Problem Set 6

This sixth problem set explores formal language theory, finite automata, the regular languages, and their properties. This will be your first foray into computability theory, and I hope you find it fun and exciting!

As always, please feel free to drop by office hours, ask on Ed, or email the staff list if you have any questions. We'd be happy to help out.

Good luck, and have fun!

Due Friday, October 30th at 12:00PM noon Pacific

Problem One: Constructing DFAs

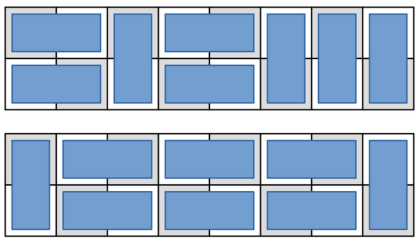
Download the starter files for Problem Set Six and extract them somewhere convenient on your system. Run the program to access the automaton editor, which you'll use throughout the problem set. The provided program offers this functionality:

- ***Automaton Editor:*** Like the Graph Editor and Relation Editor, use this tool to design automata.
- ***Automaton Tester:*** A tool you can use to test your automata on various input strings. Enter strings in the text box on the left, and the tool will show you which of those strings your automata accepts and which of those strings it rejects. You can follow each string with what you expect the automaton to do, and the tool will tell you whenever the expected output doesn't match the actual output. For example, you might start with these test strings for the automaton in part (i):
  - `BBIIB yes`
  - `BBB no`
- ***Automaton Debugger:*** A tool for single-stepping through the operation of an automaton on a string. You can use this to see how your automata process individual strings.

There's also an automated test suite you can use to check your work.

For each of the following languages over the indicated alphabets, construct a DFA that accepts precisely the strings that are in the indicated language. Your DFA does not have to have the fewest number of states possible, though for your own edification we'd recommend trying to construct the smallest DFAs possible. Use the Automaton Editor to compose your answers, and submit your finished automata to Gradescope.

i. There are many ways to tile a 2 × 8 checkerboard with dominoes, two of which are shown to the right. Notice that the horizontal dominoes must appear as stacked pairs (do you see why?) We can read each tiling from left to right as a string made from the characters `I` and `B`, where `I` denotes "a vertical domino" and `B` denotes "two horizontal dominoes." The top tiling here would be represented as `BIBIII` and the bottom tiling as `IBBBI`.



   Let Σ be the alphabet {`B`, `I`}. Construct a DFA for the language { *w* ∈ Σ* | *w* represents a domino tiling of a 2 × 8 checkerboard }.

ii. You're taking a walk with your dog along a straight-line path. Your dog is on a leash of length two, so the distance between you and your dog can be at most two units. You and your dog start at the same position. Consider the alphabet Σ = {`y`, `d`}. A string in Σ* can be thought of as a series of events in which either you or your dog moves forward one unit. For example, the string `yydd` means you take two steps forward, then your dog takes two steps forward.

   Let *L* be the language { *w* ∈ Σ* | *w* describes a series of steps where you and your dog are never more than two units apart }. Construct a DFA for *L*.

iii. Let Σ = {`a`, `b`}. Construct a DFA for the language *L* = { *w* ∈ Σ* | *w* contains the same number of instances of the substring `ab` and the substring `ba` }. Note that substrings are allowed to overlap, so we have `aba` ∈ *L* (one copy of each substring) and `babab` ∈ *L* (two copies of each substring).

iv. Let Σ = {`a`, `c`, `m`, `o`}. Construct a DFA for the language *L* = { *w* ∈ Σ* | *w* contains the word `cocoa` as a substring }. For example, `mmcocoamm` ∈ *L* and `cocoa` ∈ *L*, but `c` ∉ *L*, `cmomcmoma` ∉ *L* (though `cocoa` is a sub***sequence*** of `cmomcmoma`, it's not a sub***string***), and ε ∉ *L*.

   Some trickier cases to watch for: `cccocoa` ∈ *L*, and `cocococoa` ∈ *L*.

