1. For each of the following languages over the alphabet $\{0,1\}^*$, describe an equivalent regular expression, and briefly explain why your regular expression is correct. There are infinitely many correct answers for each language.

Rubric: 2½ points for each part = 1½ for correctness + 1 for English explanation (standard regular expression rubric, scaled and rounded). Every subproblem has infinitely many correct solutions!

(a) All strings that start with 1, that end with 1, and whose length is not a multiple of 3.

Solution: Call this language L_a. Except for the length-one string 1, Every string in L_a has the form $1x1$ for some string x such that $(|x| \bmod 3) \neq 1$.

$$\begin{aligned} L_a = {} & 1 \\ & + 1\big((0+1)(0+1)(0+1)\big)^* 1 && |x| \bmod 3 = 0 \\ & + 1(0+1)(0+1)\big((0+1)(0+1)(0+1)\big)^* 1 && |x| \bmod 3 = 2 \end{aligned}$$

More compactly, we have $L_a = 1 + 1\big(\varepsilon + (0+1)(0+1)\big)\big((0+1)(0+1)(0+1)\big)^* 1$. ■

(b) All strings in which each substring 000 is immediately followed by an odd-length run of 1s.

Solution: A *run* in a string is a maximal substring in which all symbols are equal; *every* binary string can be partitioned into alternating runs of 0s and 1s. Thus, we can represent the set of *all* binary strings as follows:

$$(0+1)^* = \underbrace{1^*}_{\text{optional run of 1s}} \cdot \Big(\underbrace{00^*}_{\text{run of 0s}} \cdot \underbrace{11^*}_{\text{run of 1s}} \Big)^* \cdot \underbrace{0^*}_{\text{optional run of 0s}}$$

Our target language L_b contains a binary string w if and only if w satisfies the following constraints on its runs of 0s:

- Every run of 0s in w has length *at most* 3.
- Each run of 0s of length 3 is immediately followed by a run of 1s with odd length. Thus, if w ends with a run of 0s, that run has length at most 2. The set of all odd-length runs of 1s is described by the regular expression $1(11)^*$.
- Each run of 0s with length 1 or 2 can either end the string or be followed by an arbitrary run of 1s.

We conclude:

$$L_b = \underbrace{1^*}_{\text{optional run of 1s}} \cdot \Big(\underbrace{0 \cdot 11^* \;+\; 00 \cdot 11^* \;+\; 000 \cdot 1(11)^*}_{\text{run of 0s followed by run of 1s}} \Big)^* \cdot \underbrace{(\varepsilon + 0 + 00)}_{\text{optional run of 0s}}$$

■

(c) All strings that contain both 010 and 101 as *subsequences*, but do not contain either 010 or 101 as *substrings*. *[Hint: This is simpler than it looks; think about runs.]*

Solution: A binary string w contains both 010 and 101 as *subsequences* if and only if w consists of at least four runs. On the other hand, a binary string w avoids 010 and 101 as *substrings* if and only if every run in w, except possibly the first and last run, has length at least 2.

Let's call this target language L_c. To build a regular expression for L_c, we consider four cases, depending on whether the first and last runs contain 0s or 1s. If the first and last runs use the same symbol, the string must have an odd number of runs, and therefore at least five runs.

$$\begin{aligned} L_c = {} & 00^* \cdot 111^* \cdot 000^* \cdot (111^* \cdot 000^*)^* \cdot 111^* \cdot 00^* \\ & + 00^* \cdot 111^* \cdot 000^* \cdot (111^* \cdot 000^*)^* \cdot 11^* \\ & + 11^* \cdot 000^* \cdot 111^* \cdot (000^* \cdot 111^*)^* \cdot 00^* \\ & + 11^* \cdot 000^* \cdot 111^* \cdot (000^* \cdot 111^*)^* \cdot 000^* \cdot 11^* \end{aligned}$$

Or more concisely:

$$\begin{aligned} L_c = {} & 00^* \cdot 111^* \cdot 000^* \cdot (111^* \cdot 000^*)^* \cdot 11^*(\varepsilon + 100^*) \\ & + 11^* \cdot 000^* \cdot 111^* \cdot (000^* \cdot 111^*)^* \cdot 00^*(\varepsilon + 011^*) \end{aligned}$$

■

(d) All strings ***over the alphabet*** **{0, 1, 2}** in which every pair of adjacent symbols differs by exactly 1.

