Problem Set 2: Finite Automata
CS254, Fall 2019, Anna Rafferty

Solution: Please do not post these solutions or share them with anyone outside this class. Thanks!

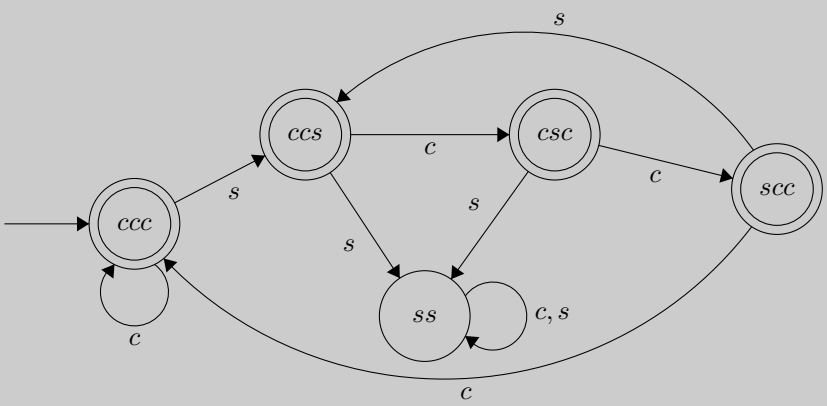
You are welcome to talk to others about the problems on this problem set, but you should **read and attempt each problem prior to discussing it with others.** The write up must be your own work. That means that **you should write your answers by yourself, without looking at notes from sessions with collaborators, and you must be able to explain any answer you write down.**

Due dates: Any 3 problems due on Monday, 23 September. The remaining 3 problems are due on Wednesday, 25 September. As for all homeworks, if a problem is due on a specific day it must be turned in by 9PM that day. See homework handout (a copy is posted on Moodle) for more specific homework policies. Remember to also fill out the collaboration form on Moodle; this form must be turned in for each problem set in order to get credit for your work.

1. Construct DFAs that recognize the following languages, and draw the diagram for each automaton (you do not need to provide the tuples).[1] Justify why your DFAs are correct. Your justifications need not be formal proofs, but should explain clearly and specifically why what you've drawn accepts **all** and **only** those strings in the given language.

   (a) Let $\Sigma = \{c, s\}$. Draw a DFA with no more than 5 states that recognizes the following language:
   $L = \{w \mid$ every substring of 3 symbols or less has at most 1 $s\}$
   For instance, *sccs* and *csccc* are in the language, but *cscs* is not.
   Updated 9/18 to clarify whether all strings with less than 3 symbols are in the language: they are not all in the language.

   Solution:

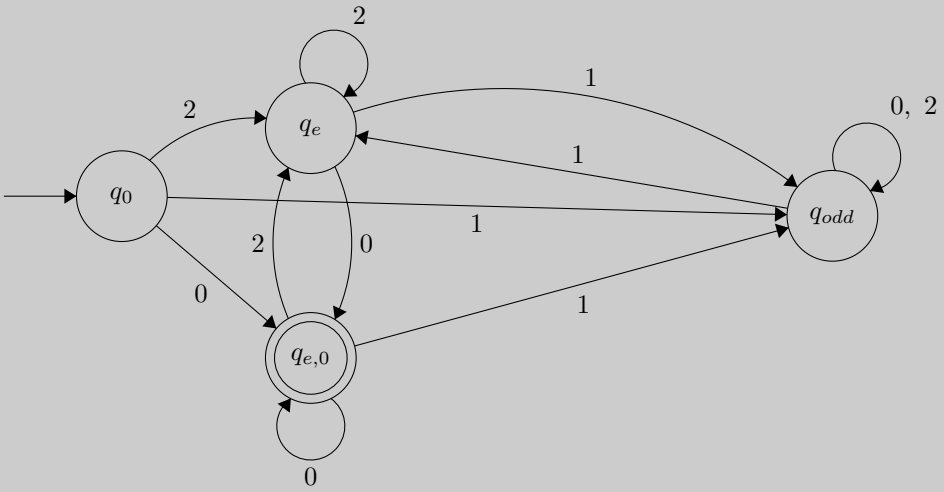
   The state labels reflect where in the last three characters an s has appeared (if at all). The far right means we have just seen an s (e.g., *ccs*), whereas the far left means we have seen the s followed by two more characters. If we have seen an s in the last two characters and we see another one, we move to the non-accepting *ss* state and remain there forever. Otherwise, we remember when we have seen an s using the state.

