Homework 1

COMS W3261, Summer A 2023

This homework is due **Monday, 5/29/2023, at 11:59pm EST**. Submit to GradeScope (course code: K3VK75). If you use late days, the absolute latest we can accept a submission is the following Friday at 11:59 PM EST.

Grading policy reminder: LaTeX is preferred, but neatly typed or handwritten solutions are acceptable.[1] Feel free to use the .tex file for the homework as a template to write up your answers, or use the template posted on the course website. Your TAs may dock points for indecipherable writing.

Proofs should be complete; that is, include enough information that a reader can clearly tell that the argument is rigorous.

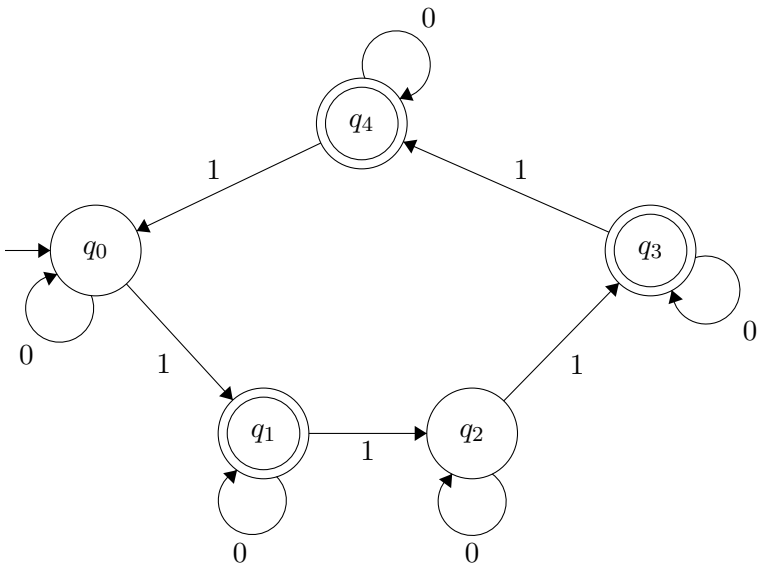
If a question is ambiguous, please state your assumptions. This way, we can give you credit for correct work. (Even better, post on Ed so that we can resolve the ambiguity.)

LaTeX resources.

- Detexify is a nice tool that lets you draw a symbol and returns the LaTeX codes for similar symbols.
- The tool Table Generator makes building tables in LaTeX much easier.
- The tool Finite State Machine Designer may be useful for drawing automata. See also this example (PDF) (.tex) of how to make fancy edges (courtesy of Eumin Hong).
- The website mathcha.io allows you to draw diagrams and convert them to LaTeX code.
- To use the previous drawing tools (and for most drawing in LaTeX), you'll need to use the package Tikz (add the command "\usepackage{tikz}" to the preamble of your .tex file to import the package).
- This tutorial is a helpful guide to positioning figures.

[1] The website Overleaf (essentially Google Docs for LaTeX) may make compiling and organizing your .tex files easier. Here's a quick tutorial.

Problem 1



1. (3 points.) The DFA state diagram above is defined on the alphabet $\Sigma = \{0, 1\}$. Write out its formal definition (as a 5-tuple). When specifying the transition function δ, you can draw a table or simply describe the output of δ for all states on all possible inputs.

   We can write the DFA as a 5-tuple $\{Q, \Sigma, \delta, q_0, F\}$, in which the state set $Q = \{q_0, q_1, q_2, q_3, q_4\}$, $\Sigma = \{0, 1\}$, and $F = \{q_1, q_3, q_4\}$. (q_0 is already labeled, so it needs no further definition.), finally, the transition function δ has $\delta(q_i, 0) = q_i$ and $\delta(q_i, 1) = q_{(i+1) \pmod 5}$ for $i \in \{0, 1, 2, 3, 4\}$. (Other ways of writing this out are fine.)

2. (3 points.) Describe the language recognized by the DFA in one or two sentences, then explain why the DFA accepts a string if and only if it matches your description. [Hint: Test some strings and look for a pattern; never forget the empty string ϵ!]

   This DFA accepts all strings in which the number of 1's is 1, 3, or 4 mod 5.

