1. For each of the following languages assume that the alphabet is $\{\mathtt{0}, \mathtt{1}\}$ unless it is specified otherwise. Give a regular expression that describes that language, and **briefly argue why your expression is correct**.

   (a) All strings that end in $\mathtt{010}$ *and* contain $\mathtt{000}$ as a substring.

   Solution: $\boxed{(\mathtt{0}+\mathtt{1})^*\mathtt{000}(\mathtt{0}+\mathtt{1})^*\mathtt{010} + (\mathtt{0}+\mathtt{1})^*\mathtt{00010}}$

   The first part is the case where the $\mathtt{000}$ substring and the $\mathtt{010}$ at the end occur separately. The second part covers the case where they overlap, i.e. when the $\mathtt{00010}$ at the end contributes to both the $\mathtt{000}$ substring and the $\mathtt{010}$ termination. ■

   Rubric: 2 points: Scaled regular expression rubric (10 point regular expression rubric at the end of the solutions).

   (b) All strings that do not contain the *subsequence* $\mathtt{010}$.

   Solution: $\boxed{\mathtt{1}^*\mathtt{0}^*\mathtt{1}^*}$

   For any given string w, we can organize the string into alternating blocks (of length > 0) where each block is comprised of consecutive $\mathtt{1}$s or $\mathtt{0}$s. If there are two blocks of $\mathtt{0}$, they must be separated by a block of $\mathtt{1}$, so our string would contain the subsequence $\mathtt{010}$. Thus the only way to avoid this subsequence is to have (at most) one block of $\mathtt{0}$, which can optionally have blocks of $\mathtt{1}$ on either side. ■

   Rubric: 2 points: Scaled regular expression rubric (10 point regular expression rubric at the end of the solutions).

   (c) The complement of the language $\{\mathtt{0}, \mathtt{01}, \mathtt{10}, \mathtt{101}, \mathtt{011}, \mathtt{111}\}$.

   Solution: $\boxed{\begin{aligned}\epsilon + \mathtt{1} + \mathtt{00} + \mathtt{11} + \mathtt{000} + \mathtt{001} + \mathtt{010} + \mathtt{100} + \mathtt{110} \\ +(\mathtt{1}+\mathtt{0})(\mathtt{1}+\mathtt{0})(\mathtt{1}+\mathtt{0})(\mathtt{1}+\mathtt{0})(\mathtt{1}+\mathtt{0})^*\end{aligned}}$

   The first 9 terms represent all strings of length 3 or less that are in the complement of the language. The final term represents all strings of length 4 or more (all of which are in the complement, since they are longer than any string in the initial language). ■

   Rubric: 2 points: Scaled regular expression rubric (10 point regular expression rubric at the end of the solutions). More succinct representations are possible, but not required for credit.

   (d) The language $\{\mathtt{0}^a\mathtt{1}^b\mathtt{2}^c \mid a, b, c \geq 0, a \equiv b + c \mod 2\}$ over the alphabet $\{\mathtt{0}, \mathtt{1}, \mathtt{2}\}$.

   Solution: $\boxed{\begin{aligned}(\mathtt{00})^*(\mathtt{11})^*(\mathtt{22})^* + \mathtt{0}(\mathtt{00})^*\mathtt{1}(\mathtt{11})^*(\mathtt{22})^* + \\ \mathtt{0}(\mathtt{00})^*(\mathtt{11})^*\mathtt{2}(\mathtt{22})^* + (\mathtt{00})^*\mathtt{1}(\mathtt{11})^*\mathtt{2}(\mathtt{22})^*\end{aligned}}$

   There are four possible cases to consider depending on the parity of a, b, and c: all even, a and b odd, a and c odd, and b and c odd. The four terms in the above regular expression cover each of these, respectively. ■

Rubric: 2 points: Scaled regular expression rubric (10 point regular expression rubric at the end of the solutions).

(e) Let r be a regular expression and let $L(r)$ be the language denoted by r. We want to remove the empty string from $L(r)$ and create a regular expression r' such that $L(r') = L(r) - \{\epsilon\}$.[1] One might try to "understand" $L(r)$ and write a regular expression r' from scratch for the new language, but this may be difficult if r is complicated and long, and doesn't give us a general procedure. Here we explore a recursive algorithmic procedure to obtain r' from r (in an automatic way) by considering the following ideas.

- For each of the base cases $\emptyset, \epsilon, \mathtt{0}, \mathtt{1}$ obtain r' assuming that r is one of them.