   (b) Let $\Sigma = \{0, 1, 2\}$, and assume that for string w, $v(w)$ is the number represented by the base-3 string w. For example, if $w = 02101$, then $v(w) = 64$. Draw a DFA that recognizes $L = \{w \mid v(w)$ is divisible by $6, w \neq \epsilon\}$.
   Hint: In a base-3 string, the place values are powers of 3 - $3^0, 3^1, 3^2, \ldots$ Each power of three is odd. What's true about the sum of an odd number of odd numbers? What about the sum of an even number of odd numbers? Use this to help you determine if a string represents a number that's divisible by 6. Make sure you briefly explain why these properties hold and how they relate to your solution if using them in your justification for why your DFA accepts language L.

   Solution: We first determine that multiples of 6 in base 3 are exactly those numbers divisible by 3 and by 2, and then we determine what properties multiples and multiples of two have in base 3. We note that

[1] Some students in past classes have liked the website http://madebyevan.com/fsm/ for creating diagrams of automata, although you are also welcome to neatly draw your diagram by hand, scan it, and embed the diagram in your PDF submission.

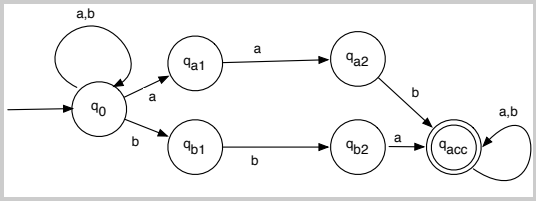
any 3^i for $i \geq 0$ is odd, as 3^i does not include a factor of 2. To be divisible by 2, the number must be even. The sum of an even number of odd numbers is even, since we can write each odd number as $k+1$ for some even k and then we have m 1s, where m is the number of odd numbers. Since m is even and each k is even, the total sum must be even. Similarly, the sum of an odd number of odd numbers is always odd (and thus not divisible by two): there is an extra 1, and 1 plus an even number is an odd number. Thus, to make a detector for multiples of 2 in base 3, we need only make a detector that sums up the digits of the number and if the sum is even, accepts. To modify this machine to accept numbers that are divisible by both 3 and 2, we recognize that a number in base 3 is divisible by 3 if the digit in the ones place is 0. Thus, we also make our machine only accept if the number ends in 0. In the machine below, q_0 is entered only for the string ϵ, which we should note accept, q_{odd} represents strings where the sum of the digits is odd, q_e represents strings where the sum of the digits is even but the last digit was not 0 (either 1 or 2), and $q_{e,0}$, the accept state, represents cases where the sum of the digits was even and the last digit was 0 - i.e., if this is the end of the string, the strings represents a number divisible by 6 in base 3.



2. Construct NFAs that recognize the following languages, and draw the diagram for each automaton (you do not need to provide the tuples). Your justifications need not be formal proofs, but should explain why what you've drawn accepts all and only those strings in the given language.

   (a) Let $\Sigma = \{a, b\}$. Draw an NFA that recognizes $L = \{w \mid w \text{ contains } aab \text{ or } bba\}$. Justify why the NFA you have drawn accepts language L. *Note: When we use or, we mean its logical definition: $a \vee b$ is true if a, b, or both are true.*

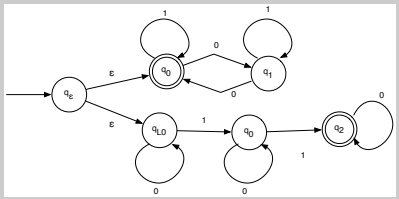
Solution:



The single accept state is reached after reading either aab or bba. This substring may be preceded by any number of a's or b's, as well as followed by any number of a's or b's. The machine nondeterministically guesses when to transition to reading the substring.

   (b) Sipser 1.7c.

