CS/ECE 374 Sec A ✧ Spring 2025

Homework 3

Due Wednesday, Feb 12th, 2025 at 9am

- You can work in a group of up to **three** students. Read the instructions on the course website for additional details.
- **Submit your solutions electronically on the course Gradescope site as PDF files.** Submit a separate PDF file for each numbered problem. If you plan to typeset your solutions, please use the LaTeX solution template on the course web site. If you must submit scanned handwritten solutions, please use a black pen on blank white paper and a high-quality scanner app (or an actual scanner, not just a phone camera).
- Late submissions will be accepted (for 75% credit) until midnight the day of the deadline.

☞ Some important course policies ☜

- **You may use any source at your disposal**—paper, electronic, or human—but you ***must*** cite ***every*** source that you use, and you ***must*** write everything yourself in your own words. See the academic integrity policies on the course web site for more details including the policy on using AI tools.
- **Avoid the Two Deadly Sins!** Any homework or exam solution that breaks any of the following rules will be given an ***automatic zero***, unless the solution is otherwise perfect. Yes, we really mean it. We're not trying to be scary or petty (Honest!), but we do want to break a few common bad habits that seriously impede mastery of the course material.
  - Always give complete solutions, not just examples.
  - Always declare all your variables, in English. In particular, always describe the specific problem your algorithm is supposed to solve.

See the course web site for more information.

If you have any questions about these policies,
please don't hesitate to ask in class, in office hours, or on Ed.

---

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 $\Sigma = \{\texttt{0}, \texttt{1}\}$.

   (a) All strings that contain `0110` or end in `101`.

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

   (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 $\{xbyb | x \in \Sigma^*, b \in \Sigma, y \in \Sigma^{372}\}$.

   (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.

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 $w = w_1 w_2 \ldots w_n$ is a non-empty string, $\text{shift}(w) = w_n w_1 \ldots w_{n-1}$, 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.

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

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

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

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

   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.

3. **Not for submission:**

   (a) Draw an NFA for the regular expression $(010)^* + (01)^* + 0^*$. (If you use Thompson's algorithm for this part, you may want to simplify out some extraneous states to make the next part less tedious.)

   (b) Now using the powerset construction (also called the subset construcion), design a DFA for the same language. Label the states of your DFA with names that are sets of states of your NFA. You should use the incremental construction so that you only generate the states that are reachable form the start state.

4. **Not for submission:** In this problem, we explore what happens when we take the transformation tricks we know for DFAs and attempt to apply them to NFAs instead.

   (a) For an NFA $N = (Q, \Sigma, \delta, s, A)$, let $N' = (Q, \Sigma, \delta, s, Q - A)$. Give an example of an NFA N such that $L(N') \neq \Sigma^* - L(N)$. (This shows that unlike with DFAs, "reversing" the accept / reject states in an NFA does not necessarily cause it to decide the complement of the original language.)

   (b) Given two NFAs $N_1 = (Q_1, \Sigma, \delta_1, s_1, A_1)$ and $N_2 = (Q_2, \Sigma, \delta_2, s_2, A_2)$, we want to define a "product NFA" $N = (Q, \Sigma, \delta, s, A)$ that accepts $L(N_1) \cap L(N_2)$. As with DFAs, we can take $Q = Q_1 \times Q_2$, $s = (s_1, s_2)$, and $A = \{(p, q) | p \in A_1 \wedge q \in A_2\}$. Show how to define δ (being careful with the syntax!), and argue that the resulting definition of N correctly decides $L(N_1) \cap L(N_2)$.

   (c) Suppose that in the previous part, we took $A = \{(p, q) | p \in A_1 \wedge q \notin A_2\}$. Give an example of NFAs N_1 and N_2 where the resulting N (using the same δ as in the previous part) has $L(N) \neq \{w | w \in L_1 \wedge w \notin L_2\}$. (This shows that unlike with DFAs, the product construction doesn't work with all possible boolean functions.)

Solved problem

5. Let L be an arbitrary regular language. Prove that the language $\mathit{half}(L) := \{w \mid ww \in L\}$ is also regular.

Solution: Let $M = (\Sigma, Q, s, A, \delta)$ be an arbitrary DFA that accepts L. We define a new NFA $M' = (\Sigma, Q', s', A', \delta')$ with ε-transitions that accepts $\mathit{half}(L)$, as follows:

$$Q' = (Q \times Q \times Q) \cup \{s'\}$$
$$s' \text{ is an explicit state in } Q'$$
$$A' = \{(h, h, q) \mid h \in Q \text{ and } q \in A\}$$
$$\delta'(s', \varepsilon) = \{(s, h, h) \mid h \in Q\}$$
$$\delta'((p, h, q), a) = \{(\delta(p, a), h, \delta(q, a))\}$$

M' reads its input string w and simulates M reading the input string ww. Specifically, M' simultaneously simulates two copies of M, one reading the left half of ww starting at the usual start state s, and the other reading the right half of ww starting at some intermediate state h.

- The new start state s' non-deterministically guesses the "halfway" state $h = \delta^*(s, w)$ without reading any input; this is the only non-determinism in M'.
- State (p, h, q) means the following:
  - The left copy of M (which started at state s) is now in state p.
  - The initial guess for the halfway state is h.
  - The right copy of M (which started at state h) is now in state q.
- M' accepts if and only if the left copy of M ends at state h (so the initial non-deterministic guess $h = \delta^*(s, w)$ was correct) and the right copy of M ends in an accepting state.

■

Rubric: 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, and if it is an NFA, whether it uses ε-transitions.
  - 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 $\varnothing$, or $\{\varepsilon\}$, 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.