HW2 : Regular Languages and Automata Constructions

CSE105Sp22 Solutions

In this assignment,

You practiced designing multiple representations of regular languages and working with general constructions of automata to demonstrate the richness of the class of regular languages.

Reading and extra practice problems: Sipser Section 1.1, 1.2, 1.3. Chapter 1 exercises 1.4, 1.5, 1.6, 1.7, 1.8, 1.9, 1.10, 1.11, 1.12, 1.14, 1.15, 1.16, 1.17, 1.19, 1.20, 1.21, 1.22.

Assigned questions

1. (*Graded for correctness*[1]) Over the alphabet $\{a, b\}$, consider the language

   $$L = \{w \in \{a,b\}^* \mid (ab \text{ is a substring of } w) \wedge (ba \text{ is a substring of } w) \wedge (w \text{ starts with } a)\}$$

   In this question, you will use two different approaches to proving that this language is regular by building (different) DFA that recognize this language.

   (a) Design a DFA recognizing the language $\{w \in \{a,b\}^* \mid ab \text{ is a substring of } w\}$ and a DFA recognizing the language $\{w \in \{a,b\}^* \mid ba \text{ is a substring of } w\}$ and a third DFA recognizing the language $\{w \in \{a,b\}^* \mid w \text{ starts with } a\}$. Then, use the construction we discussed in class to combine these DFA to get a DFA that recognizes L. A complete solution will include the (clearly labelled) state diagrams for each of the three building-block DFAs, along with a description of the result of combining these DFAs that includes the formal definition of the resulting DFA and at the least the part of the state diagram that includes the start state, all the outgoing edges from the start state, and specifies how many states the full DFA will have.

   Solution: For definiteness, let's call the DFA recognizing the language $\{w \in \{a,b\}^* \mid ab \text{ is a substring of } w\}$,

   $$M_{ab} = (\{q0, q1, q2), \{a, b\}, \delta_{ab}, q0, \{q2\})$$

[1] This means your solution will be evaluated not only on the correctness of your answers, but on your ability to present your ideas clearly and logically. You should explain how you arrived at your conclusions, using mathematically sound reasoning. Whether you use formal proof techniques or write a more informal argument for why something is true, your answers should always be well-supported. Your goal should be to convince the reader that your results and methods are sound.

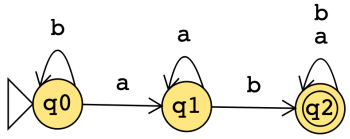
the DFA recognizing the language $\{w \in \{a,b\}^* \mid ba \text{ is a substring of } w\}$,

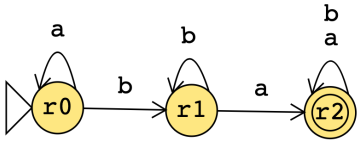
$$M_{ba} = (\{r0, r1, r2\}, \{a,b\}, \delta_{ba}, r0, \{r2\})$$

and the DFA recognizing the language $\{w \in \{a,b\}^* \mid w \text{ starts with } a\}$,

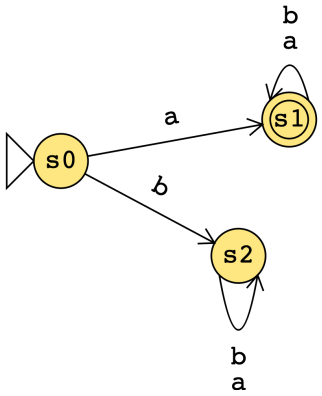
$$M_{ends} = (\{s0, s1, s2\}, \{a,b\}, \delta_{ends}, s0, \{s1\})$$

Each of the transition functions is given by the respective state diagrams below.





