1. **(20 points):** Kozen, Homework 4, question 1 – Show that the following sets are not regular.

   (a) $B = \{a^n b^m | n = 2m\}$

   Solution:

   For the sake of contradiction, assume that B is regular. Let k be the pumping lemma constant for B. Consider the string $s = a^{2k}b^k \in B$. Let $x = \epsilon$, $y = a^k$, and $z = a^k b^k$. Clearly $s = xyz$. By the pumping lemma, there exist strings u, v, and w, such that $y = uvw$ and $|v| > 0$, and $xuv^iwz \in B$ for any $i \geq 0$. Let $j = |v| > 0$. Then, $uv^iw = a^{k+(i-1)j}$, and $xuv^iwz = a^{2k+(i-1)j}b^k$. By the definition of B, $xuv^iwz \in B$ iff $2k + (i-1)j = 2k$, which means $i = 1$ (because $j \neq 0$). In particular, if $i = 0$, then $xuv^0wz = a^{2k-j}b^k \notin B$. Thus, B does not satisfy the pumping lemma, and we conclude that B is not regular.

   (b) $B = \{x \in \{a, b, c\}^* | x = \mathsf{rev}(x)\}$

   Solution:

   (Note that the solution to part (a) provides a template to cut-and-paste and then change the definitions of the various x, y, z, u, v, and w strings to get the desired counter-example.)

   For the sake of contradiction, assume that B is regular. Let k be the pumping lemma constant for B. Consider the string $s = a^k bba^k \in B$. Let $x = \epsilon$, $y = a^k$, and $z = bba^k$. Clearly $s = xyz$. By the pumping lemma, there exist strings u, v, and w, such that $y = uvw$ and $|v| > 0$, and $xuv^iwz \in B$ for any $i \geq 0$. Let $j = |v| > 0$. Then, $uv^iw = a^{k+(i-1)j}$, and $xuv^iwz = a^{k+(i-1)j}bba^k$. By the definition of B, $xuv^iwz \in B$ iff $k + (i-1)j = k$, which means $i = 1$ (because $j \neq 0$). In particular, if $i = 0$, then $xuv^0wz = a^{k-j}bba^k \notin B$. Thus, B does not satisfy the pumping lemma, and we conclude that B is not regular.

   (c) $B = \{x \in \{a, b, c\}^* | |x| \text{ is a perfect square}\}$

   Solution:

   For the sake of contradiction, assume that B is regular. Let k be the pumping lemma constant for B. Consider the string $s = a^{(k+1)^2} \in B$. Let $x = \epsilon$, $y = a^k$, and $z = a^{k^2+k+1}$. Clearly $s = xyz$. By the pumping lemma, there exist strings u, v, and w, such that $y = uvw$ and $0 < |v| \leq k$, and $xuv^iwz \in B$ for any $i \geq 0$. Let $j = |v|$, and consider the case with $i = 0$. We have, $|xuv^0wz| = k^2 + k + 1 > k^2$ (because $k \geq 0$). Furthermore $|xuv^0wz| = k^2 + k + 1 < k^2 + 2k + 1 = (k+1)^2$. We conclude: $k^2 < |xuv^0wz| < (k+1)^2$. Thus, $|xuv^0wz|$ is not a perfect square, which means that $xuv^0wz \notin B$. Thus, B does not satisfy the pumping lemma, and we conclude that B is not regular.

   (d) The set PAREN of balanced strings of parentheses.

   Solution:

   For the sake of contradiction, assume that B is regular. Let k be the pumping lemma constant for B. Consider the string $s = (^k)^k$. Let $x = \epsilon$, $y = (^k$, and $z =)^k$. Clearly $s = xyz$. By the pumping lemma, there exist strings u, v, and w, such that $y = uvw$ and $0 < |v| \leq k$, and $xuv^iwz \in B$ for any $i \geq 0$. Let $j = |v|$, and consider the case with $i = 0$. We have, $xuv^0wz = (^{k-j})^k$. Because $j > 0$, there are fewer left parentheses than right parentheses in this string. Thus, $xuv^0wz \notin B$. Thus, B does not satisfy the pumping lemma, and we conclude that B is not regular.

