CPSC 421: Introduction to Theory of Computing
Assignment #2, due Tuesday September 20th by 11:59pm, via GradeScope

[6] 1. Assume that $\Sigma = \{0, 1\}$.

[1] a. What is the definition of **the language accepted by a finite automaton**. You do not need to explain the definition of whether a finite automaton accepts a string. (My answer is one sentence long.)

[1] b. Using the 5-tuple definition of a finite automaton, write down the definition of a finite automaton M_{every} that accepts every string using the alphabet Σ. (My answer is one sentence long.)

[4] c. Let L be an arbitrary language (again, using the alphabet Σ). For every string $x \in L$, the string x is accepted by M_{every}. Does this imply that L is regular? Explain why or why not.

[8] 2. For each of the following languages, provide a DFA that accepts it. You should define each DFA by drawing it as a directed graph with accepting states marked by double concentric circles. You do not need to justify your answers.

Your DFA can have undefined transitions that implicitly go to a rejection state.

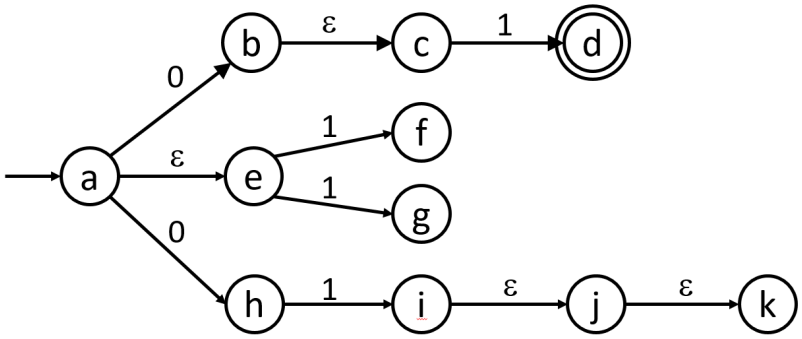
Your solution does not need to give the smallest possible DFA, but marks may be deducted if your solution is unnecessarily complicated. For each of the questions there is a DFA with less than 10 states.

[4] a. The language $L_2 = \{ x \in \{0,1\}^* : x \text{ is nonempty and begins and ends with the same symbol } \}$. (The language L_2 includes both strings of length 1.)

[4] b. The language $L_3 = \{ x \in \{0,1\}^* : \text{the 3rd from last symbol in } x \text{ is 1 } \}$. For example, $000100 \in L_3$ but $001000 \notin L_3$.

[5] 3. In Lecture 3 we discussed a construction that converts an NFA $M = (Q, \Sigma, \delta, q_0, F)$ into an equivalent DFA $M' = (2^Q, \Sigma, \delta', E(q_0), \{ S : S \cap F \neq \emptyset \})$ where $\delta'(S, a) = E\big(\bigcup_{s \in E(S)} \delta(s, a)\big)$ and $E(S)$ contains all states reachable from S by following any number of ϵ-transitions.

[1] a. For the following example (the same as given in lecture), how many states does the resulting DFA have?



[4] b. For the same example, draw the DFA resulting from the construction, but *only draw the states that are connected to its start state.*

Note: Connected doesn't mean by just a single transition, it means those reachable by a path of multiple transitions.

[10] 4. For each claim below, state whether it is true or false, and prove your answer. (A and B can be arbitrary languages, not just regular languages.)

[5] a. Is it true that, for all languages A and B, we have $(A^* \cap B^*)^* = (A \cap B)^*$?

[5] b. Is it true that, for all languages A and B, we have $(A^* \cup B^*)^* = (A \cup B)^*$?

[10] 5. **Efficiency of Regular Expression Software**

This question is about the performance for standard software for matching regular expressions. You can choose if you want to do it with Python, Perl, or Java. Consider this these program:

- in Python: http://www.cs.ubc.ca/~nickhar/F22/PythonTest.py
- in Perl: http://www.cs.ubc.ca/~nickhar/F22/PerlTest.pl
- in Java: http://www.cs.ubc.ca/~nickhar/F22/JavaTest.java

The program contains a string `text` and a regular expression `pattern`. Both the text and the pattern are strings containing 77 characters. The program tests if the text matches the pattern; if so, it outputs "`Match`".

Measure how much time it takes for Python (or Perl, or Java) to perform this regular expression match. (Don't use your watch — what Unix/Python/Perl/Java tools can do this for you?) By modifying the code and trying similar strings of different lengths, formulate a hypothesis of the following form:

> Perl/Python/Java's regular expression matching code requires time at least $\Omega(\cdots)$ to match a text and pattern, each of which has length exactly n.

[1] a. What are the strings that you used?

[8] b. What is your hypothesis? How did you derive it? Provide data supporting it.

[1] c. Theorize on why the regular expression software has that sort of efficiency. (You may wish to refer to question 3.)

[4] 6. **OPTIONAL BONUS QUESTION:** Let $\Sigma = \{0, 1\}$. Let

$$SS_k = \{\, ss \,:\, s \in \Sigma^* \text{ and } s \text{ is of length } k \,\}\,.$$

(Here ss means s concatenated with itself, so all strings in SS_k have length $2k$ not k.)

[1] a. Show that, for each k, any DFA that recognizes SS_k must have at least 2^k states. The solution must be a proof, not a heuristic argument.

[2] b. Let

$$L_k \;=\; \{\, xy \,:\, x, y \in \Sigma^* \text{ and } |x| = |y| = k \text{ and } x \neq y \,\}\,.$$

Describe an NFA that recognizes the language L_k and has a number of states that is polynomial in k.

[1] c. Conclude that, for infinitely many n, there is a language L such that, the smallest DFA recognizing L has n states, and the smallest NFA recognizing L has polylog(n) states.