Problem Two: Constructing NFAs

For each of the following languages over the indicated alphabets, use the Automaton Editor to design an NFA that accepts precisely the strings that are in the indicated language. As before, while you don't have to design the smallest NFAs possible, we recommend that you try to keep your NFAs small both to make testing easier and for your own edification.

i. Let Σ = {a, b, c}. Construct an NFA for { $w \in \Sigma^*$ | w ends in a, bb, or ccc }.

While it's possible to do this completely deterministically, it's a bit easier if you use the "guess-and-check" framework we talked about in class.

ii. Let Σ = {a, b, c, d, e}. Construct an NFA for the language L = { $w \in \Sigma^*$ | the letters in w are sorted alphabetically }. For example, abcde ∈ L, bee ∈ L, a ∈ L and ε ∈ L, but decade ∉ L.

*You could do this deterministically, but that will require a **lot** of transitions. You can dramatically reduce that number by using ε-transitions strategically.*

iii. Let Σ = {a, b, c, d, e}. Construct an NFA for the language { $w \in \Sigma^*$ | the last character of w appears nowhere else in w, and $|w| \geq 1$ }.

This problem is all about embracing nondeterminism. Use the "guess-and-check" framework. What information would you want to guess? How would you check it? Please don't try to solve this problem by building a DFA for this language; you'll need at least 50 states if you tried to approach things this way.

Stuck? Try reducing the alphabet to two or three letters and see if you can solve that version.

iv. For the alphabet Σ = {a, b}, construct an NFA for the language { $w \in \Sigma^*$ | w contains at least two b's with exactly five characters between them }. For example, baaaaab is in the language, as is aabaabaaabbb and abbbbbabaaaaaaab, but bbbbb is not, nor are bbbab or aaabab.

The smallest DFA for this language is pretty big, so please don't solve this one deterministically. Embrace the nondeterminism! What would you like to guess? How would you check that your guess is right?

Problem Three: ℘(Σ*)

Let Σ be an alphabet. Give a short English description of the set ℘(Σ*). Briefly justify your answer.

We think that there is a single "best answer." You should be able to describe the set in at most ten words.

Problem Four: Much Ado About Nothing, Part II

We've covered a bunch of terminology in the past week. This problem is designed to help you make sense of what all these terms mean.

i. Are there any languages L where ε ∈ L? If so, give an example of one. If not, explain why not.

ii. Are there any languages L where ε ∉ L? If so, give an example of one. If not, explain why not.

iii. Are there any languages L where ε ⊆ L? If so, give an example of one. If not, explain why not.

iv. Are there any languages L where ε ⊈ L? If so, give an example of one. If not, explain why not.

v. Does Ø = ε? Briefly explain your answer.

vi. Does Ø = {ε}? Briefly explain your answer?

Problem Five: Hard Resets

A ***hard reset string*** for a DFA is a string w with the following property: starting from any state in the DFA, if you read w, you end up in the DFA's start state.

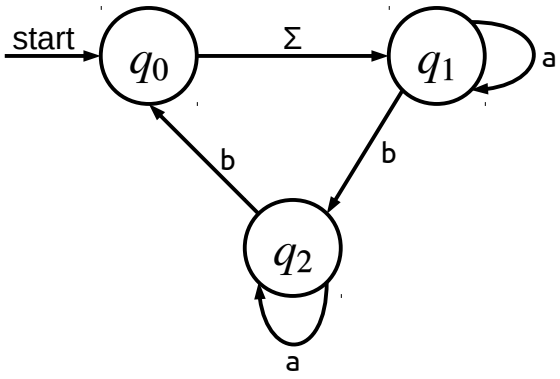
Hard reset strings have many practical applications. For example, suppose you're remotely controlling a Mars rover whose state you're modeling as a DFA. Imagine there's a hardware glitch that puts the Mars rover into a valid but unknown state. Since you can't physically go to Mars to pick up the rover and fix it, the only way to change the rover's state would be to issue it new commands. To recover from this mishap, you could send the rover a hard reset string. Regardless of what state the rover got into, this procedure would guarantee that it would end up in its initial configuration.

