Homework 2

COMS W3261, Summer A 2022

This homework is due **Tuesday, 6/7/2022, at 11:59pm EST**. Submit to GradeScope (course code: 2KGDW8).

Grading policy reminder: LaTeX is preferred, but neatly typed or handwritten solutions are acceptable. 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.

- The website Overleaf (essentially Google Docs for LaTeX) may make compiling and organizing your .tex files easier. Here's a quick tutorial.
- 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 Problem 1 (14 points)

Examine each of the following regular expressions and write down the language it describes using set notation or 1-2 sentences. (Example: $01^+ = \{w \mid w$ consists of a single 0 followed by at least one 1$\}$ or "This regular expression describes the language of strings that consist of a single 0 followed by at least one 1".)

1. (1 point.) $(00)^+ \cup (11)^+$.

   This is the language of even-length, non-empty strings that contain all 0's or all 1's.

2. (1 point.) $a\Sigma^* a$, where Σ denotes the alphabet $\{a, b, c\}$.

   This is the language of all strings over the alphabet $\Sigma = \{a, b, c\}$ that have length at least 2 and that start and end with a.

3. (1 point) $(000 \cup 0000)^*$.

   This is the language of strings over $\Sigma = \{0\}$ that can be divided into substrings of length 3 or 4 (including the empty string). (This language contains all strings of 0's except 0, 00, and 00000, so we could also write this as $0^* \setminus \{0, 00, 00000\}$.)

4. (1 point) $0xHH$, where H denotes the set $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F\}$ and the alphabet is $\Sigma = H \cup \{x\}$.

   This is the language of all strings that start with '0x' and then contain two characters from H. This regular expression produces hexadecimal bytes.

5. (1 point) $1\big((B \cup D)(\epsilon \cup F \cup M) \cup (A \cup C)(\epsilon \cup E)\big)$, where the alphabet is

   $$\Sigma = \{1, 2, 3, 4, 5, 6, A, B, C, D, E, F, G, J, L, M, N, Q, R, W, Z\}.$$

   This regular expression produces the set

   $$\{1B, 1D, 1BF, 1BM, 1DF, 1DM, 1A, 1C, 1AE, 1CE\},$$

   all of which are subway routes from Columbia University to Washington Square Park. (This is a non-exhaustive list).

6. (1 point) $PS0202 \cup W\big(1004 \cup 3(134 \cup 203 \cup 251 \cup 261 \cup 998) \cup 4(111 \cup 7(01 \cup 05 \cup 71) \cup 995)\big)$, where the alphabet is $\Sigma = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, W, P, S\}$.

   This regular expression produces the course numbers offered by the CS department during Summer 2022. (Writing out the list of strings is also fine.)

Write regular expressions that evaluate to the languages given.

7. (1 point.) All strings over $\{a, b\}$, including the empty string, that can be divided into concatenated copies of the string a and the string bb. (For example, this language includes a, $bbaa$, $bbbba$, and $aabba$, but not aba or $babb$.)

   The expression $(a \cup bb)^*$ generates all strings, including ϵ, that can be made out of 'a' and 'bb'.

8. (2 points.) All strings over $\{a, b\}$, including the empty string, that can be divided into concatenated copies of the string a and the string bb **and don't contain the substring** aa. (For example, this language includes a and $bbbba$ but not $bbaa$, $aabba$, aba or $babb$.)

   Note that the set difference operator ($\setminus$) isn't part of our definition of a regular expression.

   This requires a little more thought, because our original regular expression includes strings like aa and aab.

   We'll consider two cases. If our string starts with a, then it must include at least one repetition of bb before we see another a. If it starts with b, it begins with at least one repetition of bb, and then the same pattern holds. So the regular expression

$$(\epsilon \cup a)(bb \cup bba)^* = (\epsilon \cup a)((bb)^+(\epsilon \cup a))^*$$

   should work. Another valid regular expression is

$$(bb)^* \cup (a(bb)^+)^* \cup ((bb)^+a)^* \cup a((bb)^+a)^* \cup bb(a(bb)^+)^*.$$

   Any solution should include ϵ, a, all strings in $(bb)^*$, and all strings that alternate single a's with $(bb)^+$, including those starting and ending with (a, b), (b, a), (a, a), and (b, b).

   Be careful of constructions like $(bb \cup abb \cup bba)^*$: this doesn't work, as it allows strings like $bbaabb$.