  Solution: The base cases are summarized in this table:

r	$\emptyset$	ϵ	$\mathtt{0}$	$\mathtt{1}$
r'	$\emptyset$	$\emptyset$	$\mathtt{0}$	$\mathtt{1}$

  For the case where $r = \epsilon$, $L(r) = \{\epsilon\}$, so we want $L(r') = \emptyset$. For all the other three base cases, ϵ is not in $L(r)$, so we can take $r' = r$.

  ■

- Suppose $r = r_1 + r_2$ and r'_1 and r'_2 are regular expressions for $L(r_1) - \{\epsilon\}$ and $L(r_2) - \{\epsilon\}$ respectively. How would you obtain r' from r'_1, r'_2, r_1, r_2? Note that there may be multiple equivalent expressions; you can give any correct one.

  Solution: $\boxed{r' = r'_1 + r'_2}$

  We need to allow r' to match any non-empty string from either $L(r_1)$ or $L(r_2)$.

  ■

- Suppose $r = r_1 r_2$. How would you obtain r' from r'_1, r'_2, r_1, r_2?

  Solution: $\boxed{r' = r'_1 r_2 + r_1 r'_2}$

  We need to allow r' to match any string obtained by concatenating a string in $L(r_1)$ with $L(r_2)$, as long as at least one of those two strings is non-empty.

  ■

- Suppose $r = (r_1)^*$. How would you obtain r' from r'_1, r_1?

  Solution: $\boxed{r' = r'_1 (r_1)^*}$

  We need to allow r' to match with the concatenation of any *non-zero* number of strings from $L(r_1)$, as long as at least one of those strings is non-empty.

  ■

Given the ideas above, describe a recursive algorithm to obtain r' from an arbitrary regular expression r. You do not need to justify the constructions explicitly but it is in your own interest to think about the correctness of each case.

Solution: If the regular expression falls into one of the base cases, we stop recursion and return the r' defined by that base case. Otherwise, we can break r into smaller expressions as $r_1 + r_2$, $r_1 r_2$, or $(r_1)^*$. We recurse on r_1 and (where it exists) r_2 to obtain r'_1 and (possibly) r'_2. Finally, we return r' as defined in the appropriate case above.

■

[1] It is not immediately obvious that $L(r) - \{\epsilon\}$ is regular but it turns out to be, and in fact we will end up giving a proof of that by solving this problem.

Rubric: 2 points. .25 points for each correct r' (four base cases and three inductive cases), and .25 points for correctly describing the recursive algorithm.

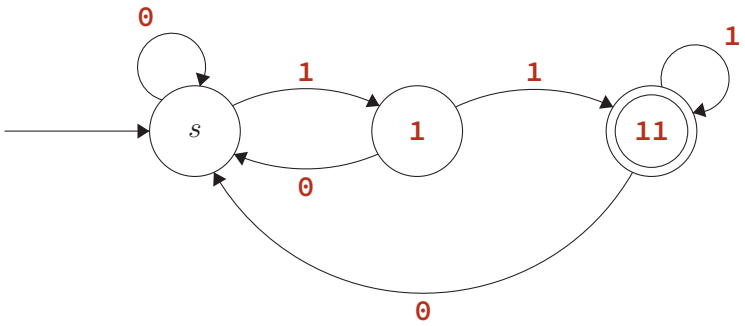
Rubric: Standard regular expression rubric. 10 points =

- 2 points for a syntactically correct regular expression.
- **Homework only**: 4 points for a brief English explanation of your regular expression. This is how you argue that your regular expression is correct.
  - For longer expressions, you should explain each of the major components of your expression, and separately explain how those components fit together.
  - We do not want a transcription; don't just translate the regular-expression notation into English.
- 4 points for correctness. (8 points on exams, with all penalties doubled)
  - -4 for incorrectly answering $\emptyset$ or Σ^*.
  - -1 for a single mistake: one typo, excluding exactly one string in the target language, or including exactly one string not in the target language. (The incorrectly handled string is almost always the empty string ϵ.)
  - -2 for incorrectly including/excluding more than one but a finite number of strings.
  - -4 for incorrectly including/excluding an infinite number of strings.
- Regular expressions that are more complex than necessary may be penalized. Regular expressions that are significantly too complex may get no credit at all. On the other hand, minimal regular expressions are not required for full credit.

