Homework 1

COMS W3261, Summer A 2022

This homework is due **Tuesday, 5/31/2022, at 11:59pm EST**. Submit to GradeScope (course code: 2KGDW8).

Grading policy reminder: LaTeX is preferred, but neatly typed or handwritten solutions are acceptable.[1] I recommend using the .tex file for the homework as a template to write up your answers. 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.

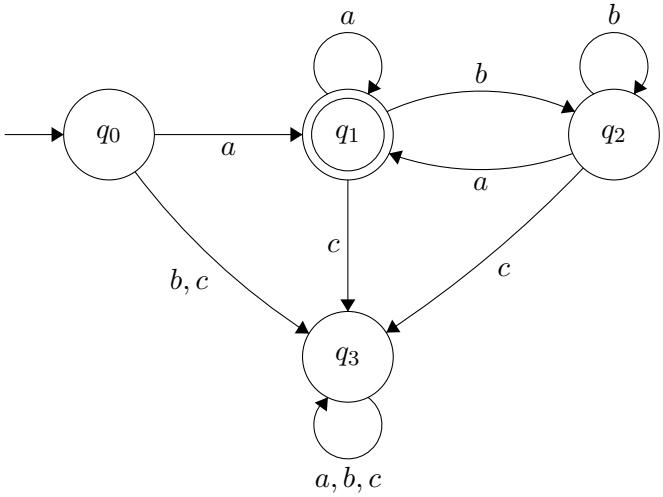
If a question is ambiguous, please state your assumptions. This way, we can give you credit for correct work. (Even better, post on Ed so that we can resolve the ambiguity.)

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.

1 Problem 1 (6 points)



1. (3 points.) The DFA state diagram above is defined on the alphabet $\Sigma = \{a, b, c\}$. 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 on all possible inputs.

   The state diagram represents a DFA $D = (Q, \Sigma, \delta, q_0, F)$, with a set of states $Q = \{q_0, q_1, q_2, q_3\}$, the alphabet $\Sigma = \{a, b, c\}$, and set of accept states $F = \{q_1\}$.

   The transition function δ can be summarized as follows:

   (a) $\delta(q_0, a) = \delta(q_1, a) = \delta(q_2, a) = q_1$ (whenever we see a, go to q_1, unless we're in q_3).
   (b) $\delta(q_0, b) = q_3$. $\delta(q_1, b) = \delta(q_2, b) = q_2$.
   (c) $\delta(q, c) = q_3$ for all $q \in Q$ (whenever we see the character c, we go to state q_3).
   (d) $\delta(q_3, x) = q_3$ for all $x \in \Sigma$ (once we're in q_3, we stay in q_3.)

2. (3 points.) Describe the language recognized by the DFA in one sentence, 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!]

   D recognizes the language of strings over $\{a, b, c\}$ that start and end in a and do not contain a c. (Or, similarly: D recognizes strings over $\{a, b\}$ that start and end in a.)

   Both writing out the formal definition and testing strings may help us work out the function of this DFA. A first observation is that whenever we see the character c, we move to state q_3. q_3 is a reject state from which our computation cannot escape, so this DFA never accepts a string that contains a c.

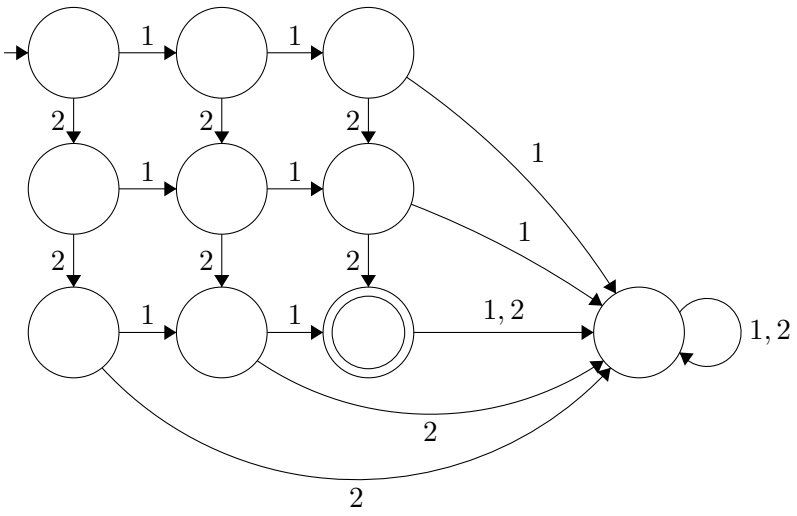
   We're left with strings made up of a's and b's. First, we observe that if we start with b, we go to q_3 and cannot accept, so we must start with a. After this, our computation bounces between q_1 and q_2. After reading an a, we always go to q_1 (an accept state), while we always go to q_2 (a reject state) after reading a b. Thus we'll accept if the last character of the string is an a.