Solution:



The machine nondeterministically guesses which of the two cases to pursue: the bottom path accepts if there are exactly two ones, and the top accepts if there are an even number of 0s (alternating between the top two states based on even versus odd zeros seen so far).

3. Let $A \subseteq \Sigma^*$ and let $x \in \Sigma^*$ be an arbitrary string. Write $x \circ A$ to denote the set $\{xy \mid y \in A\}$; that is, the set of all strings xy where $y \in A$. For example, if we let $\Sigma = \{c\}$ and let $ODD = \{c, ccc, ccccc, \ldots\}$, then if $w = c$, $w \circ ODD = \{cc, cccc, ccccc, \ldots\}$.

   Let Σ be an arbitrary alphabet. Show that if $A \subseteq \Sigma^*$ is any regular language and $x \in \Sigma^*$ is any string, then $x \circ A$ is also regular. Proceed by induction on x. (Hint: Start by thinking about what if $x = \epsilon$. Then consider the case $x = ax'$ where $a \in \Sigma$ and $x' \in \Sigma^*$.) Do not use closure properties of the regular languages; prove this result from scratch.

   You may find the following associative property of string concatenation useful, which you may use without proof: for any strings $x, y, z \in \Sigma^*$, $(xy)z = x(yz)$.

Solution: We will use structural induction on x to show that $x \circ A$ is regular if A is regular.

Base case $x = \epsilon$: If $x = \epsilon$, then $x \circ A = \{xy \mid y \in A\} = \{\epsilon y \mid y \in A\} = \{y \mid y \in A$ by our definition of string concatenation. This language is regular as it is language A, which we are given as regular.

Inductive case: We assume that the $x' \circ A$ is regular and seek to show that for $x = ax'$ $(a \in \Sigma)$, $x \circ A$ is regular. By our definition of $\circ$ and associativity, we have:

$$\begin{aligned}
x \circ A &= \{(ax')y \mid y \in A\} && \text{definition of concatenation} \\
&= \{a(x'y) \mid y \in A\} && \text{associativity} \\
&= \{az \mid z \in x' \circ A\} \\
&= a \circ (x' \circ A) && \text{definition of concatenation}
\end{aligned}$$

By the inductive hypothesis, $x' \circ A$ is regular, so all that remains is to show that $a \circ (x' \circ A)$ is regular.

Let $M = (Q, \Sigma, \delta, q_0, F)$ be the DFA that accepts $x' \circ A$. We construct M' which accepts $a \circ (x' \circ A)$ as follows. Let $M' = (Q \cup \{r, t\}, \Sigma, \delta', t, F)$. δ' is identical to δ except modified so that $\delta(t, a) = q_0$ and for any $b \in \Sigma$ where $b \neq a$, $\delta(t, b) = r$. For all $b \in \Sigma$, $\delta(r, b) = r$. That is, the new start state t transitions to the start state for M on symbol a, and to a reject state on any other symbol.

We can show this machine M' accepts exactly $a \circ (x' \circ A)$, letting $w = w_0 \ldots w_{n-1}$ (i.e., each w_i is the character in the ith position of the string w):

$$\begin{aligned}
w \in L(M') &\iff \hat{\delta}(t, w) \in F && \text{definition of accept} \\
&\iff \hat{\delta}(\delta(t, w_0), w_1 \ldots w_{n-1}) && \text{definition of } \hat{\delta} \\
&= \begin{cases} \hat{\delta}(r, w_1 \ldots w_{n-1}) \text{if } w_0 \neq a, \\ \hat{\delta}(q_0, w_1 \ldots w_{n-1}) \quad \text{if } w_0 = a \end{cases}
\end{aligned}$$

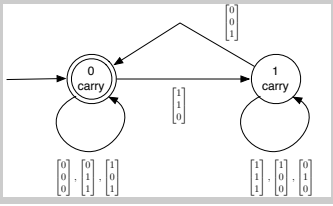
$\hat{\delta}(r, w_1 \ldots w_{n-1}) \notin F$ by our definition of r, and $\hat{\delta}(q_0, w_1 \ldots w_{n-1}) \in F$ iff $w_1 \ldots w_{n-1} \in (x' \circ A)$. Thus, this machine accepts exactly the language $a \circ (x' \circ A)$, meaning that $a \circ (x' \circ A)$ is regular.