State diagrams for DFA recognizing the languages $\{w \in \{a,b\}^* \mid ab \text{ is a substring of } w\}$ (on the left) and $\{w \in \{a,b\}^* \mid ba \text{ is a substring of } w\}$ (on the right)



State diagram for DFA recognizing the language $\{w \in \{a,b\}^* \mid w \text{ starts with } a\}$

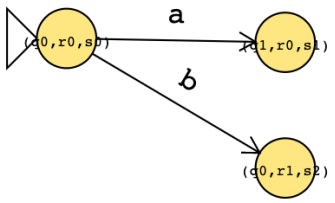
The formal definition of the DFA D, resulting from combining these DFAs is

$$D = (\{q0, q1, q2\} \times \{r0, r1, r2\} \times \{s0, s1, s2\}, \{a,b\}, \delta, (q0, r0, s0), \{(q2, r2, s1)\})$$

where

$$\delta((q', q'', q'''), x) = (\delta_{ab}(q', x), \delta_{ba}(q'', x), \delta_{ends}(q''', x))$$

for each (q', q'', q''') in the Cartesian product of $\{q0, q1, q2\}$, $\{r0, r1, r2\}$, and $\{s0, s1, s2\}$, and each $x \in \{a,b\}$. By properties of Cartesian products, the DFA D has 27 states. The part of the state diagram that includes the start state of D and its outgoing edges is in the figure below. The destination of the outgoing edge from the state $(q1, r0, s1)$ on input a is $(q1, r0, s1)$, the destination of the outgoing edge from the state $(q1, r0, s1)$ on input b is $(q2, r1, s1)$, the destination of the outgoing edge from the state $(q0, r1, s2)$ on input a is $(q1, r2, s1)$, and the destination of the outgoing edge from the state $(q0, r1, s2)$ on input b is $(q0, r1, s2)$.



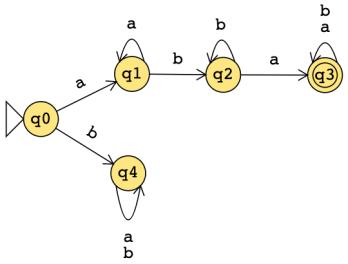
Part of state diagram for D that includes the start state and its outgoing edges

(b) Rewrite the language L in a simpler form and use this simpler form to design a DFA with at most 5 states that recognizes L. A complete solution will include the complete state diagram of this DFA and a justification for why the DFA recognizes L.

Solution: Since the alphabet has only two characters, strings starting with a and having both ab and ba as substrings are exactly those strings that start with some (positive) number of as, then have some (positive) number of bs, followed by at least one a and then any suffix of as and bs. Thus,

$$L = L(aa^*bb^*a(a \cup b)^*)$$

and we can design a DFA accordingly:



State diagram of DFA that we will show recognizes $L(aa^*bb^*a(a \cup b)^*)$

We need to explain how every string accepted by this DFA is in $L(aa^*bb^*a(a \cup b)^*)$, and that every string rejected by this DFA is not in $L(aa^*bb^*a(a \cup b)^*)$

- Let w be an arbitrary string accepted by the DFA above. This means that the computation of the DFA on w ends in the state $q3$, as it is the DFA's only accept state. The only path to $q3$ from the start state $q0$ must include (in turn, and potentially with repetitions) $q0, q1, q2, q3$. The only transition from $q0$ to $q1$ is labelled a so w must start with a. While the computation stays at $q1$, it must be reading as from w (because that's the only label of a self-loop at $q1$). The computation must transition to $q2$ at some point, which means that the prefix of as in w is followed by a b. A similar analysis shows that some number of bs followed by an a are the next characters

in w. There are self loops at $q3$ labelled by each of the alphabet characters so the suffix of w can be any string over this alphabet.