2. DFA design. For the first three parts describe a DFA by drawing it explicitly and **briefly explain the *meaning* of each state**. For the last two do not draw the DFA. Instead use formal notation, **still briefly explaining the *meaning* of each state**.

   (a) All strings in $\{\mathtt{0}, \mathtt{1}\}^*$ that end in $\mathtt{11}$.

   Solution: Our DFA is as follows:



   Meaning of the states:

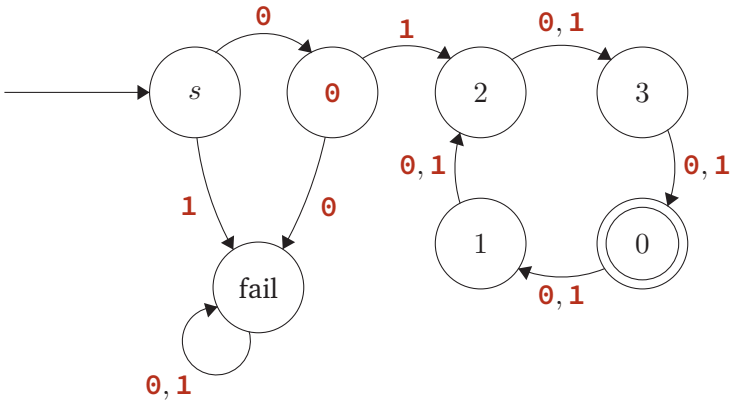
   - s: We have just read $\mathtt{0}$, or we haven't read anything yet.
   - $\mathtt{1}$: We have just read a $\mathtt{1}$, but the last two symbols read are not $\mathtt{11}$.
   - $\mathtt{11}$: The last two symbols read are $\mathtt{11}$.

■

Rubric: 2 points: Scaled DFA rubric (10 point DFA rubric at the end of the solutions).

(b) All strings in $\{0,1\}^*$ that start with 01 and whose length is divisible by 4.

Solution: Our DFA is as follows:



Meaning of the states:

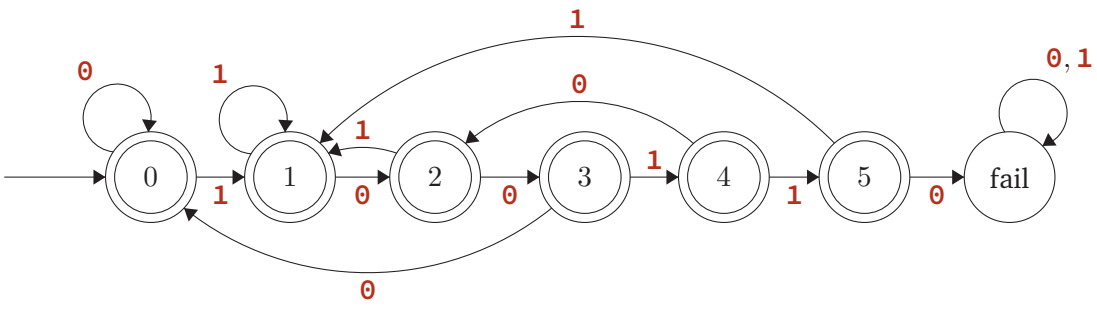
- s: start state.
- 0: we've read a 0 and nothing else.
- fail: the string starts with something besides 01 and hence cannot be in the language.
- o-3: the string started with 01, and the length of the string so far is $\ell \mod 4$, where ℓ is the label of the state.

[Note: an alternative solution is to use a product construction, though the number of states would be a little bigger.] ■

Rubric: 2 points: Scaled DFA rubric (10 point DFA rubric at the end of the solutions).

(c) All strings in $\{0,1\}^*$ that do *not* contain 100110 as a substring.

Solution: Here is our DFA:



Meaning of the states:

- 0-5: The last ℓ symbols read match the beginning of 100110 (where ℓ is the label of the state), but no larger number of symbols do.
- fail: We have seen the substring 100110, so cannot accept this string.

Note that it is easier to interpret the complement of this DFA: we are essentially writing down a DFA that accepts strings containing **100110**, and then changing the accept states. ■

Rubric: 2 points: Scaled DFA rubric (10 point DFA rubric at the end of the solutions).

