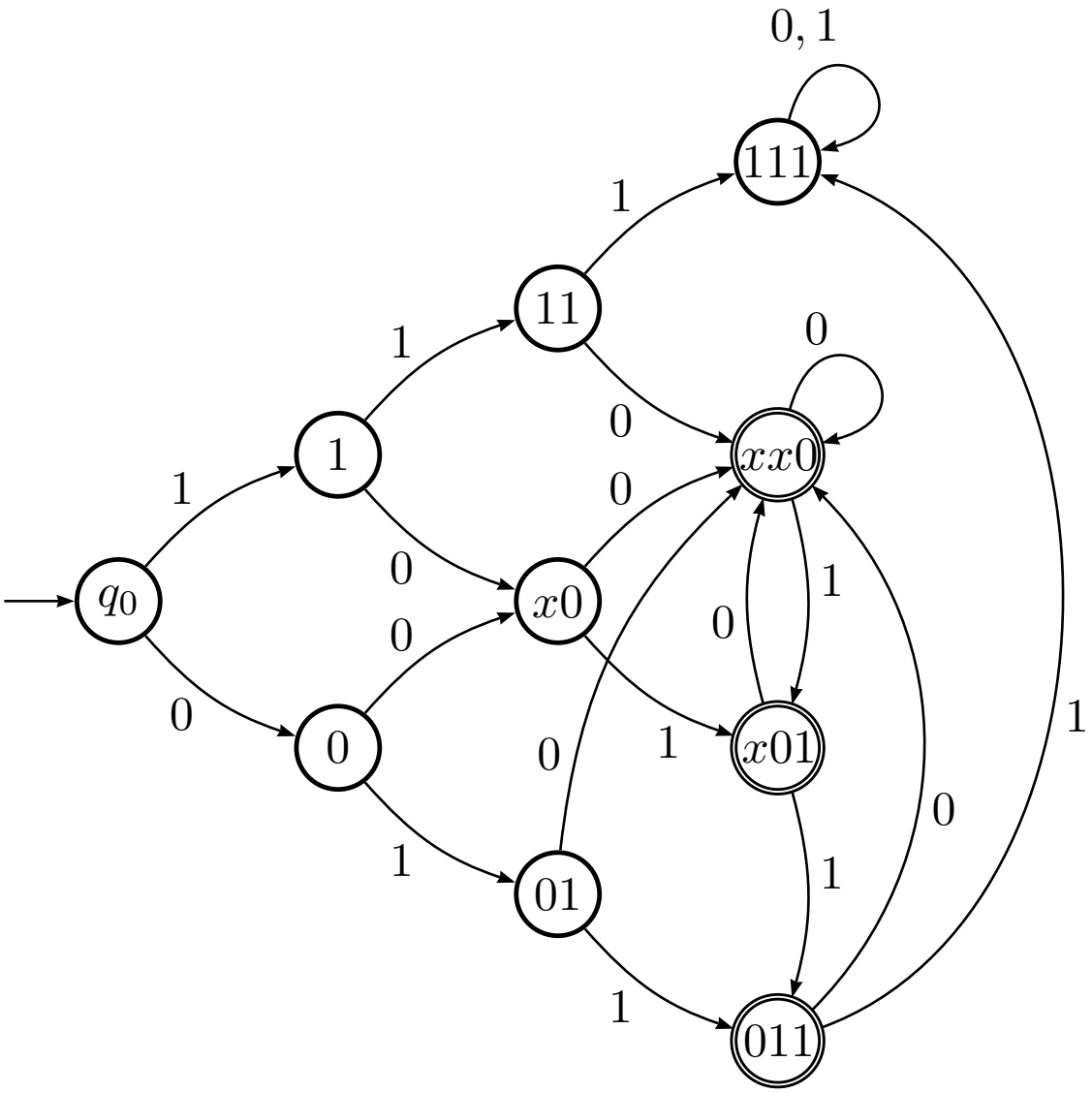


3.4. Here, it seems we really have no way out of keeping track of the precise last three symbols seen. In particular, we need to differentiate strings of length < 3 from those of length ≥ 3. We can formally define the following 15-state DFA.

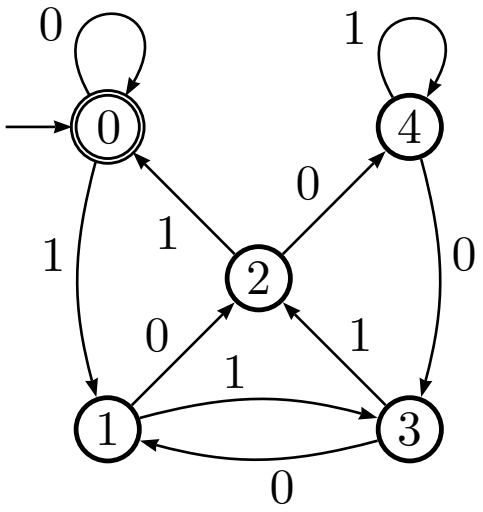
Let $M = (Q, \Sigma, \delta, q_0, F)$, where

$$\begin{aligned} Q &= \{q_x \in \Sigma^* : |x| \leq 3\}, \\ \Sigma &= \{0, 1\}, \\ q_0 &= q_\epsilon, \\ F &= \{q_x \in Q : |x| = 3, x \neq 111\}, \text{ and, for any } q_x \in Q, a \in \Sigma, \\ \delta(q_x, a) &= \begin{cases} q_{xa} & \text{if } |x| < 3 \\ q_{x_2 x_3 a} & \text{if } q_x \in F, \text{ where } x_i \text{ is the } i^{th} \text{ symbol in } x \\ q_x & \text{otherwise} \end{cases} \end{aligned}$$

However, if we do not go for generality, and instead observe that our machine need only be looking for the string 111, the problem reduces to one very similar to Problem 3.3, and like 3.3 can be solved in only four states. Of course, we need several more states to keep track of the string before it reaches length three, but even this part can be reduced to six states (instead of the default seven). We end up with a 10-state DFA, depicted below.



3.5. The key idea is that if suffices to keep track of the input (as a binary number) *modulo* 5. If we are in state 0, then the input is divisible by 5, so we accept. Transitions are consistently defined owing to the fact given in the hint, namely, that $(mn+p) \bmod 5 = ((m \bmod 5) \cdot n + p) \bmod 5$. If we've seen (binary number) m so far, then seeing an additional symbol p would give us a new value of $2m+p$. That is, apply the hint with $n = 2$, $p \in \{0, 1\}$ the next symbol seen, and m the numerical value of the string seen so far. We thus get the following DFA.



3.6. Generalizing from the formal definition of the (unsimplified) DFA from Problem 3.4, we immediately get the nearly identical machine below.

Let $M = (Q, \Sigma, \delta, q_0, F)$, where

$$
\begin{aligned}
Q &= \{q_x \in \Sigma^* : |x| \leq 100\}, \\
\Sigma &= \{0, 1\}, \\
q_0 &= q_\epsilon, \\
F &= \{q_x \in Q : |x| = 100, x \text{ contains at least ten 0's}\}, \text{ and} \\
\delta(q_x, a) &= \begin{cases} q_{xa} & \text{if } |x| < 100 \\ q_{x_2 x_3 \ldots x_{100} a} & \text{if } q_x \in F, \\ q_x & \text{otherwise} \end{cases}
\end{aligned}
$$