Solution: Every string in this language alternates between 1s and even symbols.

$$(\varepsilon + 0 + 2) \cdot (1(0 + 2))^* \cdot (\varepsilon + 1)$$

■

*(e) All strings ***over the alphabet {0, 1, 2, 3}*** in which every pair of adjacent symbols differs by exactly 1.

Solution: Let's call this language L_e. Just like in part (d), we'll use 1s as signposts; the definition of L_e constrains what substrings can appear before, after, and between 1s. Our regular expression has the following high-level structure:

$$L_e = \langle \textit{no 1s} \rangle \;+\; \langle \textit{before first 1} \rangle \cdot \big(1 \cdot \langle \textit{between 1s} \rangle\big)^* \cdot 1 \cdot \langle \textit{after last 1} \rangle$$

The substring between two consecutive 1s is either a single 0 or a substring of alternating 2s and 3s that begins and ends with 2.

$$\langle \textit{between 1s} \rangle \;=\; 0 + 2(32)^*$$

The prefix before the first 1 is either empty, a single 0, or a string of alternating2s and 3s that *ends* with 2.

$$\langle \textit{before first 1} \rangle \;=\; 0 + (32)^* + (23)^*2$$

The suffix after the last 1 is either empty, a single 0, or a string of alternating 2s and 3s that *begins* with 2.

$$\langle \textit{after last 1} \rangle \;=\; 0 + (23)^* + 2(32)^*$$

Finally, a string in L_e with no 1s at all is either empty, a single 1, or alternating 2s and 3s.

$$\langle \textit{no 1s} \rangle \;=\; 0 + (23)^* + (32)^* + (23)^*2 + (32)^*3$$

Putting all the pieces together, we obtain our final regular expression:

$$\begin{aligned} L_e = {} & 0 + (23)^* + (32)^* + (23)^*2 + (32)^*3 \\ & + \big(0 + (32)^* + (23)^*2\big) \cdot \big(1 \cdot (0 + 2(32)^*)\big)^* \cdot 1 \cdot \big(0 + (23)^* + 2(32)^*\big) \end{aligned}$$

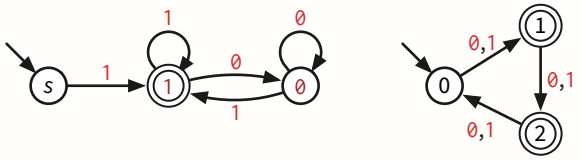
■

2. For each of the following languages over the alphabet $\{0, 1\}$, describe a DFA that accepts the language, and briefly describe the purpose of each state. You can describe your DFA using a drawing, or using formal mathematical notation, or using a product construction; see the standard DFA rubric.

Rubric: 10 points = 2½ points for each part = 1½ for correctness + 1 for English explanation of states (standard DFA rubric, scaled and rounded). Again, every subproblem has infinitely many correct solutions. These solutions are more detailed than necessary for full credit.

(a) All strings that start with 1, that end with 1, and whose length is not a multiple of 3.

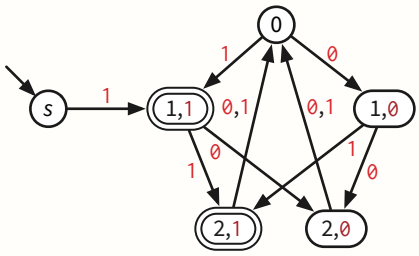
Solution (product): We construct the product of two DFAs:



- The first DFA accepts all strings that start and end with 1. Non-start states are labeled with the last symbol read. Missing transitions lead to a hidden dump state.
- The second DFA accepts all strings whose length is not divisible by 3. States are labeled with the number of symbols read so far, modulo 3.

