CS 601 Spring 2023: Problem Set 1.

Problem 1. (10 points) Give regular expressions to generate each of the following languages.

a) $\{w \in \{a,b\}^* : \ w \text{ does not end in } ba\}$
b) $\{w \in \{0,1\}^* : w = \alpha \circ \beta \ \text{ and } \alpha \text{ has an even number of } 1\text{'s and } \beta \text{ has an even number of } 0\text{'s}\}$

Problem 2. (10 points) For any string $w = w_1 w_2 \cdots w_n$, the *reverse* of w, written w^R, is the string w in reverse order, $w_n \cdots w_2 w_1$. For any language A, let $A^R = \{w^R | w \in A\}$. Show that if A is regular, so is A^R.

Problem 3. (10 points) In this problem you will design an FSA that checks if the sum of two numbers equals a third number. Each number is an arbitrarily long string of bits. At each step, the input to the FSA is a symbol that encodes 3 bits, one from each number. In other words, the alphabet of the FSA is:

$$\Sigma = \left\{ \begin{bmatrix}0\\0\\0\end{bmatrix}, \begin{bmatrix}0\\0\\1\end{bmatrix}, \begin{bmatrix}0\\1\\0\end{bmatrix}, \begin{bmatrix}0\\1\\1\end{bmatrix}, \begin{bmatrix}1\\0\\0\end{bmatrix}, \begin{bmatrix}1\\0\\1\end{bmatrix} \begin{bmatrix}1\\1\\0\end{bmatrix}, \begin{bmatrix}1\\1\\1\end{bmatrix} \right\}$$

In the input string $\begin{bmatrix}0\\0\\1\end{bmatrix}\begin{bmatrix}1\\0\\1\end{bmatrix}$ the bits in the top row represent the number 01, the bits in the middle row represent 00 and the third row represents 11. In this case, since 01+00 ≠ 11, the FSA must reject the input. On the other hand, the input string $\begin{bmatrix}0\\0\\1\end{bmatrix}\begin{bmatrix}0\\1\\0\end{bmatrix}\begin{bmatrix}1\\1\\0\end{bmatrix}$ will be accepted by the FSA since $001 + 011 = 100$.

Formally, let $B = \{w \in \Sigma^* : the\ bottom\ row\ of\ w\ equals\ the\ sum\ of\ the\ top\ two\ rows\}$, where Σ^* represents all finite strings over the alphabet Σ. Your goal is to design an FSA for the language B.

To get started, first design a 2-state FSA that recognizes B^R – this should be straightforward because the input arrives least significant bits first and most significant bits at the end. Next, use the technique of Problem 2 to design the final FSA for B.

Problem 4. (50 points) In this problem we will develop an algorithm to minimize the number of states of a DFA. Specifically, given a DFA M that accepts the language $L(M)$ our aim is to find a DFA M' such that $L(M') = L(M)$ and such that no DFA that also accepts $L(M)$ has fewer states than M'.

The minimization algorithm proceeds in two stages:

1. Eliminate the states that are unreachable from the start state. Clearly, removing these states and their associated transitions does not change the language accepted. This process is carried out

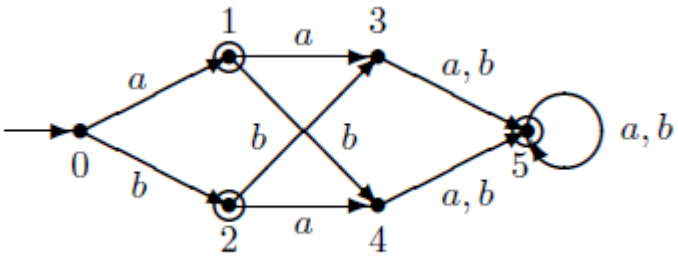
using a depth-first search on the transition graph and starting from the start state. This procedure is very fast, taking time linear in the size of the transition graph (number of states plus number of transitions).

2. Collapse "equivalent" states. If two states are equivalent (we'll define this shortly), then combine them into a single state. Repeat this step until no two states are equivalent.

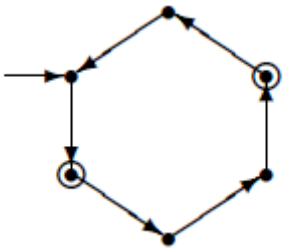
To proceed, we need to define what it means for two states to be equivalent and how we can recognize equivalent states. Next, we'll need to figure out how equivalent states can be merged into a single state without changing the language accepted. Finally, we'll need to prove that when no two states are equivalent the resulting DFA has the minimum number of states required to accept the original language.

Part 1. (3 points) Consider the 6-state DFA shown below (states 1, 2, 5 are accept states).

a. Would you consider states 3 and 4 equivalent? If so, merge them into a single state. Show the resulting DFA. Can you further reduce the number of states?



b. Next, consider the following DFA. The alphabet contains only the symbol a, so you should assume that every transition is labeled with a.



i. What is the language accepted by this automaton?
ii. What is the minimum state DFA for the language of this automaton?

It should be clear that we will never merge an accepting state $p \in F$ with a non-accepting state $q \notin F$. Since every state is reachable from the start state, there are strings $x, y \in \Sigma^*$ such that $\Delta(q_0, x) = p,\ \Delta(q_0, y) = q$. If we merged $p\ and\ q$, then both x, y would either be accepted, or both rejected, and the language would change.