Here is an algorithm that, given any DFA, will let you find every hard reset string for that DFA:

1. Add a new start state q_s to the automaton with ε-transitions to every state in the DFA.
2. Perform the subset construction on the resulting NFA to produce a new DFA called the ***power automaton***.
3. If the power automaton contains a state corresponding solely to the original DFA's start state, make that state the only accepting state in the power automaton. Otherwise, make every state in the power automaton a rejecting state.

This process produces a new automaton that accepts all the hard reset strings of the original DFA. It's possible that a DFA won't have any hard reset strings (for example, if it contains a dead state), in which case the new DFA won't accept anything.

Apply the above algorithm to the following DFA and give us a hard reset string for that DFA. For simplicity, please give the subset-constructed DFA as a transition table rather than a state-transition diagram. We've given you space for the table over to the right, and to be nice, we've given you exactly the number of rows you'll need.



	a	b

Sample hard reset string: _____

*We **strongly** recommend reading the Guide to the Subset Construction before starting this problem.*

Finding a hard reset strings for this DFA is a lot easier if you take a few minutes to think about what the power automaton does.

Problem Six: Identities, Monoids, and Kleene Stars

This problem explores formal language theory from an algebraic perspective, culminating in a new perspective about the Kleene star. We'll begin with a new definition. An ***identity language*** is a language I where the following is true for all languages L:

$$IL = L \quad \wedge \quad LI = L$$

As a reminder, the notation IL means "the concatenation of languages I and L."

i. Give an example of an identity language. No justification is necessary.

ii. Prove that there is exactly one identity language. You're free to assume that the language you came up with in part (i) is indeed an identity language and don't need to prove this. (More generally, feel free to assume your language from part (i) is an identity language going forward.)

To prove there's at most one identity language, assume that both I_1 and I_2 are identity languages, then prove that $I_1 = I_2$. This shows I_1 and I_2 were different names for the same thing, not two different languages.

There's a special type of language called a ***monoid*** that pops up when exploring formal language theory. Specifically, we define monoids as follows: if M is a language over Σ, then

$$M \text{ is a monoid} \quad \text{if} \quad I \subseteq M \text{ and } MM \subseteq M.$$

Let's take a minute to get a handle on what monoids look like.

iii. Give us two examples of monoids, one of which contains finitely many strings and one of which contains infinitely many strings. No justification is required.

Your next task is to prove an important result about powers of languages.

iv. Let L be a language over some alphabet Σ. Prove, by induction, that $L^m L^n = L^{m+n}$ for all natural numbers m and n. Please call back to the formal definition of language powers:

$$L^0 = \{\varepsilon\} \qquad L^{n+1} = LL^n$$

Feel free to use the fact that language concatenation is associative: $A(BC) = (AB)C$.

While intuitively we think of L^k as "all strings made of k strings from L," the formal definition of L^k is the inductive one given here. Use that formal definition rather than the higher-level intuition.

This problem asks you to prove something by induction that involves two natural numbers m and n. Look at Problem Set Five for information about how to do this. As a hint, use induction on m, not n.

There is a deep, fundamental connection between Kleene stars and monoids.

v. Let L be an arbitrary language over Σ. Prove that L^* is a monoid over Σ. In the course of writing this proof, please call back to the formal definition of language concatenation and the Kleene star:

$$L^* = \{ w \mid \exists n \in \mathbb{N}.\ w \in L^n \}$$

$$L_1 L_2 = \{ x \in \Sigma^* \mid \exists w_1 \in L_1.\ \exists w_2 \in L_2.\ x = w_1 w_2 \}$$

This is a great spot to write out two columns, one for "what I'm assuming" and one for "what I need to show." Again, don't assume anything about language powers or concatenation without first proving it.

vi. Let L be a language over Σ and let M be a monoid over Σ. Prove that if $L \subseteq M$, then $L^* \subseteq M$.

This problem is all about finding the right way to formalize things. Break the task down into smaller pieces and make sure you organize everything in a way that makes the logical flow easy to read and rigorously covers all cases. Once you have the setup put together, dive in and fill out each section.

Parts (v) and (vi) of this problem collectively say "L^* is the language formed by adding the fewest strings to L to make it a monoid." To see why, note that L^* is a monoid, and we can't take any strings out of it and be left with a monoid, since L^* is a subset of all monoids containing L. That's a very different perspective on the Kleene star than what we started with!