   We can see this by observing that the states of the DFA form a cycle of length 5, and that the state never changes on an 0, so the state 'advances' one step if and only if we read in a 1. The state counts the number of 1's (mod 5), and the states corresponding to 1, 3, and 4 mod 5 are accept states.

Rationale: The goal of this question is to make sure you're comfortable reading state diagrams, evaluating DFAs on strings, and reasoning about what they do.

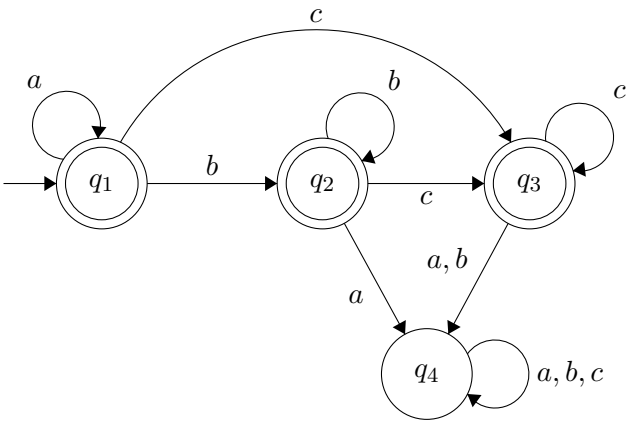
References: Sipser 1.1 pp. 34-37; Lightning review of DFAs from the Resource Archive section of the course webpage.

Problem 2.

1. (4 points.) Consider the DFA drawn below. What is the language recognized by this DFA (over what alphabet)? Describe your reasoning.

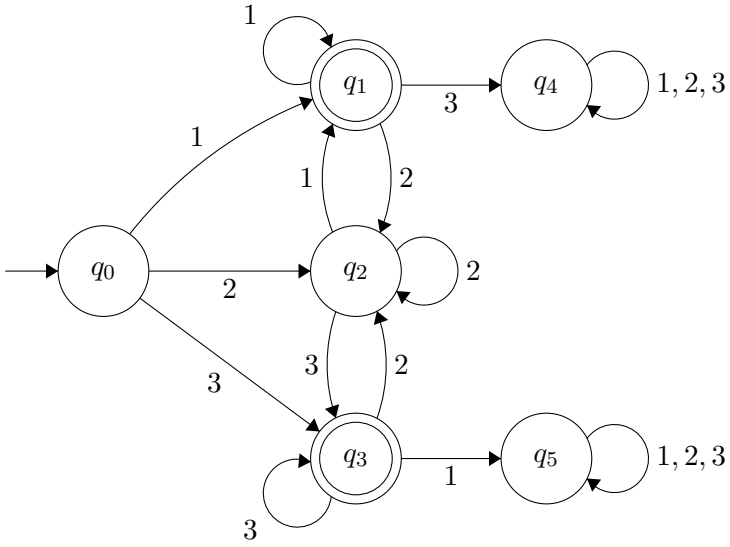
   The DFA below accepts strings on the alphabet $\{a, b, c\}$ in which b's never precede a's and c's never precede a's or b's.

   We can see this by observing that, as long as we stay in the set $\{q_1, q_2, q_3\}$, we are guaranteed to accept. These states correspond to having read only a's, having read some number of a's, followed by at least one b, and having read some number of a's and b's, followed by at least one c, respectively. We reject only if we take a transition to q_4, which occurs if we see an a after a b or an a or a b after a c.



2. (4 points.) The DFA drawn below recognizes all strings over the alphabet $\{1, 2, 3\}$ that end in a 1 or a 3 and don't contain 1's and 3's adjacent to each other (that is, the substrings 13 and 31 are forbidden). Draw a DFA state diagram that recognizes the same language (that is, accepts exactly the same set of strings over $\{1, 2, 3\}$) but which has **at most four states** in total.

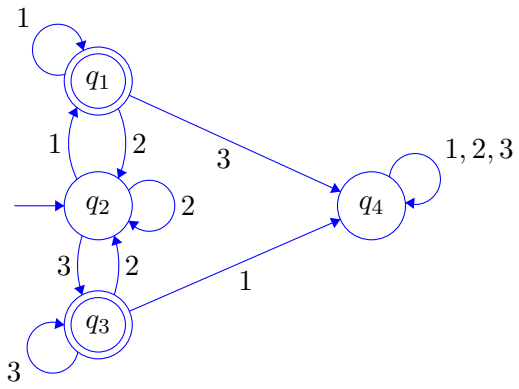
   Explain in words why your DFA recognizes the same language as the pictured DFA.



