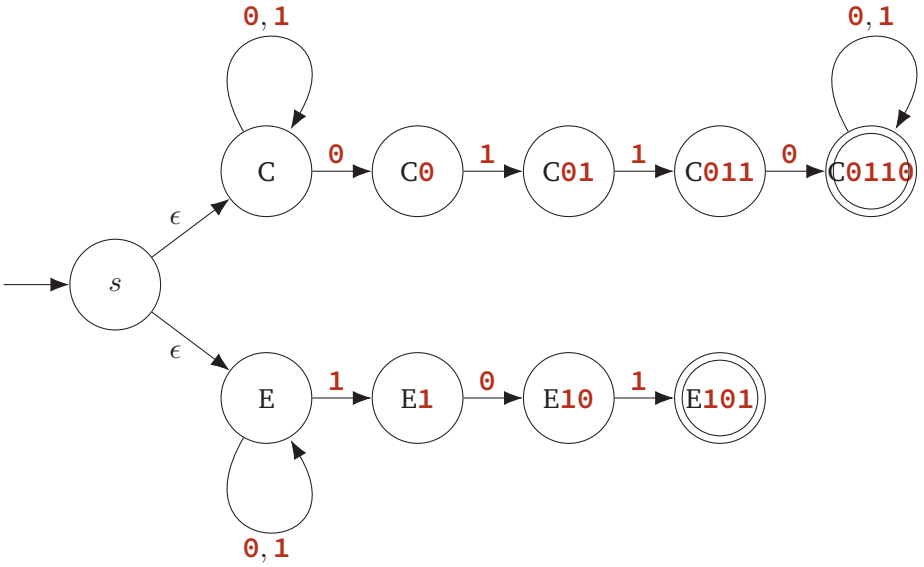
1. For each of the following parts, describe an NFA that accepts the given language **and briefly describe what each state in your automaton means**. Unless otherwise specified, we let $\Sigma = \{\mathtt{0}, \mathtt{1}\}$.

   (a) All strings that contain $\mathtt{0110}$ or end in $\mathtt{101}$.

   Solution: Our NFA is as follows:



   Meaning of the states:

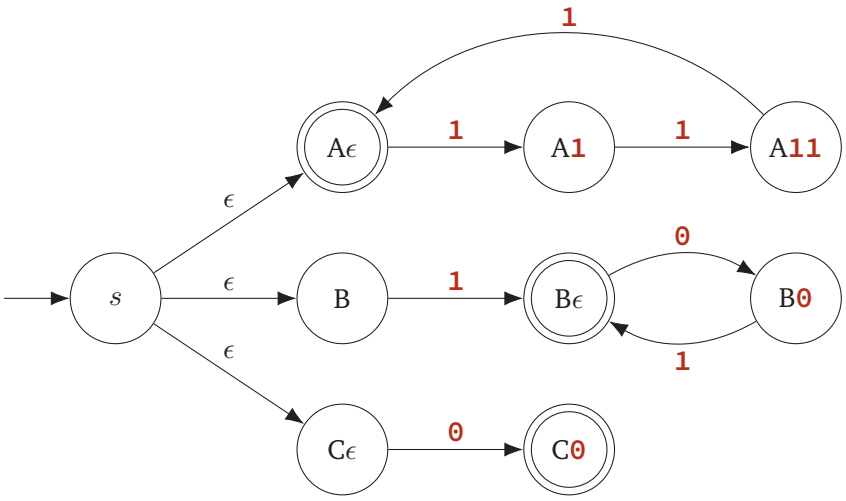
   - s: We have just started the NFA, and need to decide whether to check if the string contains $\mathtt{0110}$ or if it ends in $\mathtt{101}$.
   - C: We have decided to check if the string contains $\mathtt{0110}$, but have not yet started that check.
   - Cw: We have started checking if the string contains $\mathtt{0110}$, and have so far successfully seen the substring w.
   - E: We have decided to check if the string ends in $\mathtt{101}$, but have not yet started the check.
   - Ew: We have started checking if the string ends in $\mathtt{101}$, and have seen exactly the string w since starting the check.