(d) All strings whose 8th-to-last symbol is **1**, or equivalently, the set

$$\{x\mathbf{1}y \mid x \in \Sigma^* \text{ and } y \in \Sigma^7\}$$

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

$$\begin{aligned} Q &= \Sigma^8 \\ s &= \mathbf{00000000} \\ A &= \{\mathbf{1}x \mid x \in \{\mathbf{0}, \mathbf{1}\}^7\} \\ \delta(ax, b) &= xb \qquad \text{for all } a, b \in \Sigma \text{ and } x \in \Sigma^7 \end{aligned}$$

The label of each state represents the last eight symbols read. To avoid additional states that only deal with the first seven symbols in the input string and simplify notation, we pretend that we have read eight **0**s before the actual input string begins; this ensures that we will never accept any string of length shorter than 8. ■

Rubric: 2 points: Scaled DFA rubric (10 point DFA rubric at the end of the solutions).

(e) All strings w such that $(\#(\mathbf{0}, w) \mod 3) + (\#(\mathbf{1}, w) \mod 7) = (|w| \mod 4)$.

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

$$\begin{aligned} Q &= \{0,1,2\} \times \{0,1,2,3,4,5,6\} \times \{0,1,2,3\} \\ s &= (0,0,0) \\ A &= \{(i,j,k) \in Q \mid i + j = k\} \end{aligned}$$

$$\begin{aligned} \delta((i,j,k), \mathbf{0}) &= (i+1 \bmod 3,\ j \qquad\qquad ,\ k+1 \bmod 4) \\ \delta((i,j,k), \mathbf{1}) &= (i \qquad\qquad ,\ j+1 \bmod 7,\ k+1 \bmod 4) \end{aligned}$$

Each state is represented by a triple of integers (i, j, k), where

- $i = \#(\mathbf{0}, w) \bmod 3$
- $j = \#(\mathbf{1}, w) \bmod 7$
- $k = |w| \bmod 4$

where w is the portion of the input string read so far. ■

Rubric: 2 points: Scaled DFA rubric (10 point DFA rubric at the end of the solutions).

Rubric: Standard DFA/NFA design rubric. 10 points =

- 2 points for an unambiguous description of a DFA or NFA, including the states set Q, the start state s, the accepting states A, and the transition function δ.
  - Drawings:
    - Use an arrow from nowhere to indicate the start state s.
    - Use doubled circles to indicate accepting states A.

    - If $A = \emptyset$, say so explicitly.
    - If your drawing omits a junk/trash/reject/hell state, say so explicitly.
    - Draw neatly! If we can't read your solution, we can't give you credit for it.
  - Text descriptions: You can describe the transition function either using a 2d array, using mathematical notation, or using an algorithm.
    - You must explicitly specify $\delta(q, a)$ for every state q and every symbol a.
    - If you are describing an NFA with ϵ-transitions, you must explicitly specify $\delta(q, \epsilon)$ for every state q.
    - If you are describing a DFA, then every value $\delta(q, a)$ must be a single state.
    - If you are describing an NFA, then every value $\delta(q, a)$ must be a set of states.
    - In addition, if you are describing an NFA with ϵ-transitions, then every value $\delta(q, \epsilon)$ must be a set of states.
  - Product constructions: You must give a complete description of each of the DFAs you are combining (as either drawings, text, or recursive products), together with the accepting states of the product DFA. In particular, we will not assume that product constructions compute intersections by default.
- Homework only: 4 points for briefly explaining the purpose of each state in English. This is how you argue that your DFA or NFA is correct.
  - In particular, each state must have a mnemonic name.
  - For product constructions, explaining the states in the factor DFAs is both necessary and sufficient.
  - Yes, we mean it. A perfectly correct drawing of a perfectly correct DFA with no state explanation is worth at most 6 points.
- 4 points for correctness. (8 points on exams, with all penalties doubled)
  - -1 for a single mistake: a single misdirected transition, a single missing or extra accepting state, rejecting exactly one string that should be accepted, or accepting exactly one string that should be rejected. (The incorrectly accepted/rejected string is almost always the empty string ϵ.)
  - -4 for incorrectly accepting every string, or incorrectly rejecting every string.
  - -2 for incorrectly accepting/rejecting more than one but a finite number of strings.
  - -4 for incorrectly accepting/rejecting an infinite number of strings.
- DFAs or NFAs that are more complex than necessary may be penalized. DFAs or NFAs that are significantly more complex than necessary may get no credit at all. On the other hand, minimal DFAs are not required for full credit, unless the problem explicitly asks for them.
- Half credit for describing an NFA when the problem asks for a DFA