Part 2. (5 points) Generalize the preceding argument to prove that we cannot merge states p, q if there exists a string $x \in \Sigma^*$ such that $\Delta(p, x) \in F$ and $\Delta(q, x) \notin F$. (Hint: Induction on the length of x.)

As we will see, it turns out that this yields a condition that is both necessary and sufficient for merging states. Formally, if $\forall x \in \Sigma^*: (\Delta(p, x) \in F \iff \Delta(q, x) \in F)$, then p, q are equivalent and can be merged.

We begin by formalizing the notion of equivalent states.

Part 3. (2 points) Define the relation $\sim$ over Q as follows:

$$p \sim q \stackrel{\text{def}}{=} \forall x \in \Sigma^*: (\Delta(p, x) \in F \leftrightarrow \Delta(q, x) \in F)$$

Prove that $\sim$ is an equivalence relation (i.e., it is reflexive, symmetric, and transitive).

Now we use the fact that every equivalence relation over a set partitions the elements of the underlying set into equivalence classes. Let $[p]$ denote the equivalence class of $p \in Q$, so that

$$[p] = \{q \in Q : q \sim p\}$$

Part 4. (5 points) Prove that $p \sim q \implies \delta(p, a) \sim \delta(q, a)$. Equivalently, if $[p] = [q]$ then $[\delta(p, a)] = [\delta(q, a)]$. (Hint: to proceed, let $a \in \Sigma, y \in \Sigma^*$. Show that $\Delta(\delta(p, a), y) \in F \iff \Delta(\delta(q, a), y) \in F$.) In plain English this simply says that if you start out in equivalent states then after the next symbol you will be in states that are equivalent.

The property we just established is crucial because it ensures that when we merge equivalent states, the resulting machine will be a DFA.

Next, consider the following machine $M' \stackrel{\text{def}}{=} (Q', \Sigma, \delta', s', F')$, where:

$$\begin{aligned} Q' &= \{[p] : p \in Q\} \\ \delta'([p], a) &= [\delta(p, a)] \\ s' &= [q_0] \\ F' &= \{[p] : p \in F\} \end{aligned}$$

Part 5. (2 points) Prove that $p \in F \iff [p] \in F'$. (Hint: the forward direction is immediate. For the converse, consider using the empty string for x in the definition of $\sim$.)

Part 6. (5 points) Prove that $\forall x \in \Sigma^*: \Delta'([p], x) = [\Delta(p, x)]$. Induct on the length of x.

Part 7. (3 points) Prove that $L(M') = L(M)$. Use the results of the last two parts in your proof.

Part 8. (5 points) Why can't M' be further reduced by repeating the process?

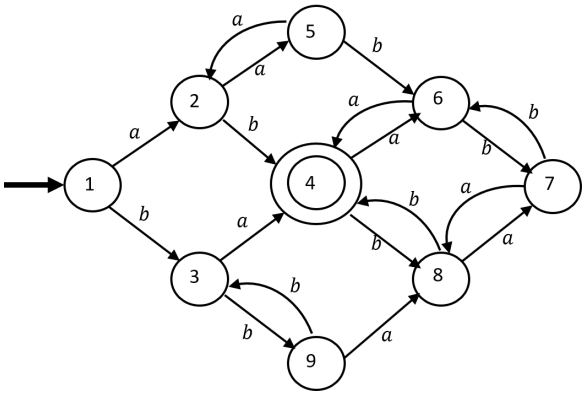
At this point, all we need to derive the minimization algorithm is to determine the equivalence classes. But determining if two states are equivalent isn't straightforward because the definition requires us to check infinitely many strings!

Here's the trick: instead of checking for pairs of equivalent states, we instead look for pairs of states that are NOT equivalent! In part 5 we showed that $p \sim q \;\Rightarrow\; \delta(p,a) \sim \delta(q,a)$. The contrapositive is $\delta(p,a) \not\sim \delta(q,a) \Rightarrow p \not\sim q$. So, starting with pairs of states in which one state is an accept state and the other is not, we can work backwards to determine all inequivalent pairs of states. Here is the final algorithm:

1. Make a table of pairs of states $\{p,q\}$. Initially leave all entries unmarked.[1]
2. Mark each entry $\{p,q\}$ if $(p \in F \;\wedge\; q \notin F) \vee (p \notin F \;\wedge q \in F)$
3. Scan the entire table, and if there is an unmarked pair $\{p,q\}$ such that $\{\delta(p,a), \delta(q,a)\}$ is marked for some $a \in \Sigma$, then mark $\{p,q\}$. If no mark was created during the scan, go to step 4. Otherwise repeat step 3.
4. After the final scan, if any pair $\{p,q\}$ is unmarked then merge p,q into one state. If pairs $\{p,q\}$ and $\{p,r\}$ are both unmarked, merge p,q,r into one state.
5. Use this to define M', the equivalent minimum state DFA as follows: if there was a transition labeled x from state p to state q in the original DFA, then draw a transition labeled x from the state that p was merged into to the state that q was merged into. The new start state is the one that the original start state got merged into.

Part 10. (10 points) Prove that the above algorithm is sound and complete. In other words, show that a pair of states is marked if and only if the two states are not equivalent. Hint: use induction.

Part 11. (10 points) Apply the DFA minimization algorithm to the DFA shown below. Show the minimized DFA, and construct a matrix indicating, for each distinguishable pair of states, the iteration in which it is checked off.



[1] Keep a matrix with one row and one column for each state; initially all entries are blank.