   ■

   Rubric: 2.5 points: Scaled NFA rubric (10 point NFA rubric at the end of the solutions). Providing a DFA is also a valid solution.

   (b) The language defined by the regular expression $(\mathtt{111})^* + \mathtt{1}(\mathtt{01})^* + \mathtt{0}$.

   Solution: Our NFA is as follows:



Meaning of the states:

- s: We have just started the NFA, and need to decide whether to check the string against $\mathtt{111}^*$, $\mathtt{1}(\mathtt{01})^*$, or $\mathtt{0}$.
- Aw: We have decided to check the string against $(\mathtt{111})^*$, and have so far seen a string of the form $(\mathtt{111})^*w$.
- B: We have decided to check the string against $\mathtt{1}(\mathtt{01})^*$, and have not yet read any characters
- Bw: We have decided to check the string against $\mathtt{1}(\mathtt{01})^*$, and have so far seen a string of the form $\mathtt{1}(\mathtt{01})^*w$.
- Cw: We have decided to check the string against $\mathtt{0}$, and have so far read w.

■

Rubric: 2.5 points: Scaled NFA rubric (10 point NFA rubric at the end of the solutions). Providing a DFA or the NFA from Thompson's algorithm are also valid solutions.

(c) The language consisting of all strings whose 374-to-last character is the same as its last character. An equivalent definition of this language is $\{xbyb | x \in \Sigma^*, b \in \Sigma, y \in \Sigma^{372}\}$.

Solution: We define an NFA $N = (Q, \delta, s, A)$ as follows:

- $Q = \{s\} \cup \big(\{\mathtt{0}, \mathtt{1}\} \times \{1, 2, \ldots, 374\}\big)$
- $\delta(q, a) = \begin{cases} \emptyset & \text{if } a = \epsilon \\ \{s, (a, 1)\} & \text{if } q = s \text{ and } a \in \Sigma \\ \{(b, i+1)\} & \text{if } q = (b, i) \text{ for } i < 373 \text{ and } a \in \Sigma \\ \{(b, 374)\} & \text{if } q = (b, 373) \text{ and } a = b \\ \emptyset & \text{if } q = (b, 373) \text{ and } a \neq b \\ \emptyset & \text{if } q = (b, 374) \end{cases}$
- $s = s$
- $A = \{(\mathtt{0}, 374), (\mathtt{1}, 374)\}$

Intuitively, our NFA is guessing when it's seen the 374-to-last character, and storing that character in the first half of the state. It then waits for the next 372 characters, and transitions to an accepting state if the next character after those matches with

the one it stored. (This accepting state has no outgoing transitions, since if we read another character after getting there it means our guess at which character was 374-to-last was wrong.)

Meaning of the states:

- s: We have not yet guessed that we saw the 374-to-last character.
- $(\texttt{0}, i)$: We guessed that the character we read i positions ago was the 374-to-last, and that character was a `0`.
- $(\texttt{1}, i)$: We guessed that the character we read i positions ago was the 374-to-last, and that character was a `1`.

■

Rubric: 2.5 points: Scaled NFA rubric (10 point NFA rubric at the end of the solutions). Providing a DFA is also a valid solution.

(d) All strings over $\Sigma = \{\texttt{a}, \texttt{b}, \ldots, \texttt{z}\} \cup \{\texttt{0}, \texttt{1} \ldots \texttt{9}\}$ where every instance of the *substring* `cs173` occurs before any instance of the *substring* `cs374`. Examples of strings in this language include:

- `cs173isaprerequisiteforcs374`
- `hellocs374students`
- `examplesofstringsinthislanguage`

Examples of strings **not** in this language include:

- `cs173cs374cs173`
- `cs374andcs173arecool`

When building this NFA, you are allowed to assume you already have two **D**FAs: $M_{173} = (Q_{173}, \Sigma, \delta_{173}, s_{173}, A_{173})$ accepts all strings containing the substring `cs173`, and $M_{374} = (Q_{374}, \Sigma, \delta_{374}, s_{374}, A_{374})$ accepts all strings containing the substring `cs374`. You do not have to show how to construct these two DFAs, nor are you required to use them.