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 linked in the Lecture 1 Course Skeleton.

2 Problem 2 (8 points).

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

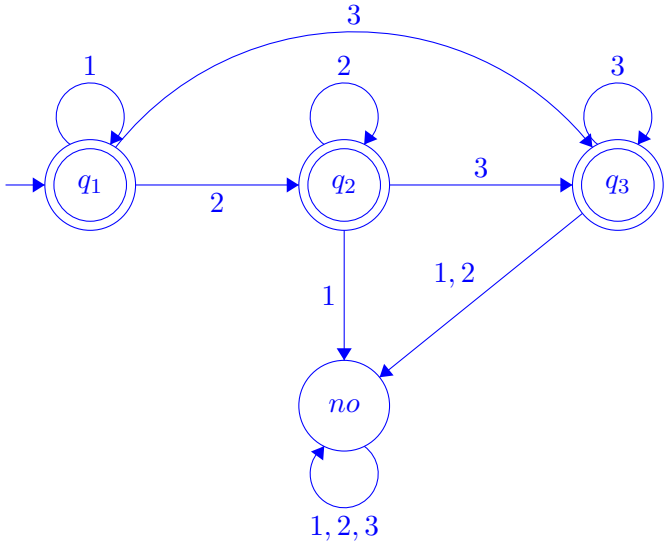


This DFA accepts all strings over the alphabet $\Sigma = \{1, 2\}$ that contain exactly two 1's and exactly two 2's, in any order.

To see this, observe that every path from the start state to the single accept state contains exactly two 1's and two 2's (every down transition is a 2, and every right transition is a 1.) Whenever we see a third 1 (transition on a 1 from the right-hand column) or a third 2 (transition on a 2 from the bottom row) we move to a reject state from which there is no escape.

2. (4 points.) Draw a state diagram for a DFA **with at most 4 states** that recognizes the language over the alphabet $\Sigma = \{1, 2, 3\}$ consisting of all strings of digits that never decrease (that is, we never see a 1 after we've seen a 2 or a 3, or a 2 after we've seen a 3.)

Explain in words why your DFA recognizes the language specified.



The key here is identifying three core states: we need a state that allows any next character (when we've seen only 1's, or nothing at all), a state that rejects on a 1 (when we've already seen a 2, but no 3's yet) and a state that rejects on 1's and 2's (when we've already seen a 3). In the diagram above, we've used q_1, q_2, and q_3 for these purposes. We transition between them based on the next character (and note that we can't 'go back'; once we've seen a 2 we're never allowed to see a 1 again). When we see a character that's not allowed, we go to the *no* state, from which we can't escape. Note that this DFA accepts the empty string ϵ, which contains no decreasing digits.

Because we originally defined the language as "strings of digits", a reasonable person might conclude that we don't include the empty string. An answer consistent with this interpretation is OK.

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

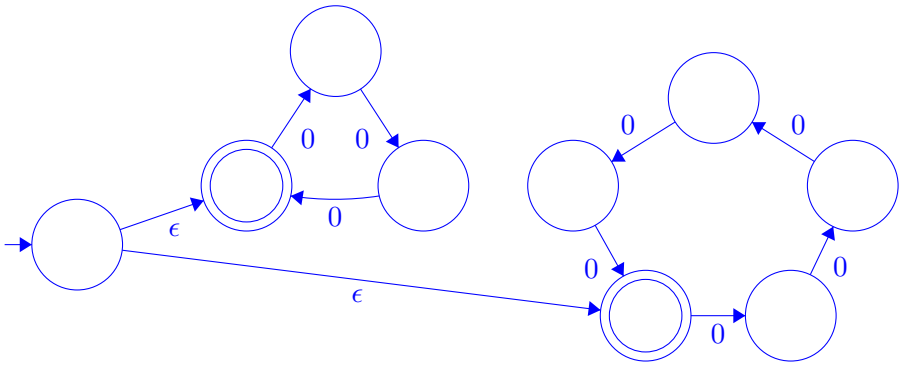
References: Sipser 1.1 pp. 34-37 (DFAs); Sipser 1.1 pp. 41-43 (designing DFAs); Lightning reviews of DFAs and designing DFAs linked in the Lecture 1 Course Skeleton.