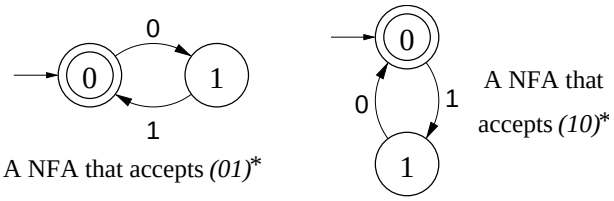

Figure 1: NFAs for $(01)^*$ and $(10)^*$

2. **(20 points):** Kozen, Homework 4, question 2 –
The operation of ***shuffle*** is important in the theory of concurrent systems. If $x, y \in \Sigma^*$, we write $x \| y$ for the set of all strings that can be obtained by shuffling strings x and y together like a deck of cards, for example:

$$ab \| cd \quad = \quad \{abcd, acbd, acdb, cabd, cadb, cdab\}$$

The set $x \| y$ can be defined formally by induction:

$$\begin{array}{rcl} \epsilon \| y & \stackrel{\text{def}}{=} & \{y\} \\ x \| \epsilon & \stackrel{\text{def}}{=} & \{x\} \\ xa \| yb & \stackrel{\text{def}}{=} & ((x \| yb) \cdot a) \cup ((xa \| y) \cdot b) \end{array}$$

The shuffle of two languages, A and B, denoted $A \| B$, is the set of all strings obtained by shuffling a string from A with a string from B:

$$A \| B \quad \stackrel{\text{def}}{=} \quad \bigcup_{\substack{x \in A \\ y \in B}} x \| y$$

For example,

$$\{ab\} \| \{cd, e\} \quad = \quad \{abe, aeb, eab, abcd, acbd, cabd, cadb, cdab\}$$

(a) What is $(01)^* \| (10)^*$?

Solution:

First, I constructed NFAs for the languages $(01)^*$ and $(10)^*$. Figure 1 shows these NFAs. Then, I constructed a NFA for the language $(01)^* \| (10)^*$. Here, the idea is that each symbol of the input string is consumed by one of the two component NFAs, but not both. So, I built something that resembles the product machine construction. Rows of states in my construction are copies of the NFA for $(01)^*$, and columns of states are copies of the NFA for $(10)^*$. Because each input symbol is consumed by exactly one of the original machines, moves in the joint machine are either horizontal (for a move by the $(01)^*$ machine), or vertical (for moves by the $(10)^*$ machine). There are no diagonal moves (unlike a true, product machine). Accepting states are states that are accepted by both of the original machines. Figure 2 shows this composite machine.

Given this NFA, it's straightforward to construct a regular expression for the language. Note that the state 11 could be eliminated, because the set of states reachable from it is identical to the set reachable from state 00, but I won't make this optimization. This makes my solution here correspond directly to my solution for part (b) below. Ignoring state 11, we see that the machine accepts strings that make any number of round trips to state 01 and back to 00 and any number of round trips to state 10 and back to 00. This is the language $(01 + 10)^*$.

A solution that presents a DFA, NFA, or RE for the language is acceptable. It's not necessary to present both a NFA and a RE like I did.

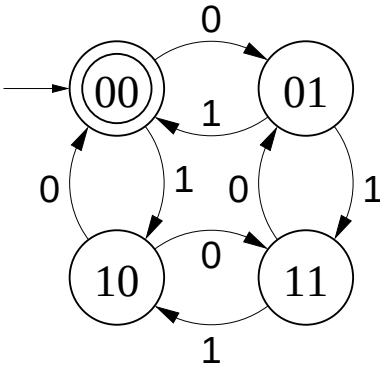

Figure 2: A NFA for $(01)^* \| (10)^*$

(b) Show that if A and B are regular sets, then so is $A \| B$.