Solution: We first note that an equivalent way of understanding this language is the set of all strings w that can be written as xy where x does *not* contain the substring `cs374` and y does *not* contain the substring `cs173`. To see why this is, we make the following two observations:

- Any string $w \in L$ can be written in this way by taking x to be the prefix of w up to and including the last instance of the substring `cs173`,[1] and y to be the remainder of the string. Since $w \in L$, every instance of `cs374` has to happen after all the instances of `cs173`, and so cannot occur in x. Additionally, x contains all the instances of `cs173`, so none can be in y.
- No string $w \notin L$ can be written in this way. In order to have $w \notin L$, w must contain an instance of `cs374` and an instance of `cs173` that occurs after it. In order to write $w = xy$ where x does not contain `cs374`, the split between x and y has to occur before the end of the `cs374` substring—but if this happens, y must contain the `cs173` substring!

[1] If w does not contain `cs173`, this will set x to the empty string.

With this alternative characterization of L, we can now give an NFA for it as follows:

- $Q = Q_{173} \cup Q_{374}$
- $\delta(q,a) = \begin{cases} \{\delta_{173}(q,a)\} & \text{if } q \in Q_{173}, a \in \Sigma \\ \{\delta_{374}(q,a)\} & \text{if } q \in Q_{374}, a \in \Sigma \\ \{s_{173}\} & \text{if } q \in Q_{374} - A_{374}, a = \epsilon \\ \emptyset & \text{if } q \in A_{374}, a = \epsilon \\ \emptyset & \text{if } q \in Q_{173}, a = \epsilon \end{cases}$
- $s = s_{374}$
- $A = Q_{173} - A_{173}$

States from Q_{374} represent that our NFA has guessed that it's still reading the x portion of w, while states from Q_{173} represent that our NFA has guessed it's moved on to reading the y portion. The ϵ transitions allow us to move from the former to the latter as long as we *haven't* yet seen the substring **cs374**, while the accepting states guarantee that we only accept if we *don't* see **cs173** in the y portion. ■

Rubric: 2.5 points: Scaled NFA rubric (10 point NFA rubric at the end of the solutions). The formal proof of the xy characterization of L is not needed for full credit. Providing an explicit NFA or DFA is also a valid solution.

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 $A = \varnothing$, 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 $\delta(q,\varepsilon)$ 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 $\delta(q,\varepsilon)$ 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 accepted. (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.

2. Fix the alphabet $\Sigma = \{\texttt{0}, \texttt{1}\}$. We define the function shift that shifts every bit in its input string one place to the right and moves the final bit to the front. Formally, if $w = w_1 w_2 \dots w_n$ is a non-empty string, $\text{shift}(w) = w_n w_1 \dots w_{n-1}$, while $\text{shift}(\epsilon) = \epsilon$.

   (a) Let L be an arbitrary regular language. Prove that $L_{\text{shifted}} = \{\text{shift}(w) | w \in L\}$ is also regular.

   Solution: Let $M = (Q, \delta, s, A)$ be a DFA for L. We construct a DFA M' for L_{shifted} intuitively by remembering the first character we read (which is the last character of the string we need to check for L), then running M on the remaining string. When we reach the end, we accept iff M reaches an accepting state after also reading our stored character. Formally, we have:

   - $Q' = \{s'\} \cup (Q \times \Sigma)$
   - $\delta'(q', a) = \begin{cases} (s, a) & \text{if } q' = s' \\ (\delta(q, a), b) & \text{if } q' = (q, b) \end{cases}$
   - $s' = s'$
   - $A' = \{(q, a) | \delta(q, a) \in A\}$
     - We also include s' in A' if and only if $\epsilon \in L$.

