HW2CSE105W24: Sample solutions

CSE105W24

February 9, 2024

In this assignment,

You will practice designing multiple representations of regular languages and working with general constructions of automata to demonstrate the richness of the class of regular languages. You will also distinguish between regular and nonregular languages using both closure arguments and the pumping lemma.

Resources: To review the topics for this assignment, see the class material from Weeks 2-4. We will post frequently asked questions and our answers to them in a pinned Piazza post.

Reading and extra practice problems: Sipser Chapter 1, Section 2.2. 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, 1.29, 1.30. Chapter 1 problems 1.49, 1.50, 1.51.

Assigned questions

1. **Number representations** (12 points): Integers can be represented using base b expansions, for a convenient choice of base b: for b an integer greater than 1 and n a positive integer, the **base b expansion of** n is defined to be

$$(a_{k-1} \cdots a_1 a_0)_b$$

where k is a positive integer, $a_0, a_1, \ldots, a_{k-1}$ are nonnegative integers less than b, $a_{k-1} \neq 0$, and

$$n = \sum_{i=0}^{k-1} a_i b^i$$

Notice: *The base b expansion of a positive integer n is a string over the alphabet $\{x \in \mathbb{Z} \mid 0 \leq x < b\}$ whose leftmost character is nonzero.*

An important property of base b expansions of integers is that, for each integer b greater than 1, each positive integer $n = (a_{k-1} \cdots a_1 a_0)_b$, and each nonnegative integer a less than b,

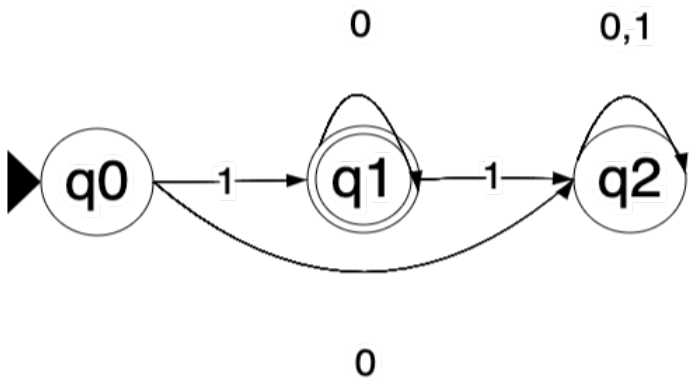
$$bn + a = (a_{k-1} \cdots a_1 a_0 a)_b$$

In other words, shifting the base b expansion to the left results in multiplying the integer value by the base. In this question we'll explore building deterministic finite automata that recognize languages that correspond to useful sets of integers.

(a) (*Graded for correctness*) Design a DFA that recognizes the set of binary (base 2) expansions of positive integers that are powers of 2. 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, as well as a brief justification about how the strings accepted and rejected by the machine connect to the specified language.

Hints: (1) A power of 2 is an integer x that can be written as 2^y for some nonnegative integer y, (2) the DFA should accept the strings 100, 10 and 100000 and should reject the strings 010, 1101, and ε (can you see why?).

Solution:



Justification: This DFA accepts strings over $\{0, 1\}$ that start with exactly a single 1 and end with 0's. This is how it does it:

- q_0 is the starting state, and its role is to not accept the empty string by not being on the accept states.
- q_1 is the state that makes sure the string starts with exactly one 1 and none comes after. We enter q_1 if the first character is a 1. If all we see afterward are zeros, we accept. Otherwise, we exit this only accept state.
- q_2 is the trap state. If the string starts with a 0 or has more than two 1s we get "stuck" in this non-accept state and reject.

Now, we need to show why strings starting with exactly a single 1 and ending with 0s are exactly the set of binary strings representing powers of 2.

- ($\subseteq$) Take any string s with a single 1 at the front and 0s afterward, its value is determined by that first 1 since all the zero coefficients do not contribute to the sum. Say $s = 1z_1z_2 \dots z_n$ where $n \geq 0$, the value represented by s is $1 \cdot 2^n$, which is a power of 2.
- ($\supseteq$) Take any binary string s representing a power of 2. The first bit contributes 2^x for some $x \leq y$ to the sum. By induction, one can prove that $\sum_{i=0}^{n-1} 2^i = 2^n - 1$ (left as an exercise). To put it another way, even if all digits of s were 1, the value it represents is $\sum_{i=0}^{x} 2^i = 2^{x+1} - 1$, which is one less than 2^{x+1}, the next power of 2 greater than 2^x.