9. (2 points.) Consider the alphabet $\Sigma = \{\rightarrow, \underset{0}{\rightarrow}, \underset{1}{\rightarrow}, \bigcirc, \odot\}$. Here $\rightarrow$ indicates a start symbol, $\underset{0}{\rightarrow}$ and $\underset{1}{\rightarrow}$ indicate labeled transitions, $\bigcirc$ indicates a reject state, and $\odot$ indicates an accept state. (So the string $\rightarrow \bigcirc \underset{0}{\rightarrow} \bigcirc \underset{0}{\rightarrow} \odot$ corresponds to an NFA that accepts only the string '00', and $\rightarrow \odot$ corresponds to an NFA that accepts only the string ϵ.)

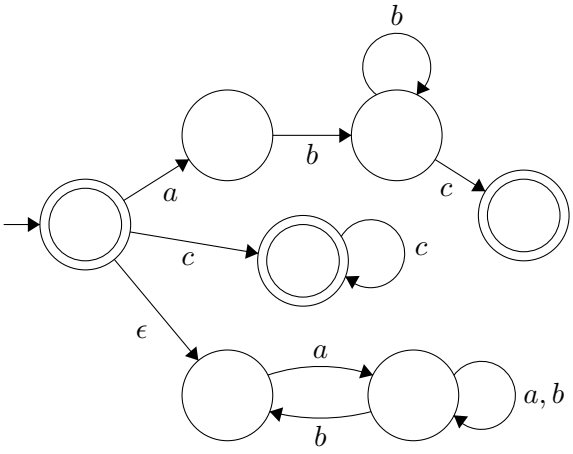
   Write a regular expression that corresponds to the language of *all* NFAs that accept a single string over the alphabet $\{0, 1\}$.

   We can do this with the regular expression

$$\rightarrow (\bigcirc(\underset{0}{\rightarrow} \cup \underset{1}{\rightarrow}))^* \odot,$$

   as any NFA that recognizes the string $w \in \{0, 1\}^n$ consists of a start arrow, n reject states with a single labeled transition exiting, and then an accept state.

10. (3 points.) Write a regular expression equivalent to the language recognized by the pictured NFA, which accepts strings over $\Sigma = \{a, b, c\}$. (You can do this by converting $NFA \rightarrow DFA \rightarrow GNFA \rightarrow$ regular expression, but it will be more efficient to reason directly or take shortcuts to simplify where possible.)



There are three possible transitions from the start state of this NFA, and we'll consider strings accepted by each in turn.

(a) If a branch begins computation by reading in an a, it enters the top sequence of three states, and accepts if the input string has the form $abb^*c = ab^+c$.

(b) If a branch begins computation by reading in a c, we accept if the input string has the form $cc^* = c^+$.

(c) If a branch begins computation by following the ϵ-transition, it will reject no matter what: there are no accept states in this direction!

(d) Finally, we should note that we'll accept the string ϵ because the start state is an accept state.

The resulting regular expression is

$$(ab^+c) \cup (c^+) \cup \epsilon = (ab^+c) \cup (c^*)$$

(full simplification is optional).

Rationale: The goal of this question is to make sure you're comfortable interpreting and building regular expressions (and reading NFAs).

References: Sipser pp. 63-66 (regular expressions, Lightning Review 3 (Regular Expressions); for 1.10 Sipser pp. 47-52; Lightning Review 2 (NFAs).

2 Problem 2 (6 points)

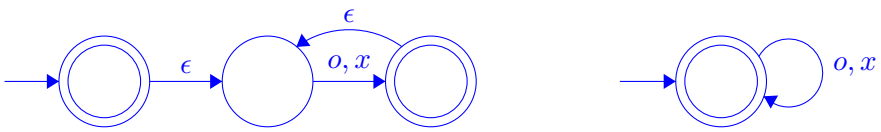
1. (6 points). Using the alphabet $\Sigma = \{o, x\}$, draw a state diagram for an NFA **with at most 4 states** that recognizes the regular expression

$$(x \cup o)^*(xx \cup oo).$$

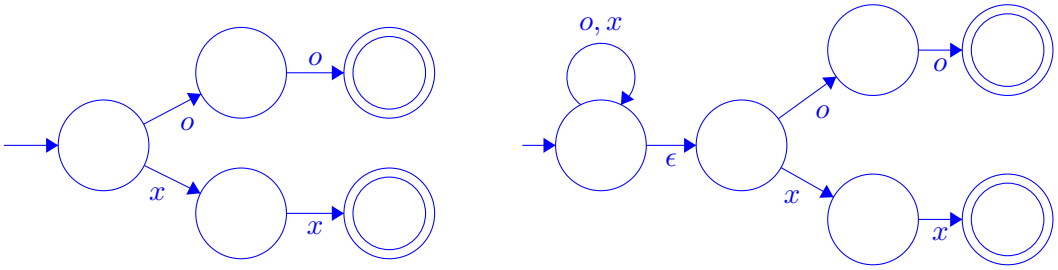
Explain in words why your NFA recognizes the language specified.