Solution:

My solution is a generalization of that which I gave for part (a). Let $M_A = (Q_A, \Sigma, \Delta_A, Q_{0,A}, F_A)$ be a NFA that accepts A, and $M_B = (Q_B, \Sigma, \Delta_B, Q_{0,B}, F_B)$ be a NFA that accepts B. Let $M_{A\|B} = (Q_{A\|B}, \Sigma, \Delta_{A\|B}, Q_{0,A\|B}, F_{A\|B})$ be the NFA with

$$\begin{aligned} Q_{A\|B} &= Q_A \times Q_B \\ Q_{0,A\|B} &= (Q_{0,A}, Q_{0,B}) \\ F_{A\|B} &= F_A \times F_B \end{aligned}$$

I'll now describe how to construct $\Delta_{A\|B}$. Let $p_j \in \Delta_A(p_i, c)$. Then, for every $q_k \in Q_B$, we include (p_j, q_k) in $\Delta_{A\|B}((p_i, q_k), c)$. This says that the $M_{A\|B}$ can make a move corresponding to the move for M_A, and the state for the M_B machine remains unchanged. Likewise, if $q_j \in \Delta_B(q_i, c)$, then for every $p_k \in Q_B$, we include (p_k, q_j) in $\Delta_{A\|B}((p_i, q_k), c)$. Putting all of this together, we get

$$\Delta_{A\|B}((p, q), c) = (\Delta_A(p, c) \times \{q\}) \cup (\{p\} \times \Delta_B(q, c))$$

$\Delta_{A\|B}((p_i, q_k), c)$.

This completes the construction of $M_{A\|B}$. If $M_{A\|B}$ accepts w, we can "unshuffle" w into strings x and y where x is the sequence of symbols in w where $M_{A\|B}$ makes a move corresponding to M_A, and y is the sequence of symbols corresponding to moves of M_B. Thus, w is a shuffle of x and y where $x \in A$ and $y \in B$. A similar argument shows that for any $x \in A$ and $y \in B$, $M_{A\|B}$ accepts any shuffle of x and y. Thus, $L(M_{A\|B}) = A \| B$. $M_{A\|B}$ is a NFA. Therefore, $L(M_{A\|B})$ is regular. Thus, $A \| B$ is regular.

Remark: Kozen said that the shuffle operation is important in the theory of concurrent systems. For example, think of some protocol for communicating in a distributed system. We can think of A and B as describing the protocols that some process has with two, independent, remote processes. The language $A \| B$ describes all possible ways events can arrive from these two remote processes. We can then use this to show the process that receives these events behaves properly (e.g. doesn't deadlock, guarantees mutual exclusion if needed, always responds, etc.).

3. **(15 points):** Kozen Homework 3, question 3.
For any set of strings A, define the set

$$\mathsf{MiddleThirds}\ A \quad = \quad \{y | \exists x, z.\ (|x| = |y| = |z|) \wedge xyz \in A\}$$

Show that if A is regular, than so is $\mathsf{MiddleThirds}\ A$.

Solution:

My approach is to construct a NFA that accepts $\mathsf{MiddleThirds}\ A$. The challenge is to figure out if strings x and z exist without actually figuring out what the strings are. In particular, we can let $M_A = (Q, \Sigma, \delta, q_0, F)$ be a DFA that accepts A. We can build some other DFA or NFA that has a number of states that depends on the number of states in M_A, but we can't have the number of states of our new machine depend on the length of y.

The key observation is that all we know about x is that it has the same length of y. So, the question becomes:

What states are reachable from q_0 after reading $|y|$ input symbols?

Likewise, once we've read y, z has $|y|$ more symbols. Thus, if we reach state q_i after processing y, we want to know

What states are reachable from q_i after reading $|y|$ input symbols?

I'll construct a machine that does this by building a machine that can move from state q_i to q_j on *any* input symbol iff M_A can move from q_i to q_j for some input symbol. I'll write Δ_* to denote this transition relation with:

$$\Delta_*(q, c) \quad = \quad \{p | \exists a \in \Sigma.\ p = \delta(q, a)\}$$

Let $Q = q_0, q_2, \ldots q_{n-1}$. I'll now define machines $R_0, \ldots R_{n-1}$ such that $q_i \xrightarrow[M_i]{y} q_j$ iff there is some string x with $|x| = |y|$ such that $q_i \xrightarrow[M_A]{x} q_j$ Here's the definition:

$$R_i \quad = \quad (Q, \Sigma, \Delta_*, \{q_i\}, \emptyset)$$

I set the accepting set to $\emptyset$ because at this point we're just handling reachability. I'll figure out the accepting set that we really want shortly.