This means that none of the bits after the first can be a 1 or else s will not represent a power of 2. This completes the proof.

(b) (*Graded for completeness*) Consider arbitrary positive integer m. Design a DFA that recognizes the set of binary (base 2) expansions of positive integers that are multiples of m. A complete solution will include the formal definition of your DFA (paramterized by m) and a brief justification of your construction by explaining the role each state plays in the machine, as well as a brief justification about how the strings accepted and rejected by the machine connect to the specified language.

Hints: (1) Consider having a state for each possible remainder upon division by m. (2) To determine transitions, notice that reading a new character will shift what we already read over by one slot.

Solution: $M = (\{q_i \mid 0 \leq i < m\} \cup \{q_{start}, q_{trap}\}, \{0,1\}, \delta, q_{start}, \{q_0\})$ where

$$\begin{aligned}
\delta(q_{start}, 0) &= q_{trap} \\
\delta(q_{start}, 1) &= q_1 \\
\delta(q_{trap}, 0) &= q_{trap} \\
\delta(q_{trap}, 1) &= q_{trap} \\
\delta(q_i, 0) &= q_{\ 2i \mod m} \forall i \\
\delta(q_i, 1) &= q_{\ (2i+1) \mod m} \forall i
\end{aligned}$$

Justification: Focusing on the last two lines, for each state q_i, i represents the number read so far in decimal modulo m. We need the machine to go back to q_0 to accept. Now, when we read a 0, the current number we have read doubles in size since we are in binary. On the other hand, when we read a 1, our number not only doubles in size but also increases by 1. Finally, because we are only interested in if the number is divisible by m, we mod by m to get the new state to transition to.

Now, since we need to only accept representations of positive integers, the empty string and strings starting with 0 should be rejected. To this end, we create a new start state make sure only strings starting with 1 can proceed. Otherwise, the machine will enter the trap state and reject eventually.

(c) (*Graded for correctness*) Choose a positive integer m_0 between 4 and 8 (inclusive) and draw the state diagram of a DFA recognizing the language over $\{0,1,2\}$

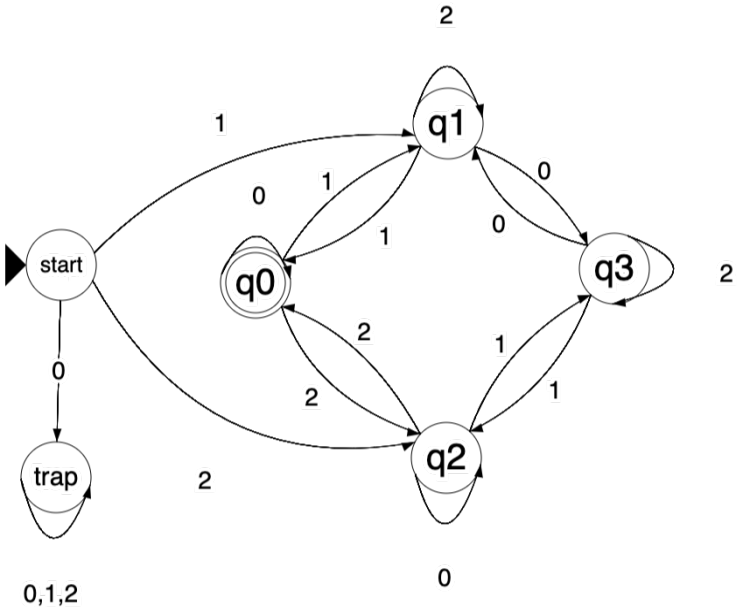
$$\{w \in \{0,1,2\}^* \mid w \text{ is a base 3 expansion of a positive integer that is a multiple of } m_0\}$$

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, as well as a brief justification about how the strings accepted and rejected by the machine connect to the specified language.

Bonus extension to think about (ungraded): Which other languages related to sets of integers can be proved to be regular using a similar strategy?

Solution: You only need to have one of the following.