4. Problem 1.32 in Sipser. (If you construct a machine for this problem, you should give a very clear justification for why this machine accepts the given language, although you do not need to use induction to prove your machine correct unless you would like to.)

Solution:

As suggested, we prove that B is regular by proving that $B^{\mathcal{R}}$ is regular and using the fact that $B^{\mathcal{R}}$ iff B is regular.

To show $B^{\mathcal{R}}$ we give an NFA M for recognizing $B^{\mathcal{R}}$ and show that the language it accepts is exactly $B^{\mathcal{R}}$:



(Note: Intuitively, this machine works by keeping track of whether the sum so far is correct as is or is holding a carry but is otherwise correct. The three characters that transition 0carry to 0carry are exactly those that sum up correctly in place, assuming there was no carryover from the previous place value. $[1,1,0]$ transitions to 1carry because we now have "carried" a one to the next place value. The characters that transition 1carry to 1carry mean that the sum is correct so far and there is still a carried digit. We transition from 1carry to 0carry on $[0,0,1]$ because we have now resolved the carried digit without creating a new carried digit. We accept on 0carry because this indicates that the values sum up correctly and there is no carried digit waiting to be resolved.)

We prove by induction on w that the language accepted by this NFA is exactly $B^{\mathcal{R}}$. We'll actually prove something a bit stronger: that if $\hat{\delta}(0carry, w)$ is equal to $\{0carry\}$, then the sum is correct; that if $\hat{\delta}(0carry, w)$ is equal to $\{1carry\}$ then the sum would be correct with an additional [0,0,1] at the end (i.e., that the sum is correct except there is an unresolved carry); and that if $\hat{\delta}(0carry, w) = \varnothing$, then it is impossible to add characters to make the sum correct. (We know these are the only three options because no state has two outgoing arrows with the same symbol and there are no ϵ-transitions, so there is only one sequence of states possible for reading any string.)

Our base case is that $w = \epsilon$. In this case, the sum is correct, and we see we end in 0carry.

For the inductive case, let $w = w'a$, where $a \in \Sigma$. We assume that the machine ends in 0carry if the sum for w' is correct and in 1carry if the sum for w' would be correct with an additional [0,0,1] at the end.

We consider each of these cases in turn. If $\hat{\delta}(0carry, w') = \{0carry\}$, then the sum is correct by our inductive hypothesis. Then if $a_{top} + a_{mid} = a_{bot}$, the sum is still correct. This occurs exactly in the states $[0,0,0], [1,0,1], [0,1,1]$. These are also exactly the states that transition to 0carry, so if a is one of these states, then the machine behaves as claims for $w = w'a$. If the $a = [1,1,0]$, then we see that 1+1=2 which is a carry in binary arithmetic. The machine transitions to $1carry$, and would be correct if the character [0,0,1] was appended. Finally, if $a \in \{[1,1,1], [0,1,0], [1,0,0], [0,0,1]\}$ then the digit for this place cannot be correct by the rules of binary arithmetic. Each of these values for a has no outgoing arrow from 0carry, corresponding to $\hat{\delta}(0carry, w) = \varnothing$. These are exactly the characters where it is impossible to add characters to make the sum correct.

Finally, we consider $w = w'a$ and assume $\hat{\delta}(0carry, w') = \{1carry\}$. Then, by our inductive hypothesis, w' would have a correct sum if $[0,0,1]$ were added. We consider the value of a. If $a = [0,0,1]$, then the sum is correct. We see in this case $\hat{\delta}(1carry, [0,0,1]) = \{0carry\}$, meaning $\hat{\delta}(0carry, w'a) = \{0carry\}$ as claimed. If $a \in \{[1,1,1], [0,1,0], [1,0,0]\}$, then the value for this place is correct because we have resolved the carry, and we have created a new carry meaning that we would have a correct sum if $[0,0,1]$ were added. With each of these three values, we have $\hat{\delta}(0carry, w'a) = \{1carry\}$, matching the claim above. Finally, with $a \in \{[0,0,0], [1,0,1], [0,1,1], [1,1,0]\}$, the the digit for this place cannot be correct by the rules of binary arithmetic. As in the case above each of these values of a as no outgoing arrow from 1carry, corresponding to $\hat{\delta}(1carry, w) = \varnothing$. These are exactly the characters where it is impossible to add characters to make the sum correct.