   Meaning of the states:

   - s': We have not yet read any characters.
   - (q, b): The first character we read was b. Running M on all characters after that, we got to state q.

   ■

Rubric: 2.5 points: Scaled language transformation rubric (10 point language transformation rubrics for both automata and regular expressions at the end of the solutions).

(b) Let L be an arbitrary regular language. Prove that $L_{\text{unshifted}} = \{w | \text{shift}(w) \in L\}$ is also regular.

Solution: Let $M = (Q, \delta, s, A)$ be a DFA for L. We construct an NFA M' for $L_{\text{unshifted}}$ intuitively by guessing what the last character of our input string will be and running M with that character shifted to the front. We need to make sure to not give the last character we read to M (since that was already shifted to the front), so we only give a character to M after we've read the next one, storing it in our state until then. When we reach the end of the string, we accept if our initial guess for the last character was correct and M reached an accepting state. (To deal with the edge case where our input string is empty, we let our start state be an accepting state iff $\epsilon \in L$.) Formally, we have

- $Q' = \{s'\} \cup (Q \times \Sigma \times \Sigma)$
- $\delta'(q', a) = \begin{cases} \{(\delta(s, \mathtt{0}), \mathtt{0}, a), (\delta(s, \mathtt{1}), \mathtt{1}, a)\} & \text{if } q' = s', a \in \Sigma \\ \{(\delta(q, c), b, a)\} & \text{if } q' = (q, b, c), a \in \Sigma \\ \emptyset & \text{if } a = \epsilon \end{cases}$
- $s' = s'$
- $A' = \{(q, b, c) | q \in A, b = c\}$
  - We also include s' in A' if and only if $\epsilon \in L$.

Meaning of the states:

- s': We have not yet read any characters.
- (q, b, c): We guessed that the last character of w is going to be b, and the most recent character we've read is c. Running M on the string we've read so far but with b at the front and c removed from the end lands us in state q.

■

Rubric: 2.5 points: Scaled language transformation rubric (10 point language transformation rubrics for both automata and regular expressions at the end of the solutions).

We also define the function deleteOnes that deletes all the $\mathtt{1}$s in a string. Thus for example, $\text{deleteOnes}(\mathtt{0101}) = \mathtt{00}$, $\text{deleteOnes}(\mathtt{111}) = \epsilon$, and $\text{deleteOnes}(\mathtt{000}) = \mathtt{000}$.

(c) Let L be an arbitrary regular language. Prove that $L_{\text{deleted}} = \{\text{deleteOnes}(w) | w \in L\}$ is also regular.

Solution (NFA): Let $M = (Q, \delta, s, A)$ be a DFA for L. We construct an NFA M' for L_{deleted} intuitively by running M on our input string, but using ϵ-transitions to "guess" when to insert $\mathtt{1}$s. If we ever see a $\mathtt{1}$ in our input string, we can immediately reject, since all strings in L_{deleted} consist only of $\mathtt{0}$s. Formally, we have

- $Q' = Q$

- $\delta'(q,a) = \begin{cases} \{\delta(q,\texttt{0})\} & \text{if } a = \texttt{0} \\ \emptyset & \text{if } a = \texttt{1} \\ \{\delta(q,\texttt{1})\} & \text{if } a = \epsilon \end{cases}$
- $s' = s$
- $A' = A$

Meaning of the states:

- q: The input string so far has consisted only of **0**s. If we inserted **1**s into our string so far in the places we've previously guessed, M would end up in state q.

■

Solution (Regular Expression): Let r be a regular expression for L. We can create a regular expression r' for L_{deleted} by simply replacing every **1** in r with ϵ. This modification works because it means that any time r says our string should have a **1**, r' says that there should instead be no character there. ■

Rubric: 2.5 points: Scaled language transformation rubric (10 point language transformation rubrics for both automata and regular expressions at the end of the solutions).