$m_0 = 4$:

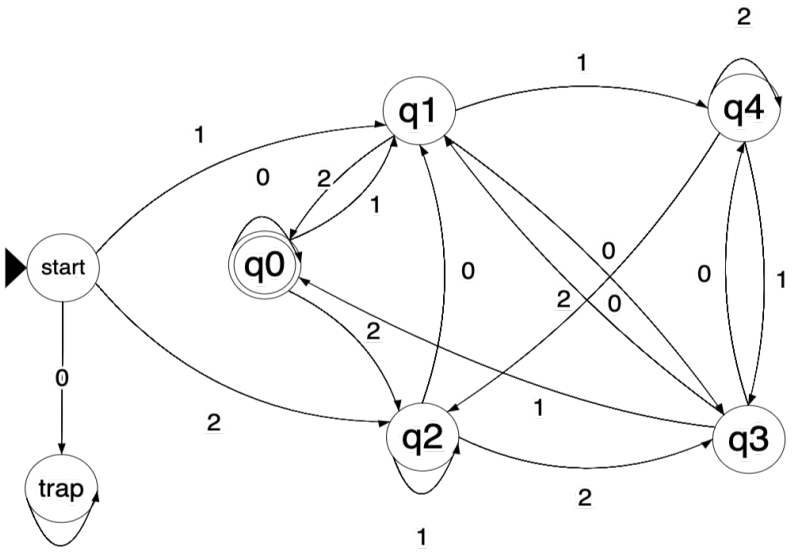


Justification:
The machine needs to accept strings that are base 3 expansions of a positive integer that is a multiple of m_0. The start state in combination with the trap state are included to reject any strings that start with a 0, as we require the first bit a_{k-1} to be nonzero. The role of the remaining states is to keep track of the current remainder when divided by m_0.

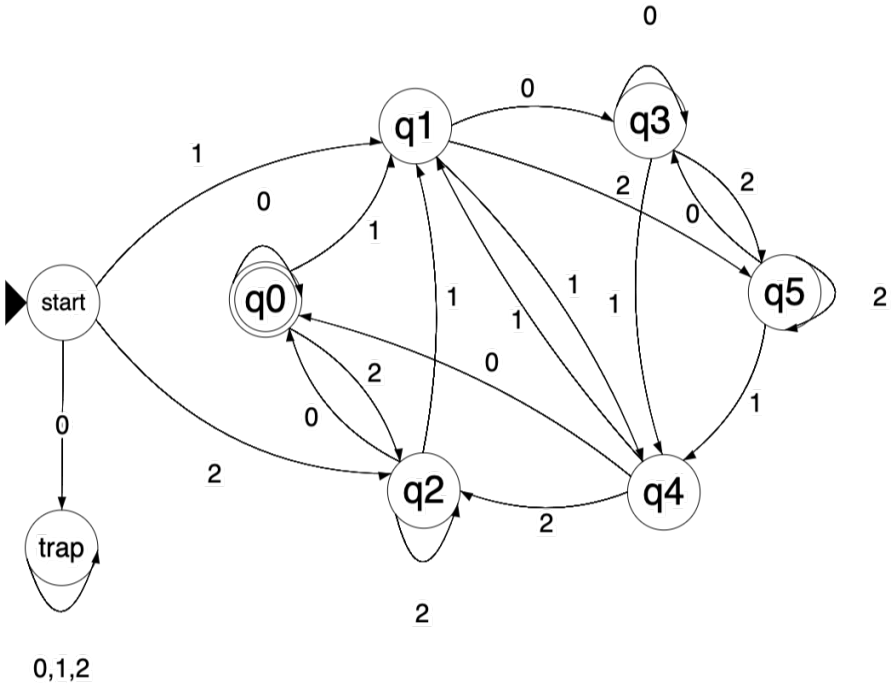
For example, in the above diagram, let $m_0 = 4$. Upon reading the first bit of the string 210, the current remainder will be updated to 2. When reading the next bit, we notice that the previous bit will be left shifted by 1, essentially multiplying the number by the base, 3. Thus, our current remainder will be updated to be $2 \cdot 3 + 1 \pmod 4 = 3$. Finally, after reading the final bit, we update the remainder to be $3 \cdot 3 + 0 \pmod 4 = 1$. Converting 210 in base 3 to 21 in decimal confirms that indeed, the remainder of 210 in base 3 divided by 4 is 1, thus we reject the string.

Below are the state diagrams for different values of m_0.