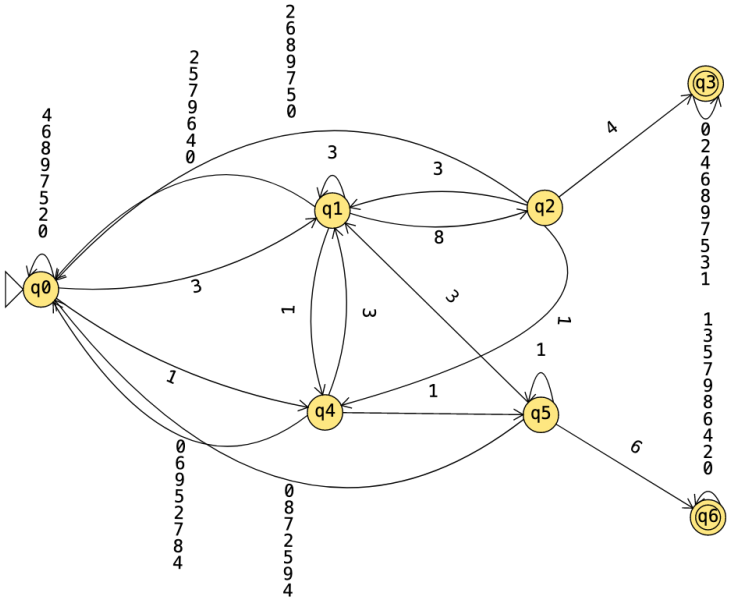
- Let u be an arbitrary string rejected by this DFA. This means that the computation of the DFA on u starts at $q0$ and ends in $q0$, $q1$, $q2$, or $q4$ (all the states that are not accepting). A similar analysis to the one we did above can be used to show that:
  - If the computation ends in $q0$, $u = \varepsilon \notin L(aa^*bb^*a(a \cup b)^*)$ because strings in the language described with the regular expression have at least one a following by at least one b, followed by at least one a, so have length at least 3.
  - If the computation ends in $q1$, $u = a^i$ for some $i > 0$, which means it has no bs so it is not an element of $L(aa^*bb^*a(a \cup b)^*)$.
  - If the computation ends in $q2$, $u = a^ib^j$ for some $i > 0, j > 0$ which means none of the bs in u are followed by an a, so $u \notin L(aa^*bb^*a(a \cup b)^*)$
  - Finally, if the computation ends in $q4$, then u starts with b, so $u \notin L(aa^*bb^*a(a \cup b)^*)$.

2. To safeguard the privacy or security of a network, some software filters the IP addresses that are allowed to send content to computers on the network. Each IP address can be broken into parts that represent the source host of incoming traffic, including geographic data. As a result, software needs to be designed to recognize whether certain substrings (representing permitted hosts) are present (if the hosts are permitted to send data) and whether others are absent (if those hosts are blocked from sending data).

   In this question, you'll design ways to detect these patterns in strings.

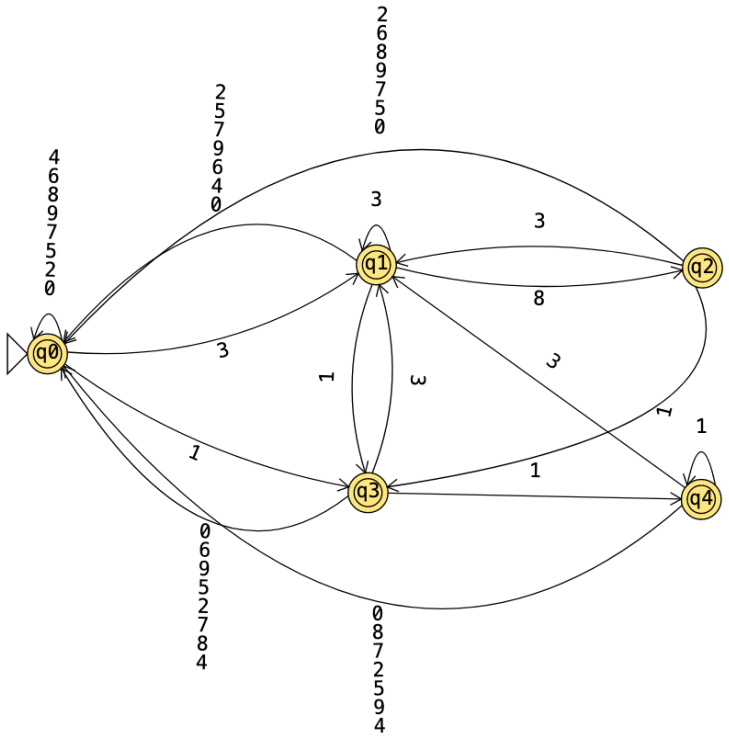
   (a) (*Graded for correctness*) Over the alphabet $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$ design a DFA that accepts each and only strings that have 384 or 116 as a substring. Your DFA should have **no more than** 8 **states**. A complete solution will include the state diagram of your DFA and a brief justification of your construction by explaining the role each state plays in the machine. *Note: you may include the formal definition of your DFA, but this is not required.*