(d) Let L be an arbitrary regular language. Prove that $L_{\text{undeleted}} = \{w \mid \text{deleteOnes}(w) \in L\}$ is also regular.

Solution: Let $M = (Q, \delta, s, A)$ be a DFA for L. We construct an DFA M' for $L_{\text{undeleted}}$ intuitively by running M but ignoring (ie, not transitioning on) every **1** we see. Formally, we have

- $Q' = Q$
- $\delta'(q,a) = \begin{cases} q & \text{if } a = \texttt{1} \\ \delta(q,a) & \text{if } a \neq \texttt{1} \end{cases}$
- $s' = s$
- $A' = A$

Meaning of the states:

- q: If we ran M on the input string so far with all **1**s removed, we would end up in state q.

■

Rubric: 2.5 points: Scaled language transformation rubric (10 point language transformation rubrics for both automata and regular expressions at the end of the solutions).

For each of these parts, you should provide a DFA, NFA, or regular expression for the modified language. If you provide an automaton, **describe what each state means** and give a short justification for why it works. If you provide a regular expression, briefly justify why it is correct.

Standard language transformation rubric (automata). For problems worth 10 points:

+ 2 for a formal, complete, and unambiguous description of the output automaton M', including the states, the start state(s), the accepting states, and the transition function, as functions of an *arbitrary* given DFA M. The description must state whether the output automaton is a DFA or an NFA.
  - No points for the rest of the problem if this is missing.
+ 2 for a *brief* English explanation of the output automaton. We explicitly do *not* want a formal proof of correctness, or an English *transcription*, but a few sentences explaining how your machine works and justifying its correctness. What is the overall idea? What do the states represent? What is the transition function doing? Why these accepting states?
  - **Deadly Sin:** No points for the rest of the problem if this is missing.
+ 6 for correctness
  + 1 for correct states — Almost always a product of the states Q of the given DFA with other side information; does the side information make sense? Could you build a transformation using *only* this side information?
  + 1 for correct start state(s)
  + 1 for correct accepting states
  + 3 for correct transition function
    - 1 for a single minor mistake
  - Double-check correctness when the input language is $\emptyset$, or $\{\epsilon\}$, or $\mathbf{1}^*$, or Σ^*.
  - Partial credit should be awarded relative to the *most similar correct solution*. For example, if a given incorrect solution can be fixed either by changing the accepting states or by changing the transition function, it should get partial credit for a good transition function.

Standard language transformation rubric (regular expressions). For problems worth 10 points:

+ 2 for a formal, complete, and unambiguous description of the output regular expression r', as a function of an *arbitrary* given regular expression r. This description can be recursive, in which case it needs to specify what to do in each base case and each inductive case.
  - No points for the rest of the problem if this is missing.
+ 2 for a *brief* English explanation of the output expression. We explicitly do *not* want a formal proof of correctness, or an English *transcription*, but a few sentences explaining how your expression works and justifying its correctness. What is the overall idea? Why these particular modifications? If your construction is recursive, you should briefly justify each base case and each recursive case.
  - **Deadly Sin:** No points for the rest of the problem if this is missing.
+ 6 for correctness
  - For recursive constructions: 1 point per case ($\emptyset$, ϵ, character, $r_1 + r_2$, $r_1 r_2$, $(r_1)^*$). Half credit on a case if it contains a single minor mistake.
  - For direct constructions:
    + 3 points for accepting all strings in the target language.

- No credit if this is only because the expression always accepts every string.
- Half credit if this is true up to a minor mistake.

+ 3 points for accepting *only* strings in the target language.

- No credit if this is only because the expression always rejects every string.
- Half credit if this is true up to a minor mistake.

- Double-check correctness when the input language is $\emptyset$, or $\{\epsilon\}$, or $\mathbf{1}^*$, or Σ^*.
- Partial credit should be awarded relative to the *most similar correct solution*.