The accepting states of the product DFA are $(1, 1)$ and $(1, 2)$. ■

Solution (explicit drawing):

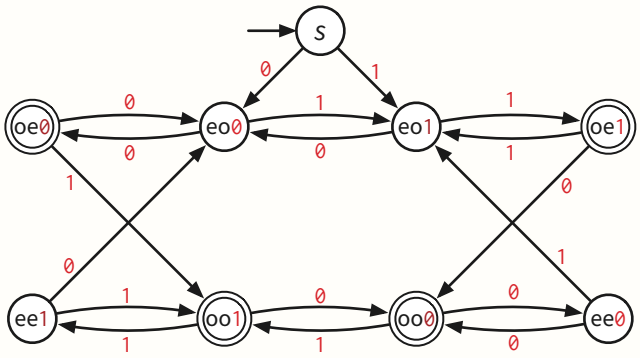


All missing transitions lead to a hidden dump state. States are labeled as follows:

- s is the start state; no symbols have been read.
- 0 — The first input symbol was 1, and we've read 0 mod 3 symbols.
- $1, 0$ — The first input symbol was 1, we've read 1 mod 3 symbols, and the last symbol read was 0.
- $1, 1$ — The first input symbol was 1, we've read 1 mod 3 symbols, and the last symbol read was 1.
- $2, 0$ — The first input symbol was 1, we've read 2 mod 3 symbols, and the last symbol read was 0.
- $2, 1$ — The first input symbol was 1, we've read 2 mod 3 symbols, and the last symbol read was 1. ■

(b) All strings that contain an odd number of even-length runs.

Solution:



Except for the start state, each state is labeled with three symbols:

- The first symbol indicates whether *the number of even-length runs* we have read so far is even (e) or odd (o). The accepting states are precisely the states whose labels start with o.
- The second symbol indicates whether *the length of the last run* we have read so far is even (e) or odd (o).
- The third symbol indicates *the last symbol* we have read so far.

The transitions from non-start states obey the following rules:

- If we read a symbol that is equal to its predecessor, the first and second state components each flip from e to o or vice versa.
- If we read a symbol that is different from its predecessor, the first state component doesn't change, and the second state component becomes o.
- The last component of the state always matches the last symbol read.

■

(c) All strings of length at least 8, whose last eight symbols contain an even number of 1s.

Solution: We formally define a DFA (Q, s, A, δ) over the alphabet $\Sigma = \{0, 1\}$ as follows:

$$Q = \left\{ w \in \Sigma^* \;\middle|\; |w| \le 8 \right\}$$

$$s = \varepsilon$$

$$A = \left\{ w \in \Sigma^8 \;\middle|\; \#(1, w) \text{ is even} \right\}$$

$$\delta(w, a) = \begin{cases} wa & \text{if } |w| < 8 \\ xa & \text{if } w = bx \text{ for some } b \in \Sigma \text{ and } x \in \Sigma^7 \end{cases}$$

Each state stores the last eight symbols read, or all symbols read if that number is less than 8. There are a total of $2^0 + 2^1 + \cdots + 2^8 = 511$ states. ■

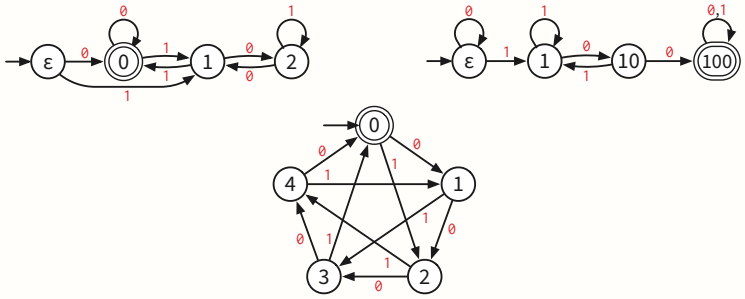
(d) All strings w that satisfy *exactly two* of the following conditions:

i. w is the binary representation of a multiple of 3, possibly with leading `0`s.

ii. w contains the substring `100`

iii. $\#(\texttt{0}, w) + 2 \cdot \#(\texttt{1}, w)$ is a multiple of 5.