   Solution: The state diagram for the DFA is below. $q0$ represents stages in the computation where we haven't made any progress on detecting the substring (the last character seen, if any, was not 3 or 1). The states $q1$, $q2$, and $q3$ are used to check for the consecutive characters in 384, while $q4$, $q5$, and $q6$ are used to check for the consecutive characters in 116. Transitions are added between them to adjust for seeing the starting character of one potential substring while in the process of checking for the other, e.g. if we see a 3 after seeing a 1, we move over to $q1$ to start checking for 384. The states $q3$ and $q6$ indicate we have detected at least one of the substrings and so all outgoing transitions are self loops to these accept states, since we don't need to check anything else about the string.



(b) (*Graded for correctness*) Now suppose the network administrators want to block traffic from IP addresses that have been associated with spammers. Over the alphabet $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$, design an NFA with at most 5 states that accepts each and only strings that do not have the substring 384 and do not have the substring 116. A complete solution will include the state diagram of your NFA and a brief justification of your construction by explaining the role each state plays in the machine.

Solution: The state diagram for the NFA is below. $q0$ represents stages in the computation where we haven't seen the beginnings of a problematic substring (the last character seen, if any, was not 3 or 1). As before, we have states tracking whether we start seeing either of 384 or 116. Notice that this design is very similar to the DFA from part (a), but we use nondeterminism to eliminate two states: the states where we would remember that we saw either 384 or 116, by simply disallowing transitions from $q4$ on input 4 and transitions from $q3$ on input 6.



(c) (*Graded for fair effort completeness*[2]) Give a regular expression that describes the set of strings over the alphabet $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$ that have 384 as a substring and give a (different) regular expression that describes the set of strings over the alphabet $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$ that do not have 384 as a substring. Briefly justify why each of your regular expression works.

Solution: We use the shorthand $\Sigma = 0 \cup 1 \cup 2 \cup 3 \cup 4 \cup 5 \cup 6 \cup 7 \cup 8 \cup 9$ in the regular expressions below.

The regular expression that describes the set of strings over this alphabet that have 384 as a substring is

$$\Sigma^* 384 \Sigma^*$$

To avoid having 384 as a substring, we have to

- Avoid 3s all together,
- Make sure any and all 3s are not followed by 8s (either by having the 3 at the end or by having a non-8 symbol after the 3), or
- Make sure any and all 38s are not followed by 4s

The regular expression gets a little messy but there's a systematic way to compute it, using the technique we discussed in class: first write the DFA that recognizes this

[2]This means you will get full credit so long as your submission demonstrates honest effort to answer the question. You will not be penalized for incorrect answers. To demonstrate your honest effort in answering the question, we ask that you include your attempt to answer *each* part of the question. If you get stuck with your attempt, you can still demonstrate your effort by explaining where you got stuck and what you did to try to get unstuck.

language, then use GNFA to remove one state at a time and describe all paths from the start state to an accept state using regular expressions. JFLAP is very helpful here: use the "Convert" menu to step through the process of translating your NFA or DFA to an equivalent regular expression.

$$(\Sigma_3 \cup 3(3 \cup 83)^*\Sigma_3\Sigma_{34})^*(\varepsilon \cup 3(3 \cup 83)^*(\varepsilon \cup 8))$$

where we have

$$\Sigma_3 = 0 \cup 1 \cup 2 \cup 4 \cup 5 \cup 6 \cup 7 \cup 8 \cup 9$$

$$\Sigma_{34} = 0 \cup 1 \cup 2 \cup 5 \cup 6 \cup 7 \cup 8 \cup 9$$

3. In this question, you'll practice working with formal general constructions for DFAs and translating between state diagrams and formal definitions. Consider the following construction in the textbook for Chapter 1 Problem 34, which we include here for reference: "Let B and C be languages over $\Sigma = \{0, 1\}$. Define

$$B \overset{1}{\leftarrow} C = \{w \in B \mid \text{ for some } y \in C, \text{ strings } w \text{ and } y \text{ contain equal numbers of 1s } \}$$