3 Problem 3 (8 points)

1. (3 points). Draw a state diagram for an NFA on the alphabet $\Sigma = \{0\}$ that recognizes the language

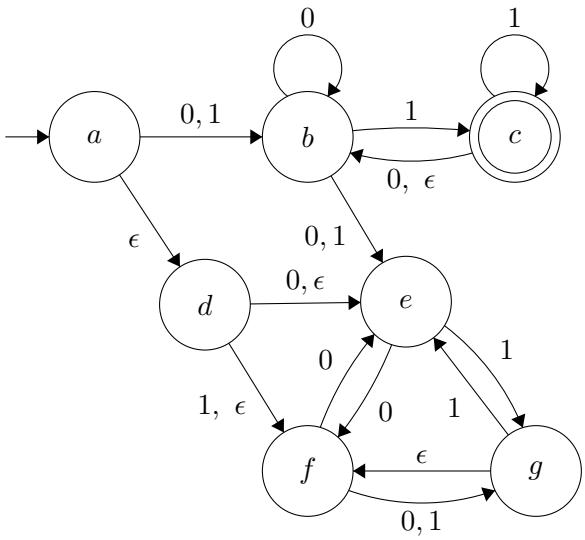
$$L = \{w \mid |w| \text{ is divisible by 3 or by 5 (or by both 3 and 5). }\}.$$

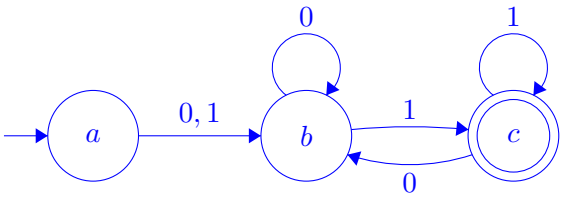
(Recall that $|w|$ denotes the length of the string w.) Explain in words why your NFA recognizes the language specified.



At the beginning of its execution, our NFA splits and takes an ϵ-transition to two smaller sub-NFAs: a divisible-by-3 tester and a divisible-by-5 tester, similar to machines we've seen in class. Because every state (except the start state) has an out-transition on 0, neither branch of computation dies until we reach the end of our string. Our DFA will then accept (via the top branch) if we've read in a number of characters divisible by 3, and likewise if we've read in a number of characters divisible by 5 on the bottom branch.

2. (5 points). Draw the state diagram of a DFA (with alphabet $\Sigma = \{0, 1\}$) **with at most 3 states** that recognizes the same language as the NFA whose state diagram is pictured below. Explain in words why your DFA captures the same language as the original NFA. [Hint: A DFA with three states can only do so much. So, if this question is possible, the NFA below must be simpler than it seems. Try to work out what it does (testing strings is never a bad option.)]





The first key here is to observe that the lower four states (d, e, f, and g) are pointless: there are no transitions from these states back to a, b, or c, so once a branch of computation reaches one of these states, it can never find an accept state.

We can then consider just a, b, and c. This sub-NFA is almost a DFA, but it has an additional ϵ-transition from c to b. We can observe the following: (1) once we reach b or c, we always stay in one of those two states. Both have out-transitions on 0 and 1 so the branch of computation will never die. (2) once we reach b or c, we are in the accept state (c) if and only if we have just read in a 1. This remains true with or without the ϵ-transition, so we can remove it. (Checking strings can help to verify this.)

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

References: Sipser 1.2 pp. 47-53 (NFAs), also see the Lightning Review video on NFAs linked in the Course Skeleton.

4 Problem 4 (8 points)

1. (5 points.) Given languages A and B, define the $XNOR$ operation $\odot$ as follows:

$$A \odot B := \{x \mid x \in (A \cap B), \text{ or } x \in (\overline{A} \cap \overline{B})\}.$$

Prove that the class of regular languages is closed under $\odot$ by showing that, given two arbitrary DFAs D_1 and D_2, we can build a new DFA D_3 that recognizes the language $L(D_1) \odot L(D_2)$. [Hint: recall our proof that the regular languages are closed under the union operation, available on p.45 of Sipser.]

To do this requires tweaking our proof that the regular languages are closed under $\cup$. (See the proof of Theorem 1.25 on Sipser pp. 45-46.)

Let $M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ and $M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ be DFAs that recognize the languages A_1 and A_2, respectively. We'll construct a new machine $M = (Q, \Sigma, \delta, q_0, F)$ that recognizes $A_1 \odot A_2$.

Let $Q = \{(r_1, r_2) \mid r_1 \in Q_1, r_2 \in Q_2\}$, i.e., the set of all pairs of states in Q_1 and Q_2, otherwise known as the Cartesian product $Q_1 \times Q_2$. We let Σ be unchanged, set $q_0 = (q_1, q_2)$, and define our transition function so that it simulates both transition functions δ_1 and δ_2: for each $(r_1, r_2) \in Q$ and $a \in \Sigma$, let $\delta((r_1, r_2), a) = (\delta_1(r_1, a), \delta_2(r_2, a))$.