Bringing these cases back together, we see that $\hat{\delta}(0carry, w) = \{0carry\}$ exactly in the case where the sum is correct (assuming we are reading the string in reverse order). Thus, this machine accepts $B^{\mathcal{R}}$, so $B^{\mathcal{R}}$ is regular. Since the regular languages are closed under reverse, the language B is also regular.

Note for grading: For this question, a proof by structural induction is not required, but a good explanation is.

5. One of the problems in your textbook describes the quotient operator: $A/B = \{w \mid wx \in A \text{ for some } x \in B\}$. This problem has you explore what the quotient operator means. You need only provide a proof for the final part of this problem.

   (a) If $A = \Sigma^*$, what is A/B? *(Hint: There are two possibilities, depending on B.)*

   Solution: If $B = \varnothing$, $A/B = \varnothing$. Otherwise, $A/B = \Sigma^*$

   (b) Find an example of two languages A and B, each of which contain an infinite number of strings, such that $(A/B)B = A$. Additionally, find an example of two languages A and B, each of which contain an infinite number of strings, such that $(A/B)B \neq A$.

   Solution: Let $A = \{w \mid w = 10^k \text{ for } k \geq 0\}$, and let $B = \{w \mid w = 0^k \text{ for } k \geq 0\}$. Then $A/B = \{1, 10, 100, \ldots\} = A$. If we then concatenate B, we add zero or more 0s, resulting in $(A/B)B = \{1, 10, 100, \ldots\} = A$.
   Let $A = \{w \mid w = 1^k \text{ for } k \geq 1\}$, and let $B = \{w \mid w = 0^k \text{ for } k \geq 1\}$. I.e., A has strings of length at least 1 that only contain 1s, and B has strings of length at least 1 that only contain 0s. Then there does not exist a w such that $wx \in A$ for some $x \in B$, since that would imply that there is some string in A that contains 0s. Thus, $(A/B) = \varnothing$. We know that $\varnothing B = \varnothing$: concatenating any set with $\varnothing$ produces $\varnothing$. Thus, $(A/B)B = \varnothing \neq A$.

   (c) Textbook problem (1.45): Let $A/B = \{w \mid wx \in A \text{ for some } x \in B\}$. Show that if A is regular and B is any language then A/B is regular. Hint: Your answer should consist of defining an NFA or DFA that accepts A/B and proving that it accepts all strings in A/B and rejects all strings not in A/B. You need not draw the DFA but likely will define each of the elements in the tuple for the NFA/DFA. Note that this definition may be based on another DFA/NFA.

   Solution: Since A is regular, there exists a DFA $D = \langle Q, \Sigma, \delta, q_0, F\rangle$ that accepts A. We now construct a machine $M = \langle Q', \Sigma', \delta', q_0', F'\rangle$ that will accept A/B. We let $Q = Q'$, $\Sigma = \Sigma'$, $\delta = \delta'$, and $q_0 = q_0'$. We let $F' = \{q \in Q \mid \hat{\delta}(q, x) \in F \text{ for some } x \in B\}$. We claim that this machine accepts A/B.
   First, we show that if string $w \in A/B$, then M accepts w. Because $w \in A/B$, we know there exists an $x \in B$ such that $wx \in A$ (by the definition of A/B). Since $wx \in A$, we know there exists a sequence of states $r_0, \ldots, r_{|wx|}$ where $r_0 = q_0$ and for $i \geq 0$, $\delta(r_i, (wx)_i) = r_{i+1}$. Here, $(wx)_i$ is the ith character of wx. Then $r_{|w|} = \hat{\delta}(q_0, w)$ by the definition of $\hat{\delta}$. This state is in F' by our definition of F', so the machine accepts w.
   Next we show that if $w \notin A/B$, then M does not accept w. We use a proof by contradiction. Assume there exists some $w \notin A/B$ that is accepted by M. Then there must exist a sequence of states $r_0, \ldots, r_{|w|}$ such that $r_0 = q_0$, for $i \geq 0$, $\delta(r_i, w_i) = r_{i+1}$, and $r_{|w|} \in F'$. By our construction of F', though, if $r_{|w|} \in F'$, then there exists an $x \in B$ such that $\hat{\delta}(r_{|w|}, x) \in F$. That means that $wx \in A$. This implies that $w \in A/B$, a contradiction. Thus, no such w can exist that is not in A/B and is accepted by M.