Problem Seven: DFAs, Formally

We've drawn DFAs both as state-transition diagrams (with circles for states and arrows for transitions) and as tables with rows for states and columns for characters. But what exactly *is* a DFA, in a mathematical sense? Formally speaking, a DFA is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where

- Q is a finite set, the elements of which we call *states*;
- Σ is a finite, nonempty set, the elements of which we call *characters*;
- $\delta : Q \times \Sigma \to Q$ is the ***transition function***, described below (δ is a lower-case Greek delta);
- $q_0 \in Q$ is the start state;
- $F \subseteq Q$ is the set of accepting states.

When drawing DFAs, we've represented transitions either by arrows labeled with characters or as a table with rows and columns corresponding to states and symbols, respectively. In this formal definition, the transition function δ is what ultimately specifies the transition. Specifically, for any state $q \in Q$ and any symbol $a \in \Sigma$, the transition from state q on symbol a is given by $\delta(q, a)$.

i. Consider the following 5-tuple definition of a DFA:

$D = (Q, \Sigma, \delta, q_0, F)$, where

$Q = \{q_0, q_1, q_2, q_3\}$,

$\Sigma = \{\texttt{r}, \texttt{s}\}$,

$$\delta(q_k, a) = \begin{cases} q_{k+1} & \text{if } a = r \text{ and } k \neq 3 \\ q_k & \text{if } a = r \text{ and } k = 3 \\ q_0 & \text{otherwise} \end{cases}, \text{ and}$$

$F = \{q_0, q_1, q_2\}$.

Use the DFA editor to draw this exact DFA.

Piece this apart one step at a time. How many states does this DFA have? Which one is the start state? What is the alphabet, and how many characters are in it? Which states are accepting?

The trickiest part is decoding the transitions. Begin with the start state. Pick any character you want. Where do you go when you see that character? See if you can take things from there.

Once you're done, a good question to ponder, but not something you need to submit: what is the language of this DFA?

ii. Consider the following 5-tuple definition of a DFA:

$D = (Q, \Sigma, \delta, q_0, F)$, where

$Q = \{q_0, q_1, q_2\}$,

$\Sigma = \{\texttt{r}, \texttt{s}\}$,

$$\delta(q_k, a) = \begin{cases} q_{(3k^2 - 7k + 4)/2} & \text{if } a = r \\ q_{(-3k^2 + 5k + 2)/2} & \text{if } a = s \end{cases}, \text{ and}$$

$F = \{q_0\}$.

Use the DFA editor to draw this exact DFA.

Again, we recommend thinking over what the language of this DFA is once you've drawn it out. It's a good exercise.

Problem Eight: Concatenation, Kleene Stars, and Complements

This problem is all about transformations on languages and the effects they have.

i. Prove or disprove: if L is a nonempty, finite language and k is a positive natural number, then $|L^k| = |L|^k$. Here, the notation $|L|^k$ represents "the cardinality of L, raised to the kth power," and the notation $|L^k|$ represents "the cardinality of the k-fold concatenation of L with itself."

Call back to formal definitions, and proceed carefully. What would you would need to show in order to prove this statement? To disprove it? Explore both branches.

ii. Prove or disprove: there is a language L where $\overline{(L^*)} = (\overline{L})^*$.

A good warm-up problem: what is Ø? Don't answer this question based off of your intuition; look back at the formal definition of the Kleene star.*

Optional Fun Problem: Why Finite?

A ***deterministic infinite automaton***, or ***DIA***, is a generalization of a DFA in which the automaton has infinitely many different states. Formally speaking, a DIA is given by the same 5-tuple definition as a DFA, except that Q must be an infinite set. Since DIAs have infinitely many states, they're mostly an object of purely theoretical study. You couldn't actually build one in the real world.

Prove that if L is an arbitrary language over some alphabet Σ, then there is a DIA that accepts L (that is, the DIA accepts every string in L and rejects every string not in L.) To do so, show how to start with a language L, formally define a 5-tuple corresponding to a DIA for L, then explain why that DIA accepts all the strings in L and nothing else.