Problem 0

a)

The formal description of a DFA M is $(\{q_1, q_2, q_3, q_4, q_5\}, \{\mathtt{u}, \mathtt{d}\}, \delta, q_3, \{q_3\})$, where δ is given by the following table. Give the state diagram of this machine.

	u	d
q_1	q_1	q_2
q_2	q_1	q_3
q_3	q_2	q_4
q_4	q_3	q_5
q_5	q_4	q_5

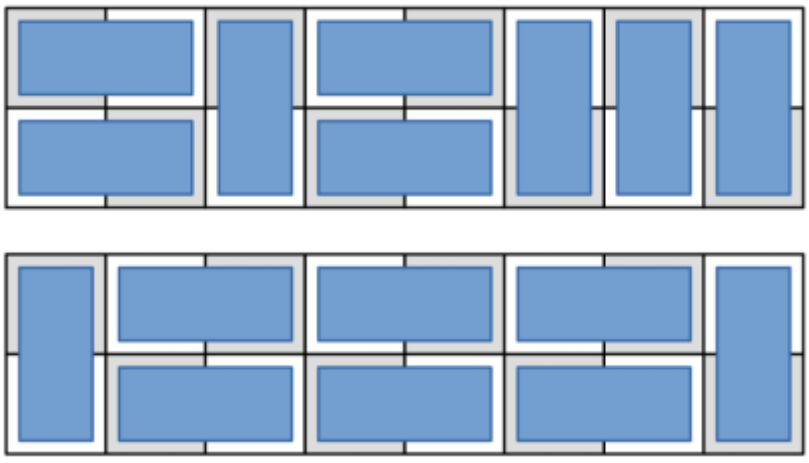
b) Each of the following languages is the intersection of two simpler languages. In each part, construct DFAs for the simpler languages, then combine them to give the state diagram of a DFA for the language given. In all parts, Σ = {a, b}.

- {w| w has at least three a's and at least two b's}.
- {w| w has exactly two a's and at least two b's}
- {w| w has an even number of a's and one or two b's}
- {w| w starts with an a and has at most one b}

Problem 1:

For each of the following languages over the indicated alphabets, construct a DFA that accepts precisely the strings that are in the indicated language. Though minimal is not required, we'd recommend trying to construct the smallest DFAs possible.

a) There are many ways to tile a 2×8 checkerboard with dominoes, two of which are shown here:



Notice that the horizontal dominoes must appear as stacked pairs. 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 { $w \in \Sigma^*$ | w represents a domino tiling of 2×8 checkerboard }.

b) 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 { $w \in \Sigma^*$ | w describes a series of steps where you and your dog are never more than two units apart }. Construct a DFA for L.

c) Let Σ={a,b}. Construct a DFA for the language L = { $w \in \Sigma^*$ | 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$\in$L (one copy of each substring, ab and ba) and bababb$\in$L (two copies of each substring).

d) Let Σ={a,c,m,o}. Construct a DFA for the language L = { $w \in \Sigma^*$ | w contains the word **cocoa** as a substring }. For example, mmcocoamm$\in$L and cocoa$\in$L, but c$\notin$L, cmomcmoma$\notin$L (though **cocoa** is a sub*sequence* of **cmomcmoma**, it's not a sub*string*), and $\varepsilon \notin$L. Some trickier cases to watch for: ccccocoa$\in$L, and cococoa$\in$L.

Problem 2:

For each of the following languages over the indicated alphabets, use the Design an NFA that accepts precisely the strings that are in the indicated language. While you don't have to design the smallest NFAs possible, try to keep your NFAs small.