Machine R_0 will handle simulating the processing of x, and all of the R_i machines together will simulate processing z. I now need to process y. We don't know at the beginning of y what state of M_A to start in; so, we build n more machines to consider the M possibilities:

$$M_i \quad = \quad (Q, \Sigma, \delta, \{q_i\}, \emptyset)$$

We're almost there. String y is in $\mathsf{MiddleThirds}\ A$ iff there are states q_i and q_j and strings x and z with $|x| = |y| = |z|$ such that $\hat{\delta}(q_0, x) = q_i$, $\hat{\delta}(q_i, y) = q_j$, and $\hat{\delta}(q_j, z) \in F_A$. We can test this with a machine that is the cross-product of the R_i and M_i machines:

$$M_{\frac{1}{3}} \quad = \quad (Q_{\frac{1}{3}}, \Sigma, \Delta_{\frac{1}{3}}, Q_{0,\frac{1}{3}}, F_{\frac{1}{3}})$$

The state space for $M_{\frac{1}{3}}$ is $Q_{\frac{1}{3}} = Q^{2n}$. The transition relation is

$$\begin{array}{l} \Delta_{\frac{1}{3}}((r_0, r_1, \ldots r_{n-1}, s_0, s_1, \ldots s_{n-1}), c) \\ \quad = \quad \{(u_0, u_1, \ldots u_{n-1}, v_0, v_1, \ldots v_{n-1}) | \\ \qquad\qquad \forall i \in \{0 \ldots n-1\}.\ u_i \in \Delta_*(r_i, c) \\ \qquad\quad \wedge \quad \forall i \in \{0 \ldots n-1\}.\ v_i \in \delta(s_i, c) \\ \qquad \} \end{array}$$

In other words, the first n components of the state keep track of the R_i machines, and the last n keep track of the M_i machines.

There is one initial state:

$$Q_{0,\frac{1}{3}} \quad = \quad \{(q_0, q_1, \ldots q_{n-1}, q_0, q_1, \ldots q_{n-1})\}$$

Going back to our observation that string y is in MiddleThirds A iff there are states q_i and q_j and strings x and z with $|x| = |y| = |z|$ such that $\hat{\delta}(q_0, x) = q_i$, $\hat{\delta}(q_i, y) = q_j$, and $\hat{\delta}(q_j, z) \in F_A$, we can construct $F_{\frac{1}{3}}$ to recognize this set. In particular, after reading y, machine R_0 must reach state q_i, machine M_i must reach q_j, and matching R_j must reach some state in F. As a mathematical formula:

$$\begin{array}{rcl} F_{\frac{1}{3}} & = & \{(r_0, r_1, \ldots r_{n-1}, s_0, s_1, \ldots s_{n-1}) | \\ & & \quad \exists i, j.\ (r_0 = q_i) \wedge (s_i = q_j) \wedge (r_j \in F) \end{array}$$

By construction, $L(M_{\frac{1}{3}}) =$ MiddleThirds A. $M_{\frac{1}{3}}$ is a NFA. Therefore, MiddleThirds A is regular.

Short Solution:

Let $M_A = (Q, \Sigma, \delta, q_0, F)$ be a DFA that accepts A. Assume that $Q = q_0, q_1, \ldots q_n$. Let,

$$\Delta_*(q, c) \quad = \quad \{p | \exists a \in \Sigma.\ p = \delta(q, a)\}$$

A machine with state transition relation Δ_* can make a move from state q_i to q_j on *any* input symbol iff M_A can move from q_i to q_j for some input symbol. Define

$$R_i \quad = \quad (Q, \Sigma, \Delta_*, \{q_i\}, \emptyset)$$

With any input string, machine R_i can reach after n input symbols any state that machine M_A can reach from q_i for some input string input string of length n. In a similar fashion, define,

$$M_i \quad = \quad (Q, \Sigma, \delta, q_i, \emptyset)$$

Machine M_0 is the same as M_A, and the others are like M_A except that they have different starting states. I'll now construct a NFA that accepts MiddleThirds$A$. Let $Q_{\frac{1}{3}} = Q^{2n}$, and let

$$Q_{0,\frac{1}{3}} \quad = \quad \{(q_0, q_1, \ldots q_{n-1}, q_0, q_1, \ldots q_{n-1})\}$$