The difference from our previous construction is in the set of accept states: we define

$$F = \{(r_1, r_2) \mid (r_1 \in F_1, r_2 \in F_2) \text{ OR } (r_1 \notin F_1, r_2 \notin F_2)\}.$$

As a result, M accepts a string w when either (1) both M_1 and M_2 accept w or (2) neither M_1 nor M_2 accepts w: in other words, when $w \in A_1 \odot A_2$.

2. (3 points.) Prove that the class of regular languages is closed under $\odot$ in a different way: by reasoning from the fact that the regular languages are closed under union, intersection, Kleene star, and complement (you may assume these facts without proof.)

Let L_1 and L_2 be regular languages; i.e., there exist DFAs D_1 and D_2 that recognize them.

By the closure of regular languages under complement, $\overline{L_1}$ and $\overline{L_2}$ are also regular languages. By the closure of regular languages under intersection, $L_1 \cap L_2$ and $\overline{L_1} \cap \overline{L_2}$ are also regular languages. Finally, by the closure of regular languages under union,

$$(L_1 \cap L_2) \cup (\overline{L_1} \cap \overline{L_2}) = L_1 \odot L_2$$

is regular.

Rationale: The goal of this question is to get comfortable building DFAs programmatically (combining two generic DFAs using their formal definitions), and also thinking about the logic of closure (if X and Y are regular and the regular languages are closed under $\clubsuit$, then $X \clubsuit Y$ is regular).

References: Sipser pp.45-46 (proof that the regular languages are closed under the union operation), Wiki: Closure.

5 Problem 5 (9 points)

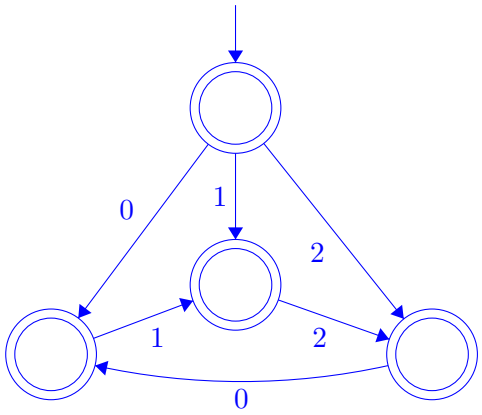
1. (9 points). Consider the language L on the alphabet $\{0, 1, 2\}$ consisting of all strings that meet both of the following conditions:

   - Every 0 is immediately followed by a 1, every 1 is immediately followed by a 2, and every 2 is immediately followed by an 0.
   - The string starts and ends with the same digit.

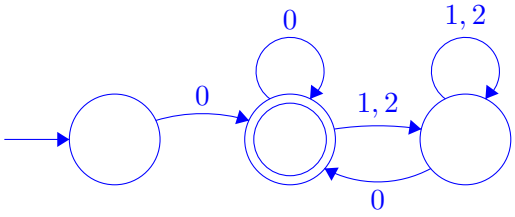
   For example, the strings '1', '0120120', and '1201' are in the language, while '111', '210', and '012012' are not.

   (a) (6 points). Show that the following languages are regular:

   i. L_{inc}, the language of strings over $\{0, 1, 2\}$ such that every 0 is immediately followed by a 1, every 1 is immediately followed by a 2, and every 2 is immediately followed by an 0.
   The following NFA recognizes L_{inc}. Note that the NFA accepts the empty string ϵ and, once we reach the bottom three nodes, any branch that fails our condition dies.



   ii. L_0, L_1, and L_2, the languages of strings over $\{0, 1, 2\}$ that start and end in 0, 1, and 2 respectively.
   The NFA below accepts L_0 (the NFAs for L_1 and L_2 are similar). Our computation dies if it does not start with 0; afterwards, we occupy the accept state if and only if we have just read in a 0.



   (b) (3 points). Use closure properties of the regular languages (closure under union, intersection, concatenation, etc), along with your results from the previous part to conclude that L is regular.

We've just proved that L_{inc}, L_0, L_1, and L_2 are regular. By definition string w is included in L if it is included in L_{inc} and in at least one of L_0, L_1, or L_2. By the closure of the regular languages over union and intersection, we conclude that

$$L = L_{inc} \cap (L_0 \cup L_1 \cup L_2)$$

is regular.

Rationale: The point of this question is to practice combining several types of regularity proofs including using DFAs, NFAs, and closure properties.

References: Sipser 1.2 pp. 47-53 (NFAs), the Lightning Review video on NFAs linked in the Course Skeleton, Wiki: Closure.

6 Problem 6 (1 point)

1. What was the muddiest (most confusing) point of class during week 1, in your opinion?
   (Any coherent response.)
2. What was your favorite part of class during week 1?
   (Any coherent response.)
3. (Optional) Any other thoughts? Thank you!

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