The class of regular languages is shown to be closed under the $\overset{1}{\leftarrow}$ operation using the construction: Let $M_B = (Q_B, \Sigma, \delta_B, q_B, F_B)$ and $M_C = (Q_C, \Sigma, \delta_C, q_C, F_C)$ be DFAs recognizing the languages B and C, respectively. We will now construct NFA $M = (Q, \Sigma, \delta, q_0, F)$ that recognizes $B \overset{1}{\leftarrow} C$ as follows. To decide whether its input w is in $B \overset{1}{\leftarrow} C$, the machine M checks that $w \in B$, and in parallel nondeterministically guesses a string y that contains the same number of 1s as contained in w and checks that $y \in C$.

1. $Q = Q_B \times Q_C$

2. For $(q, r) \in Q$ and $a \in \Sigma_\varepsilon$, define

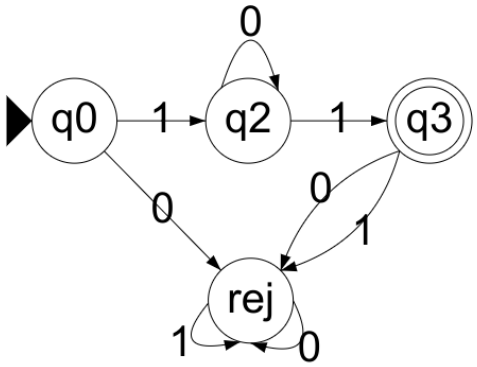
$$\delta(\ ((q, r), a)\ ) = \begin{cases} \{(\delta_B(q, 0), r)\} & \text{if } a = 0 \\ \{(\delta_B(q, 1), \delta_C(r, 1))\} & \text{if } a = 1 \\ \{(q, \delta_C(r, 0))\} & \text{if } a = \varepsilon \end{cases}$$

3. $q_0 = (q_B, q_C)$

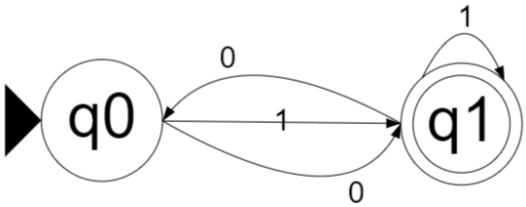
4. $F = F_B \times F_C$."

(a) (*Graded for correctness*) Illustrate this construction by defining specific example DFAs M_B and M_C and including their state diagrams in your submission. Choose M_B to have four states and M_C to have two states, and make sure that every state in each state diagram is reachable from the start state of that machine. Apply the construction above to create the NFA M and include its state diagram in your submission. *Note: you may include the formal definition of your DFAs and NFA, but this is not required. Hint: Confirm that you have specified every required piece of the state diagram for M. E.g., label the states consistently with the construction, indicate the start arrow, specify each accepting state, and include all required transitions.*

Consider M_B as follows



and M_C as follows



We apply the construction step by step

i. First, we create all states

$$Q = \{(q_0, q_0), (q_0, q_1), (rej, q_0), (rej, q_1), (q_2, q_0), (q_2, q_1), (q_3, q_0), (q_3, q_1)\}$$