Remember the "guess-and-check" design framework we talked about in class. Also consider if you can use the NFA design pattern of decomposing the language into the union of 2 or more simpler languages that it would be easy to make an NFA/DFA for, and combining them. Some of these will involve a lot of states (in one case, over 50!) and/or transitions if you do them without the special powers of an NFA. Think about what information it would be useful to "guess" in advance, and design around that.

a) Let Σ={a,b,c}. Construct an NFA for { $w \in \Sigma^*$ | w ends in **a**, **bb**, or **ccc** }.
b) 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$\in$L, bee$\in$L, a$\in$L and $\varepsilon \in$L, but decade$\notin$L.
c) 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 $|w| \geq 1$ }.
d) Let Σ={a,b}. Construct an NFA for the language L = { $w \in \Sigma^*$ | w contains at least two **b**'s with exactly five characters between them }. For example, baaaaab$\in$L, aabaabaaabbb$\in$L, and abbbbbabaaaaaaab$\in$L, but bbbbb$\notin$L, bbbab$\notin$L, and aaabab$\notin$L.

Problem 3:

a) Are there any languages L where $\varepsilon \in L$? If so, give an example of one. If not, explain why not.
b) Are there any languages L where $\varepsilon \notin L$? If so, give an example of one. If not, explain why not.
c) Are there any languages L where $\varepsilon \subseteq L$? If so, give an example of one. If not, explain why not.
d) Are there any languages L where $\varepsilon \not\subseteq L$? If so, give an example of one. If not, explain why not.
e) Does $\varnothing = \varepsilon$? Briefly explain your answer.
f) Does $\varnothing = \{\varepsilon\}$? Briefly explain your answer.

Problem 4

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.

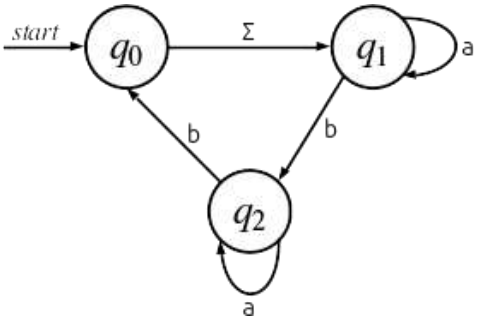
Hard reset strings have many practical applications. For example, suppose you're remotely controlling a Mars rover whose state you're modeling as a DFA. Imagine there's a hardware glitch that puts the Mars rover into a valid but unknown state. Since you can't physically go to Mars to pick up the rover 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 qs 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.



Sample hard reset string: _____.

Problem 5:

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 $(Q,\Sigma,\delta,q0,F)$, 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);
- $q0\in Q$ is the start state; and
- $F\subseteq Q$ 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 $\delta(q,a)$.

a) Consider the following 5-tuple definition of a DFA:$D=(Q,\Sigma,\delta,q0,F)$, where

$Q=\{q0,q1,q2,q3\}$,

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

δ is defined as follows:$\delta(qk,a)=\{qk+1 \text{if } a=r \text{ and } k\neq 3 qk \text{if } a=r \text{ and } k=3 q0 \text{otherwise}$

$F=\{q0,q1,q2\}$

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 submit: what is the language of this DFA?

b) Consider the following 5-tuple definition of a DFA: D=(Q,Σ,δ,q0,F), where
   A. Q={q0,q1,q2},
   B. Σ={r,s},
   C. δ is defined as follows:

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

   D. F={q0}.

Again, we recommend thinking over what the language of this DFA is once you've drawn it out, but it's not required.

Problem 6:

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

a) 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."

This is a "prove or disprove" problem, which means that your first task is to figure out whether this statement is true (in which case you'll prove it) or false (in which case you'll disprove it). To disprove a result, you'll write a proof of its negation. Typically, a disproof begins with wording like "We will prove the negation of the claim, specifically, _____" and then proceeds as if it were a regular proof of the negation.

Since you have to first decide whether this statement or its negation is true, write out both statements and explore each branch to see what you find.

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

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

To prove the statement S=T where S and T are sets, prove that $S \subseteq T$ and that $T \subseteq S$. To disprove the statement S=T, prove that there is some x where either $x \in S$ and $x \notin T$ or $x \in T$ and $x \notin S$.