Our new DFA (pictured below) removes q_0, making q_2 the new start state, and 'merges' the states q_4 and q_5.

To see why the function of this new DFA is identical, first observe that in the original DFA, q_4 and q_5 are both inescapable reject states. Any computation that reaches either state is guaranteed to reject, and merging them does not change this.

Second, compare the behavior of the old and new DFAs at the beginning of computation. On ϵ, both DFAs reject. On an initial 1, both DFAs proceed to q_1; on a 2, to q_2, and on a 3, to q_3. Since it is not possible to return to q_0, the execution thereafter must be identical.



Rationale: More practice reading state diagrams and evaluating DFAs on strings; also, practice designing DFAs to perform a task and simplifying them.

References: Sipser 1.1 pp. 34-37 (DFAs); Sipser 1.1 pp. 41-43 (designing DFAs); Lightning reviews of DFAs and designing DFAs from the Resource Archive section of the course webpage.

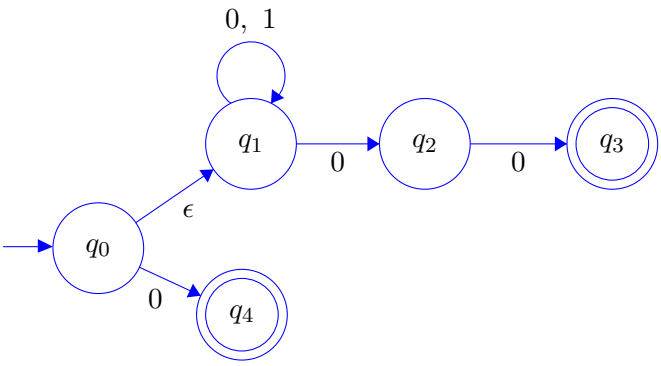
Problem 3

1. (4 points). In binary, the natural numbers are $0, 1, 10, 11, 100, 101, ...$, etc. Draw a state diagram for an NFA on the alphabet $\Sigma = \{0, 1\}$ that recognizes the language of binary natural numbers divisible by 4. (For instance, your machine should accept 100, but reject 11 and 101. Reject strings with extra leading zeroes, like 0100.)

   Explain in words why your NFA recognizes the language specified.

   Our NFA state diagram is pictured below. The key observation here is that all binary numbers divisible by 4 either (1) begin with a 1, and end with an 00, or (2) are 0. Our NFA handles these two cases.

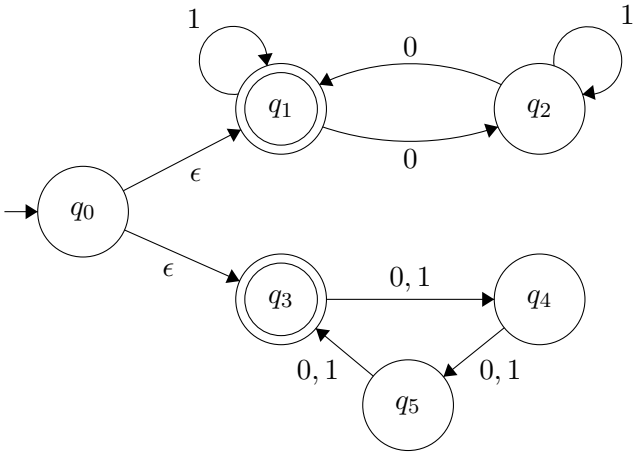
   First, we observe that our NFA accepts the string 0 and rejects all other strings beginning with 0, as the branch of computation dies (these have extra leading 0's). On the other hand, if our string begins with a 1, then a single branch of computation proceeds to q_1. From there, one branch remains in the state q_1 as we read in characters, spinning off another branch that proceeds to q_2 whenever it encounters a 0. These side branches cause the computation to accept if and only if the NFA reads in exactly two 0's before halting. Thus we accept all strings beginning with 1 and ending with 00.



2. (2 points). Over the binary alphabet $\{0, 1\}$, Let L_1 be the language of all strings whose length is divisible by 3, and L_2 be the language of all strings that have an even number of 0's. The state diagram below depicts an NFA that recognizes $L_1 \cup L_2$.