6. Write a Python program that simulates a DFA. To do this, we need to agree on how to represent the DFA. Let's assume it is a tuple in Python with the following elements (in order):

   - Q: A tuple of arbitrary Python objects. These objects must be hashable (e.g., Strings and tuples are both hashable; lists are not).
   - Σ: A tuple of characters.
   - δ: A dictionary in Python. Each key is a 2-tuple of the form (q, a) where $q \in Q$ and $a \in \Sigma$. Each value is an element of Q. If any tuple $(q, a) \in Q \times \Sigma$ is not a key in the dictionary, then it is understood that the DFA rejects when it must transition out of state q via symbol a.
   - q_0: A Python object, which must be an element in Q.
   - F: A tuple of Python objects, which must be a subset of Q.

   Here's an example DFA that recognizes the language $\{w \mid w \text{ contains an odd number of 1s}\}$:

```
delta = {("q0", "0"):"q0", ("q0", "1"):"q1",
    ("q1", "0"):"q1", ("q1", "1"):"q0"}
m = (("q0", "q1"), ("0", "1"), delta, "q0", ("q1",))
```

   Write a function `dfa` that takes two inputs, a DFA M and an input string w, and outputs True if M accepts and False if M rejects. You may assume that M is a well-formed DFA and that the alphabets of M and w

match. As in other classes, make sure to write your code using good style - descriptive variable names, comments, appropriate structure, etc. You may find it helpful to use your program to simulate the behavior of DFAs earlier in this problem set to better understand them or to test your solution.

To turn in this problem, please submit a file named `ps02-06.py` into your Hand-In directory.

Solution:

```
def dfa(M, w):
   """Runs the DFA M on the string w, returns true if M accepts, false otherwise.

   Keyword arguments:
   M: DFA is a five-dimensional tuple. Contains the following:
   Q: Tuple of arbitrary (but hashable) Python objects.
   alphabet: Tuple of characters.
   delta: Dictionary. Each key is a 2-tuple of the form (q;a), where q is in
   Q and a is in alphabet. Each value is in Q. If a key is missing, then the
   DFA rejects.
   start state: Python object. Must be one of the elements of Q
   F: Tuple of Python objects. Must be subset of Q, not necessarily in the same
   order.
   """
   curState = M[3] # start state is in third position in tuple
   for curSymbol in w:
       curTransition = (curState,curSymbol);
       if curTransition in M[2]:
           curState = M[2][curTransition];
       else:
           return False;
   return curState in M[4]
```

7. **Optional suggested problems:** I will sometimes provide optional suggested questions for problems that are in the book where you can check your answers and that may be easier than some of the regular problem set problems; these are intended to help you ramp up to the problem set if you're finding it too difficult or just to get more familiar with the class concepts. In general, you may also find it useful to try the examples in the book, covering up their solutions and trying to create your own from scratch. You are also encouraged to complete any worksheets you didn't finish in class; these often have easier problems than the problem sets. If you're struggling with making any progress on a problem set, please come talk to me about how to get more out of class/readings; if you're only struggling on a few problems, that may just be a sign that some of them are tough! Taking a break, working with others, or coming to office hours may help. My hope is that some problems *are* tough for the majority of the class because working on hard problems is often the best way to learn. Besides the optional suggested problems, I generally do not indicate on problem sets which ones are harder/easier. This is because difficulty isn't uniform across people and because part of the experience of working on problems is figuring out how difficult it is and grappling with the fact that a problem which at first appears easy may in fact stump you for a while. For this problem set, you might try some of exercises 1.5-1.10 in Sipser to get more familiar with NFAs/DFAs if you're feeling unsure.