[Hint: Feel free to use our techniques for building NFAs that recognize languages defined with regular operations, e.g., our all-purpose method for building an NFA to recognize $A \circ B$ given NFAs that recognize A and B. These techniques don't necessarily produce an NFA with the minimum number of states, so you may need to simplify your NFA. (Testing strings is one way to check if your simplification does the same thing as the original.)]

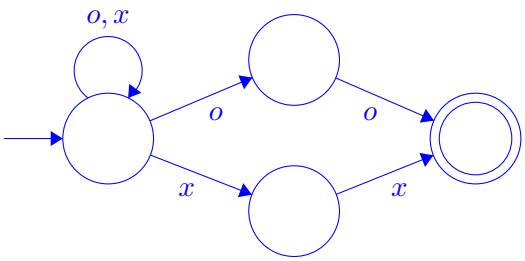
Naively combining an NFA that recognizes $\{x, o\}$ with the NFA transformation for * gives the NFA pictured below on the left. However, as $(x \cup o)^*$ is the language of all strings over $\{x, o\}$, we can simplify it to the one-state "yes DFA" on the right.



The NFA below on the left recognizes $(xx \cup oo)$; naively concatenating gives the NFA below on the right.



Finally, we observe that eliminating the ϵ-transition and merging the two accept states does not change the function of this NFA. Our final NFA is as pictured

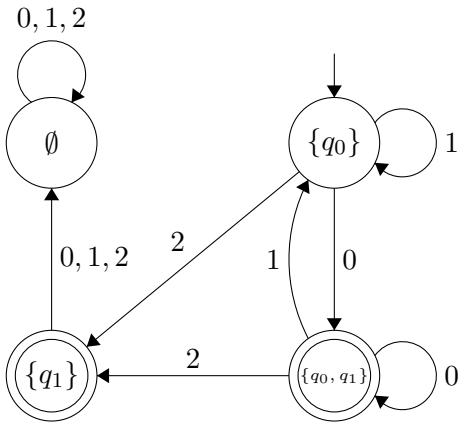


Rationale: The goal of this question is to practice building NFAs that recognize languages defined with regular operations, and simplifying the operations of NFAs.

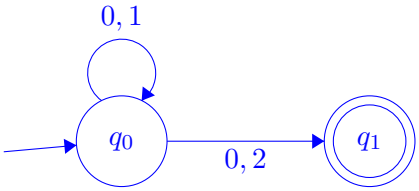
References: Sipser pp. 63-66 (NFAs), Lightning Review 2 (NFAs); Sipser pp. 59-63 (combining NFAs to recognize languages built with regular operations).

3 Problem 3 (6 points)

1. (6 points.) The DFA pictured below was created by converting from an NFA using the process described in class (also covered in video Example 2.) Work backwards to construct the original NFA and explain your reasoning. (You may assume that the original NFA has no ϵ-transitions.)



The original NFA is pictured below.



From the state set of the DFA, we can conclude that the original NFA has states $Q = \{q_0, q_1\}$. From the accept states, we conclude that q_1 and not q_0 is an accept state (if q_0 were an accept state in the NFA, $\{q_0\}$ would be an accept state in the DFA.) The edges that exit $\{q_0\}$ and $\{q_1\}$ in the DFA immediately imply the four edges below.

Rationale: The goal of this question is to practice thinking about DFAs that simulate other automata, as well as the specific process of converting an NFA into an equivalent DFA.

References: See Sipser pp. 54-58 (Converting DFAs to NFAs), Example 2 (Converting NFAs to DFAs).

4 Problem 4 (2 extra credit points)

Find a cool article or paper related to theoretical computer science and write about it!

In addition to a short summary, please include why you chose the article, what you found most interesting about it, and its potential impact. Please provide a link to your chosen article or paper, and write around 300 words. The article or paper you choose does not have to be famous or revolutionary in any way. However, please do not choose a paper that you have worked on.