   Specify the elements of a 5-tuple $(Q, \Sigma, \delta, q_0, F)$ equivalent to the NFA state diagram below. When specifying the transition function, feel free to specify δ only for inputs that transition to a non-empty set of states; for the others, you can assume δ maps to $\emptyset$. No justification required.

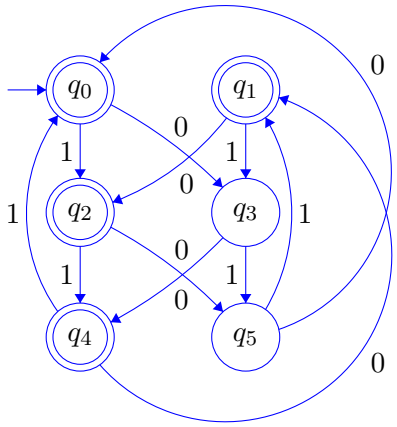
   Here $Q = \{q_0, q_1, q_2, q_3, q_4, q_5\}$, $\Sigma = \{0, 1\}$, and $F = \{q_1, q_3\}$. We can summarize the transitions specified by δ as follows (for all other inputs, δ maps to $\emptyset$): $\delta(q_0, \epsilon) = \{q_1, q_3\}$, $\delta(q_1, 0) = \{q_2\}$, $\delta(q_1, 1) = \{q_1\}$, $\delta(q_2, 0) = \{q_1\}$, $\delta(q_2, 1) = \{q_2\}$. $\delta(q_3, a) = q_4$, $\delta(q_4, a) = q_5$, and $\delta(q_5, a) = q_3$ for $a \in \{0, 1\}$.



3. (4 points). Draw the state diagram of a *DFA* over $\{0, 1\}$, **with at most 6 states**, that also recognizes $L_1 \cup L_2$. Explain in words why your DFA recognizes this language. (We'll discuss a general procedure for converting NFAs to DFAs in Lecture 3, but this question doesn't require it.)

   The original NFA accepts if the input has an even number of 0's OR if the length of the input is divisible by 3. Because we don't have access to nondeterminism when designing a DFA, we need to design a DFA that keeps track of both pieces of information at the same time.

   In particular, we'll design a DFA with 6 states, divided into two columns (corresponding to "even number of 0's" and "odd number of 0's") and three rows (corresponding to the residue classes mod 3). We'll design our transition function so that we move down a row every time we read in a character and swap columns every time we read in an 0; this will keep our DFA consistent. Finally, we accept in any state corresponding to an even number of 0's OR an input length divisible by 3.



Rationale: The purpose of this question is to get experience in building, testing, and simplifying NFAs, as well as converting NFAs to DFAs.

References: Sipser 1.2 pp. 47-53 (NFAs), also see the Lightning Review video on NFAs in the Resource Archive section of the course webpage.

Problem 4

1. (2 points.) Recall that the (Kleene) star operation is defined as follows:

$$A^* = \{x_1 x_2 \dots x_k \mid x_1, x_2, \dots, x_k \in A, k \geq 0\}.$$

Let A and B be regular languages. Is it true that $(A \cup B)^*$ and $A^* \cup B^*$ are the same language? If so, provide a proof. If not, provide a counterexample. (Recall the order of operations: $A^* \cup B^* = (A^*) \cup (B^*)$.)

No, these are not the same language. For a counterexample, let $A = \{000\}$ and $B = \{111\}$. The first expression evaluates to $\{000, 111\}^*$, which includes the string 111000, but the second language, which includes only strings in $\{000\}^*$ or $\{111\}^*$, does not.

2. (2 points.) Let A be a regular language. Is it true that A^*A^* and $(AA)^*$ are the same language? If so, provide a proof. If not, provide a counterexample.

No, these are not the same language. For a counterexample, let $A = \{000\}$. A^*A^* contains the string $000 = 000\epsilon$, as both 000 and ϵ are elements of A^*. However, $(AA)^* = \{000000\}^*$, which does not contain the string 000.

Some practice reasoning about the regular operations.
References: Sipser p.44 (definition of Union, Concatenation, and Star).