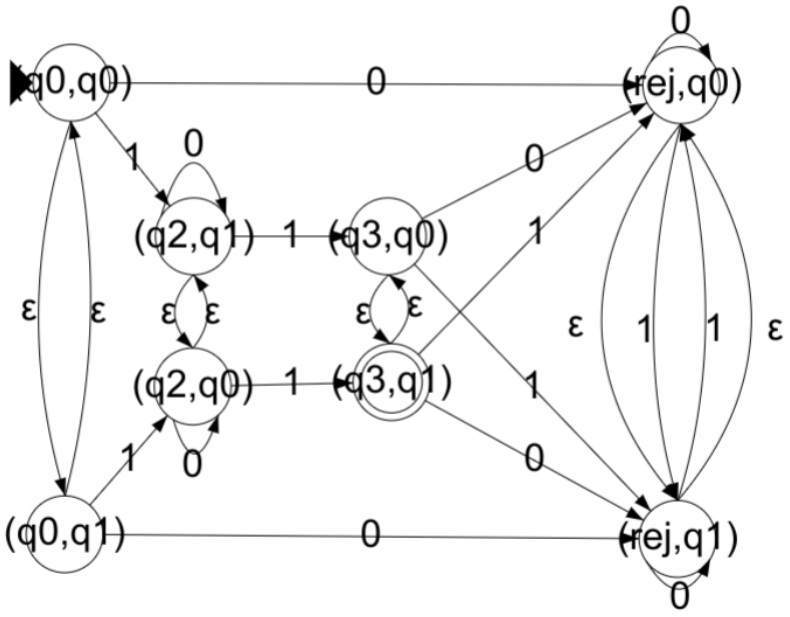
ii. Next, for the letter 0 in the alphabet, for each state $(q, r) \in Q$, send it to $(\delta_B(q, 0), r)$. Intuitively, this means we are running machine M_B as usual but we are not advancing M_C. The fact we are not advancing M_C is because C is only used as a way of counting the number of 1s in strings in B so the 0s in the string can be read at different points in the computations of M_B and M_C.

iii. Then, for the letter 1 in the alphabet, for each state $(q, r) \in Q$, send it to $(\delta_B(q, 1), \delta_C(r, 1))$. In other words, both M_B and M_C get to advance together (in parallel).

iv. We also add spontaneous moves that would correspond to advancing M_C's computation as though a 0 was read, but keeping M_B's computation at its current state.

v. Finally, set (q_0, q_0) as start and since there is only one accepting state in each of M_B and M_C, we set the set of accept states to be $\{(q3, q1)\}$

vi. After following the aforementioned steps, we get the following result, an NFA over $\{0, 1\}$



(b) (*Graded for fair effort completeness*) Describe the sets recognized by each of the machines you used in part (a): M_B, M_C, M. If possible, give an example of a string that is in B and in $B \overset{1}{\leftarrow} C$ and an example of a string that is in B and not in $B \overset{1}{\leftarrow} C$. If any of these examples do not exist, explain why not.

Solution: For the machines we chose: $L(M_B)$ is the set of strings over $\{0,1\}$ of length at least 3, where there are exactly two 1s and those 1s are at the beginning and the end of the string. The language of the machine that was constructed is identical to $L(M_B)$, because every string in B has the same number of 1s as some string in C: just by taking the witness string the string 11 (which is accepted by C, and has two 1s just like every string in B). Thus, a string that is in both B and in $B \overset{1}{\leftarrow} C$ is 10001 and there is no string that is in B and not in $B \overset{1}{\leftarrow} C$.

4. (*Graded for fair effort completeness*) In last week's homework, we saw the definitions of two functions on the set of languages over $\{0,1\}$: for L a set of strings over the alphabet $\{0,1\}$, we can define the following associated sets

$$LZ(L) = \{0^k w \mid w \in L, k \in \mathbb{Z}, k \geq 0\}$$

$$TZ(L) = \{w0^k \mid w \in L, k \in \mathbb{Z}, k \geq 0\}$$

In this question, we'll develop a general construction that will prove that if L is regular then so are $LZ(L)$ and $TZ(L)$.

Consider an arbitrary regular language L over the alphabet $\Sigma = \{0,1\}$, and we are given that $M = (Q, \Sigma, \delta, q_0, F)$ is a DFA over Σ with $L(M) = L$.

(a) Give the formal construction of an NFA M' with $L(M') = LZ(L)$. Briefly justify each parameter in the definition of M'.

Solution: The NFA $M' = (Q', \Sigma, \delta', q', F')$ where

- $Q' = Q \cup \{q_{new}\}$ where $q_{new} \notin Q$ is a fresh state that is going to serve as our start state and where we will read the leading zeros of the strings that will be accepted by the machine.
- $\Sigma = \{0, 1\}$ is unchanged
- $\delta' : Q' \times \Sigma_\varepsilon \to \mathcal{P}(Q')$ is given by