Solution (product construction): We construct the product M of *three* DFAs.



- The first DFA M_1 recognizes binary representations of multiples of 3. Each state other than the start state stores the binary value modulo 3 of the bits read so far.
- The second DFS M_2 recognizes all strings that contain the substring `100`. The first three states are labeled with the longest suffix of the bits read so far that is also a prefix of `100`; only strings that do not contain `100` can reach these states.
- The third DFA M_3 recognizes strings w such that $\#(\texttt{0}, w) + 2 \cdot \#(\texttt{1}, w)$ is a multiple of 5. Each state is labeled with the value of $\#(\texttt{0}, w) + 2 \cdot \#(\texttt{1}, w)$ mod 5, where w is the prefix of bits read so far.

The accepting states of our particular three-way product DFA are all states (q_1, q_2, q_3) such that exactly two of q_1, q_2, q_3 are accepting states in their respective machines. More formally, we have

$$A = \big((A_1 \times A_2 \times Q_3) \cup (A_1 \times Q_2 \times A_3) \cup (Q_1 \times A_2 \times A_3)\big) \setminus (A_1 \times A_2 \times A_3)$$

The product of *any* three DFAs $M_1 = (Q_1, s_1, A_1, \delta_1)$, $M_2 = Q_2, s_2, A_2, \delta_2)$, and $M_3 = (Q_3, s_3, A_3, \delta_3)$ is defined as one would expect:

$$Q = Q_1 \times Q_2 \times Q_3$$
$$s = (S_1, s_2, s_3)$$
$$\delta((q_1, q_2, q_3), a) = \big(\delta_1(q_1, a), \delta_2(q_1, a), \delta_3(q_1, a)\big)$$

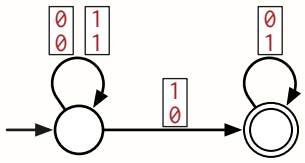
Equivalently, the three-way product is the product of M_1 with the product of M_2 and M_3. The accepting states of the three-way product depend on the target language, just as they do for products of two machines. ■

Rubric: No penalty for interpreting ε as a valid binary representation of 0. No explanation of the three-way product construction is required, but the solution must explicitly describe the accepting states.

3. ***Practice only. Do not submit solutions.***

(a) Describe a DFA that accepts the language $L_{+1} = \{w \in \Sigma_2^* \mid hi(w) = lo(w) + 1\}$.

Solution:



- state 0: $hi(w) = lo(w)$
- state 1: $hi(w) = lo(w) + 1$

Missing transitions lead to a hidden dump state. ■

(b) Describe a regular expression for L_{+1}.

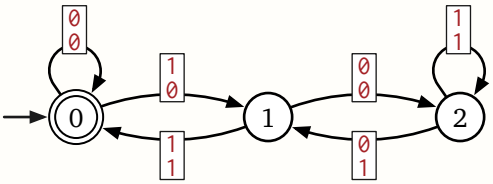
Solution: $\left(\left[\begin{smallmatrix}0\\0\end{smallmatrix}\right] + \left[\begin{smallmatrix}1\\1\end{smallmatrix}\right]\right)^* \left[\begin{smallmatrix}1\\0\end{smallmatrix}\right] \left[\begin{smallmatrix}0\\1\end{smallmatrix}\right]^*$ ■

(c) Describe a DFA that accepts the language $L_{\times 3} = \{w \in \Sigma_2^* \mid hi(w) = 3 \cdot lo(w)\}$.

Solution: We start with one state for each possible value of the difference $\Delta(w) = hi(w) - 3 \cdot lo(w)$, where w is the string we have read so far. It follows that

$$\delta(\Delta, \left[\begin{smallmatrix}a\\b\end{smallmatrix}\right]) = 2\Delta + a - 3b = \begin{cases} 2\Delta & \text{if } a = 0 \text{ and } b = 0 \\ 2\Delta + 1 & \text{if } a = 1 \text{ and } b = 0 \\ 2\Delta - 3 & \text{if } a = 0 \text{ and } b = 1 \\ 2\Delta - 2 & \text{if } a = 1 \text{ and } b = 1 \end{cases}.$$

In particular, $2\Delta - 3 \le \delta(\Delta, \left[\begin{smallmatrix}a\\b\end{smallmatrix}\right]) \le 2\Delta + 1$. Thus, if Δ is ever negative, it will stay negative forever, and if Δ is ever greater than 2, then Δ will stay greater than 2 forever. So we can collapse all states $\Delta < 0$ and $\Delta > 2$ into a single junk state, leaving us only three interesting states.