Problem 5

1. (6 points.) Given a language A over an alphabet Σ, define $doub(A)$ as follows:

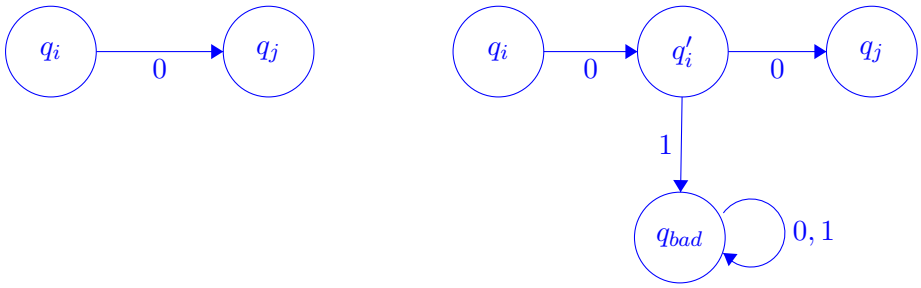
$$doub(A) := \{w_1 w_1 w_2 w_2 \dots w_n w_n \mid w_1 w_2 \dots w_n \in A; w_1, w_2, \dots, w_n \in \Sigma\}.$$

That is, each string in $doub(A)$ can be obtained by beginning with some string $w_1 w_2 \dots w_n$ in A and replacing each character with two copies of itself.

Show that the class of regular languages is closed under the operation *doub* by explaining how to modify an arbitrary DFA D_1 so that it recognizes the language $doub(L(D_1))$. For simplicity, you may consider just the case in which the alphabet is $\Sigma = \{0, 1\}$.

Let $D_1 = (Q, \Sigma, \delta, q_0, F)$ be a DFA. To create a DFA D_2 that recognizes $doub(L(D_1))$, we'll split each transition with a new state: if $\delta(q_i, 0) = q_j$, we'll create a new state q_i' and replace the old transition with $\delta(q_i, 0) = q_i'$. We'll also add the transition $\delta(q_i', 0) = q_j$.

Now we transition from q_i to q_j on an '00', instead of on an '0'. Because we want to accept only even-length strings, q_i' will not be an accept state. Finally, because we want to reject if we read in the substring '01' from q_i, we'll add the transition $\delta(q_i', q_{bad})$ for a "global" reject state q_{bad}. The picture below depicts our splitting operation before and after:



We perform this splitting operation on every 0-transition, and the reverse operation on every 1-transition (that is, we transition from q_i to q_i' to q_j on 1's, and transition from q_i' to q_{bad} on a 0).

It should be clear from the definition of the split that our new DFA accepts every string in $doub(L(D_1))$. To see that the new construction rejects strings that are *not* in the language, consider three possible cases:

- The string has odd length. In this case, our computation will always end on a newly added state (some q_i'), because we always alternate between old states and newly added states. Newly added states are reject states, so we reject.
- The string has even length, but when divided into substrings of two characters, contains an '01' or a '10'. In this case, our splitting operation ensures we end up in q_{bad}.
- The string has even length and divides evenly into the substrings '00' and '11', but does not correspond to a string in $L(D_1)$ with characters doubled. Call the new string w, and the string obtained by replacing each '00' with an '0' and each '11' with a '1' w'. In this

case, our new DFA, given w, effectively simulates D_1 on w', and ends on the same reject state.

2. (2 points.) Given a language A over an alphabet Σ, define $oct(A)$ similarly to $doub(A)$, except that we now replace each character with *eight* copies of itself.

   Show that the class of regular languages is closed under the operation *oct* by reasoning from the fact that the regular languages are closed under other, previously proved, regular expressions. (That is, prove *without* explaining how to modify a DFA.) You may assume that *doub* is regular.

   Let A be a regular language. The proof follows from observing that $oct(A) = doub(doub(doub(A)))$. Given that *doub* is a regular operation, we have that $doub(A)$, and thus $doub(doub(A))$, and finally $doub(doub(doub(A))) = oct(A)$ are all also regular languages.

Problem 6 (1 point)

1. Do you have any leftover questions or confusions from week 1 that you'd like to see addressed in class?

2. (Optional) Any other thoughts? Thank you!

   Any answer gets credit.

Rationale: We'll spend more time reviewing the toughest topics from each week, and we'll repeat the activities that are most useful. Also, feedback helps make the course better.