The state transition relation for $M_{\frac{1}{3}}$ is

$$\begin{array}{l} \Delta_{\frac{1}{3}}((r_0, r_1, \ldots r_{n-1}, s_0, s_1, \ldots s_{n-1}), c) \\ \quad = \quad \{(u_0, u_1, \ldots u_{n-1}, v_0, v_1, \ldots v_{n-1}) | \\ \qquad\qquad \forall i \in \{0 \ldots n-1\}.\ u_i \in \Delta_*(r_i, c) \\ \qquad\quad \wedge \quad \forall i \in \{0 \ldots n-1\}.\ v_i \in \delta(s_i, c) \\ \qquad \} \end{array}$$

Finally, the final states are:

$$\begin{array}{rcl} F_{\frac{1}{3}} & = & \{(r_0, r_1, \ldots r_{n-1}, s_0, s_1, \ldots s_{n-1}) | \\ & & \quad \exists i, j.\ (r_0 = q_i) \wedge (s_i = q_j) \wedge (r_j \in F) \end{array}$$

Machine $M_{\frac{1}{3}} = (Q_{\frac{1}{3}}, \Sigma, \Delta_{\frac{1}{3}}, Q_{0,\frac{1}{3}}, F_{\frac{1}{3}})$ accepts a string, y iff there are strings x and z with $|x| = |y| = |z|$ and states $q_i, q_j \in Q$ such that M_A moves from q_0 to q_i when reading x, from q_i to q_j when reading y, and from q_j to some state in F when reading z. This is demonstrated because $r_0 = q_i$ (i.e. machine R_0 was able to reach q_i in $|y|$ moves); $s_i = q_j$ (i.e. a machine like M_A starting in state q_i moved to q_j by reading y); and $r_j \in F$ (i.e. machine R_j started in state q_j and reached a state in F after $|y|$ moves).

4. **(45 points):** Let $\Sigma = \{0,1\}^3$, i.e. the set of tuples consisting of three bits. For, $(a,b,c) \in \Sigma$, define

$$\begin{aligned} \mathsf{first}((a,b,c)) &= a \\ \mathsf{second}((a,b,c)) &= b \\ \mathsf{third}((a,b,c)) &= c \end{aligned}$$

We overload first, second, and third to strings as shown below:

$$\begin{aligned} \mathsf{first}(\epsilon) &= \epsilon \\ \mathsf{first}(x \cdot \mathsf{c}) &= \mathsf{first}(x) \cdot \mathsf{first}(\mathsf{c}) \\ \\ \mathsf{second}(\epsilon) &= \epsilon \\ \mathsf{second}(x \cdot \mathsf{c}) &= \mathsf{second}(x) \cdot \mathsf{second}(\mathsf{c}) \\ \\ \mathsf{third}(\epsilon) &= \epsilon \\ \mathsf{third}(x \cdot \mathsf{c}) &= \mathsf{third}(x) \cdot \mathsf{third}(\mathsf{c}) \end{aligned}$$

For example, if $s = (0,0,0)(0,0,1)(0,1,0)(0,1,1)(1,0,0)(1,0,1)$, then $\mathsf{first}(s) = 000011$, $\mathsf{second}(s) = 001100$, and $\mathsf{third}(s) = 010101$. For $s \in \{0,1\}^*$, let $\mathsf{binary}(s)$ denote the binary value of s when the most significant bit is the first symbol of the string:

$$\begin{aligned} \mathsf{binary}(\epsilon) &= 0 \\ \mathsf{binary}(\mathsf{0}) &= 0 \\ \mathsf{binary}(\mathsf{1}) &= 1 \\ \mathsf{binary}(x \cdot \mathsf{c}) &= 2 * \mathsf{binary}(x) + \mathsf{binary}(c) \end{aligned}$$

For each of the languages described below, state whether or not the language is regular. If the language is regular, then construct a DFA, NFA, or pattern for the language to justify your conclusion; you may also use any property of regular languages that we have proven in class or that is proven in the textbook. If the language is not regular, prove that it is not. Again, you may use any property of regular languages that has been proven in class or in the textbook.