■

(d) Describe a regular expression for $L_{\times 3}$.

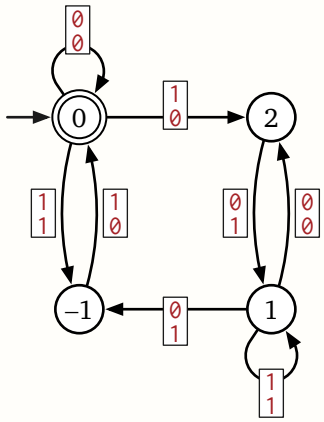
Solution: $\left(\left[\begin{smallmatrix}0\\0\end{smallmatrix}\right] + \left[\begin{smallmatrix}1\\0\end{smallmatrix}\right] \left(\left[\begin{smallmatrix}0\\0\end{smallmatrix}\right] \left[\begin{smallmatrix}1\\1\end{smallmatrix}\right]^* \left[\begin{smallmatrix}0\\1\end{smallmatrix}\right]\right)^* \left[\begin{smallmatrix}1\\1\end{smallmatrix}\right]\right)^*$ ■

*(e) Describe a DFA that accepts the language $L_{\times 3/2} = \{w \in \Sigma_2^* \mid 2 \cdot hi(w) = 3 \cdot lo(w)\}$.

Solution: We start with one state for each possible value of the difference $\Delta(w) = 2 \cdot hi(w) - 3 \cdot low$, where w is the string we have read so far.

$$\delta(\Delta, \left[\begin{smallmatrix} a \\ b \end{smallmatrix}\right]) = 2\Delta + 2a - 3b = \begin{cases} 2\Delta & \text{if } a = 0 \text{ and } b = 0 \\ 2\Delta + 2 & \text{if } a = 1 \text{ and } b = 0 \\ 2\Delta - 3 & \text{if } a = 0 \text{ and } b = 1 \\ 2\Delta - 1 & \text{if } a = 1 \text{ and } b = 1 \end{cases}.$$

In particular, we have $2\Delta - 3 \le \delta(\Delta, \left[\begin{smallmatrix} a \\ b \end{smallmatrix}\right]) \le 2\Delta + 2$. No state $\Delta \le -2$ of $\Delta \ge 3$ can ever reach state 0, so we can collapse all such states into a single junk state, leaving us with only four interesting states.



This DFA implies the following regular expression for $L_{\times 3/2}$:

$$L_{\times 3/2} = \left(\left[\begin{smallmatrix} 0 \\ 0 \end{smallmatrix}\right] + \left[\begin{smallmatrix} 1 \\ 1 \end{smallmatrix}\right]\left[\begin{smallmatrix} 1 \\ 0 \end{smallmatrix}\right] + \left[\begin{smallmatrix} 1 \\ 0 \end{smallmatrix}\right]\left(\left[\begin{smallmatrix} 0 \\ 1 \end{smallmatrix}\right]\left[\begin{smallmatrix} 1 \\ 1 \end{smallmatrix}\right]^*\left[\begin{smallmatrix} 0 \\ 0 \end{smallmatrix}\right]\right)^*\left[\begin{smallmatrix} 0 \\ 1 \end{smallmatrix}\right]\left[\begin{smallmatrix} 1 \\ 0 \end{smallmatrix}\right]\right)^*$$

---

*Similar constructions imply that for **any** integers a, b, c, the language $\{w \in \Sigma_2^* \mid a \cdot \mathrm{hi}(w) = b \cdot \mathrm{lo}(w) + c\}$ is regular.* ■