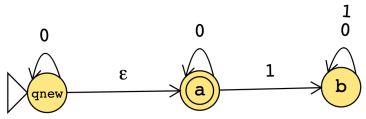
$$\delta'((q,x)) = \begin{cases} \{q_{new}\} & \text{if } q = q_{new}, x = 0 \\ \emptyset & \text{if } q = q_{new}, x = 1 \\ \{q_0\} & \text{if } q = q_{new}, x = \varepsilon \\ \{\delta((q,x))\} & \text{if } q \in Q, x \in \Sigma \\ \emptyset & \text{if } q \in Q, x = \varepsilon \end{cases}$$

  In words, the transition function has a self-loop at the new start state to let us read as many 0s as we like before spontaneously switching to simulating the given DFA M. Notice that the transition function of the NFA M' has different input and output types than the transition function of the DFA M.
- $q' = q_{new}$ The start state of the new machine M' is q_{new}
- $F' = F$ The set of accept states of the new machine M' is the same as the set of accept states of M.

Notice that this construction mirrors what we did earlier to prove that the class of regular languages is closed under concatenation, because $LZ(L) = L(0^*) \circ L$.

(b) Apply your construction from part (a) when $L_{test} = L(M_{test})$, where the state diagram M_{test} is below. Submit the state diagram of the NFA that results. If possible, give an example of a string that is in $LZ(L_{test})$ and one that is not; if either of these examples do not exist, explain why not.

Solution: The state diagram resulting from applying the construction from part (a) is below



The language recognized by this diagram is $\{0\}^*$ so an example of a string that is in this set is 00 and one that is not is 1.

(c) Give the formal construction of an NFA N' with $L(N') = TZ(L)$. Briefly justify each parameter in the definition of N'.

Solution: The NFA $N' = (Q', \Sigma, \delta', q', F')$ where

- $Q' = Q \cup \{q_{acc}\}$ where $q_{acc} \notin Q$ is a fresh state that is going to serve as our accept state and where we will read the trailing zeros of the strings that will be accepted by the machine.
- Σ is unchanged

- $\delta' : Q' \times \Sigma_\varepsilon \to \mathcal{P}(Q')$ is given by

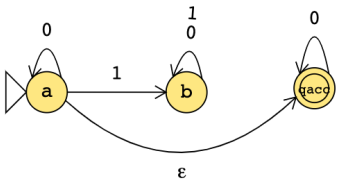
$$\delta'((q,x)) = \begin{cases} \{\delta((q,x))\} & \text{if } q \in Q, x \in \Sigma \\ \emptyset & \text{if } q \in Q \setminus F, x = \varepsilon \\ \{q_{acc}\} & \text{if } q \in F, x = \varepsilon \\ \{q_{acc}\} & \text{if } q = q_{acc}, x = 0 \\ \emptyset & \text{if } q = q_{acc}, x = 1 \\ \emptyset & \text{if } q = q_{acc}, x = \varepsilon \end{cases}$$

  In words, the transition function simulates the given DFA M, and at some point when it reaches an accept state, it can spontaneously switch to a mode of just reading trailing 0s . Notice that the transition function of the NFA N' has different input and output types than the transition function of the DFA M.
- $q' = q_0$ The start state of the new machine N' is the start state of the old machine M
- $F' = \{q_{acc}\}$ The set of accept states of the new machine M' is the singleton set that only has the new state in it.

Notice that this construction mirrors what we did earlier to prove that the class of regular languages is closed under concatenation, because $TZ(L) = L \circ L(0^*)$.

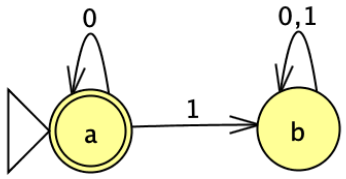
(d) Apply your construction from part (c) when $L_{test} = L(M_{test})$, where the state diagram M_{test} is below. Submit the state diagram of the NFA that results. If possible, give an example of a string that is in $TZ(L_{test})$ and one that is not; if either of these examples do not exist, explain why not.

Solution: The state diagram resulting from applying the construction from part (c) is below



The language recognized by this diagram is also $\{0\}^*$ so the same examples from part (b) work here too.

Caution: Pay attention to the types of the components, especially in the transition function. You are given a DFA and are building an NFA.



State diagram for DFA M_{test}