(a) **(15 points):** binary addition

$$B = \{w |\ \mathsf{binary}(\mathsf{third}(w)) = \mathsf{binary}(\mathsf{first}(w)) + \mathsf{binary}(\mathsf{second}(w))\}$$

Solution:

The language B is regular. My construction uses the same ideas as the "multiply-by-three" FA from HW1. I'll describe it using an NFA so I don't have to clutter my solution with transitions to a "garbage" state.

$$\begin{aligned} M &= (Q, \Sigma, \Delta, Q_0, F) \\ Q &= \{q_0, q_1\} \\ \Sigma &= \{0,1\}^3 Q_0 = \{q_0\} F = \{q_0\} \\ &\Delta(q_0,(0,0,0)) = \{q_0\}, \quad \Delta(q_1,(0,0,1)) = \{q_0\}, \\ &\Delta(q_0,(0,1,1)) = \{q_0\}, \quad \Delta(q_1,(0,1,0)) = \{q_1\}, \\ &\Delta(q_0,(1,0,1)) = \{q_0\}, \quad \Delta(q_1,(1,0,0)) = \{q_1\}, \\ &\Delta(q_0,(1,1,0)) = \{q_1\}, \quad \Delta(q_1,(1,1,1)) = \{q_1\} \end{aligned}$$

State q_0 represents the situation where the carry from the previous bit is 0, and q_1 indicates that the carry from the previous bit is 1.

By construction $L(M) = \mathsf{rev}(B)$. Thus, $B = \mathsf{rev}(L(M))$. $L(M)$ is regular because it is recognized by a NFA. Thus, B is regular because it is the reverse of a regular language.

(b) **(15 points):** unary numbers

$$B = \{w|\ \exists m_1, m_2, m_3. \quad (\mathsf{first}(w) = 1^{m_1}0^{|w|-m_1}) \\ \wedge \quad (\mathsf{second}(w) = 1^{m_2}0^{|w|-m_2}) \\ \wedge \quad (\mathsf{third}(w) = 1^{m_3}0^{|w|-m_3})\}$$

Solution:

The language B is regular. The requirement, $\mathsf{first}(w) = 1^{m_1}0^{|w|-m_1}$ corresponds to the regular expression α_1 where:

$$\begin{aligned} \alpha_1 &= \alpha_{11}^*\alpha_{10}^* \\ \alpha_{11} &= (1,0,0) + (1,0,1) + (1,1,0) + (1,1,1) \\ \alpha_{10} &= (0,0,0) + (0,0,1) + (0,1,0) + (0,1,1) \end{aligned}$$

Regular expressions α_2, and α_3 corresponding respectively to $\mathsf{second}(w) = 1^{m_2}0^{|w|-m_2}$ and $\mathsf{third}(w) = 1^{m_3}0^{|w|-m_3}$ can be defined in the same manner. By construction, $B = \alpha_1 \cap \alpha_2 \cap \alpha_3$. Thus, B is regular.

(c) **(15 points):** unary addition

$$B = \{w|\ \exists m_1, m_2. \quad (\mathsf{first}(w) = 1^{m_1}0^{|w|-m_1}) \\ \wedge \quad (\mathsf{second}(w) = 1^{m_2}0^{|w|-m_2}) \\ \wedge \quad (\mathsf{third}(w) = 1^{m_1+m_2}0^{|w|-(m_1+m_2)})\}$$

Solution:

This time, B is not regular. Let $C = B \cap (1,1,1)^*(0,0,1)^*$. It is straightforward to show that $C = (1,1,1)^k(0,0,1)^k$ (strings in C correspond to the sum $k + k = 2k$). This is equivalent to the language a^kb^k which we have already seen is not regular (see Kozen, lecture 11). Thus, C is not regular. The regular languages are closed under intersection. Therefore B is not regular either.

Note: it's perfectly acceptable to show that B is not regular by using the pumping lemma. I used this simple homomorphism argument to illustrate that sometimes there are even simpler proofs available.

One more remark: this problem shows that whether or not $\{(x,y,z)|z = x + y\}$ is regular depends on the encoding of the numbers. In particular, binary addition is a regular language but unary addition is not. If instead of entering corresponding bits of the three numbers together, we first entered all of the bits of x, then all of the bits of y, and finally all of the bits of z, then the language isn't regular either.