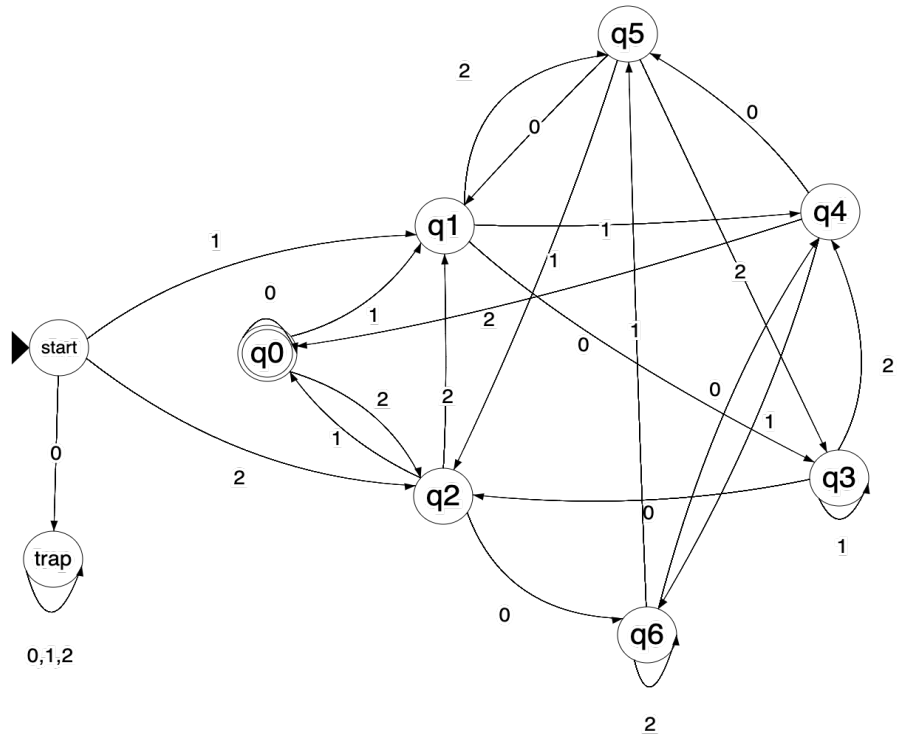
$\mathbf{m_0 = 5}$:



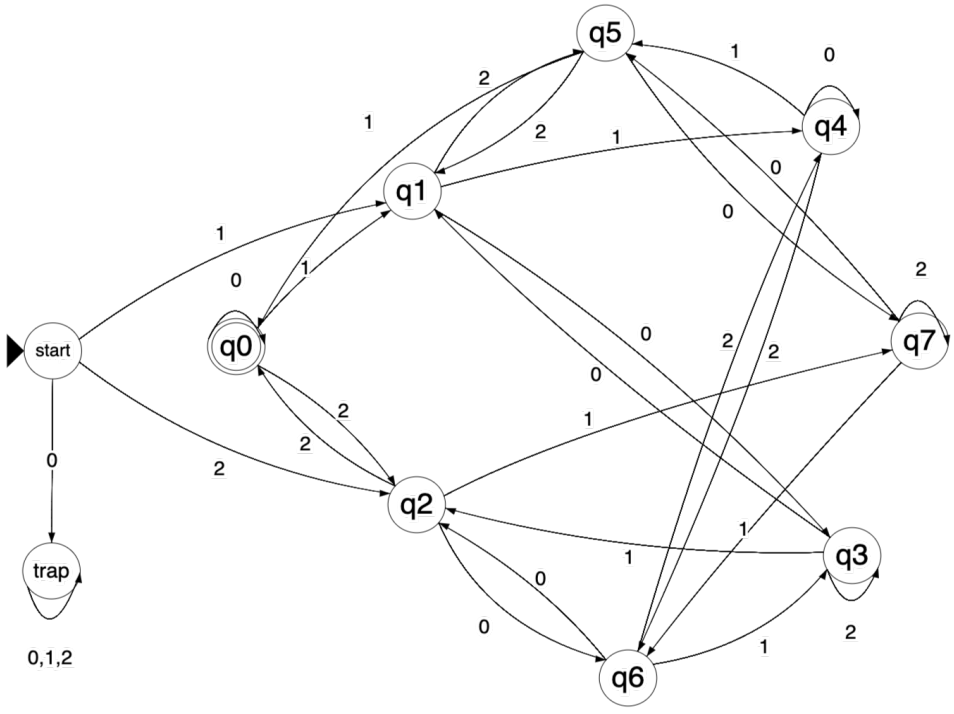
$\mathbf{m_0 = 6}$:



$\mathbf{m_0 = 7}$:



$\mathbf{m_0 = 8}$:



2. **Multiple representations** (10 points): For any language $L \subseteq \Sigma^*$, recall that we define its *complement* as

$$\overline{L} := \Sigma^* - L = \{w \in \Sigma^* \mid w \notin L\}$$

That is, the complement of L contains all and only those strings which are not in L. Our notation for regular expressions does not include the complement symbol. However, it turns out that the complement of a language described by a regular expression is guaranteed to also be describable by a (different) regular expression. For example, over the alphabet $\Sigma = \{0, 1\}$, the complement of the language described by the regular expression Σ^*0 is described by the regular expression $\varepsilon \cup \Sigma^*1$ because any string that does not end in 0 must either be the empty string or end in 1.

For each of the regular expressions R over the alphabet $\Sigma = \{a, b\}$ below, write the regular expression for $\overline{L(R)}$. Your regular expressions may use the symbols $\varnothing$, ε, a, b, and the following operations to combine them: union, concatenation, and Kleene star.

Briefly justify why your solution for each part works by giving plain English descriptions of the language described by the regular expression and of its complement and connecting them to the regular expression via relevant definitions. An English description that is more detailed than simply negating the description in the original language will likely be helpful in the justification.

Alternatively, you can justify your solution by first designing a DFA that recognizes $L(R)$, using the construction from class and the book to modify this DFA to get a new DFA that recognizes $\overline{L(R)}$, and then applying the constructions from class and the book to convert this new DFA to a regular expression.

For each part of the question, clearly state which approach you're taking and include enough intermediate steps to illustrate your work.

(a) (*Graded for correctness*) a^*b^*

Solution: $\Sigma^*b\Sigma^*a\Sigma^*$

First, we translate this regular expression to plain English. Looking at the structure of the regex, we note that it describes the set of strings whose a's are before the b's. Now, after negating this, we get the set of strings with at least one b before a. So any string inside the set must have at least one b somewhere and then at least one a after. But between these two b and a, we can have anything.

An equivalent regular expression is $\Sigma^*ba\Sigma^*$ since there are only two characters is Σ so if there's a b before an a then at some point there's a switch where consecutive characters go from b to a, namely the string will have ba as a substring.

(b) (*Graded for correctness*) $(a \cup b)ab^*$

Solution: $\varepsilon \cup a \cup b \cup (ab \cup bb)\Sigma^* \cup (aa \cup ba)\Sigma^*a\Sigma^*$

After distributing the union, we get an equivalent regex $(aa \cup ba)b^*$. This describes the set of all strings starting with aa or ba and end with only b's. So, negating this gives us (strings don't start with aa or ba) or (strings starting with aa or ba but has at least one a after). Strings that don't start with aa or ba must either be shorter than two characters or start

with ab or bb. Strings starting with aa or ba but ends in at least one a can be described by $(aa \cup ba)\Sigma^* a \Sigma^*$.

3. **Applying general constructions** (12 points): In this question, you'll practice working with formal general constructions for DFAs and translating between state diagrams and formal definitions. Consider the following general construction: Let $N_1 = (Q, \Sigma, \delta_1, q_1, F_1)$ be a NFA and assume that $q_0 \notin Q$. Define the new NFA $N_2 = (Q \cup \{q_0\}, \Sigma, \delta_2, q_0, \{q_1\})$ where

$$\delta_2 : Q \cup \{q_0\} \times \Sigma_\varepsilon \to \mathcal{P}(Q \cup \{q_0\})$$

is defined by

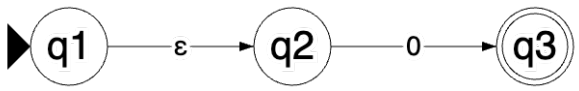
$$\delta_2(q, a) = \begin{cases} \{q' \in Q \mid q \in \delta_1(q', a)\} & \text{if } q \in Q,\ a \in \Sigma_\varepsilon \\ F_1 & \text{if } q = q_0,\ a = \varepsilon \\ \emptyset & \text{if } q = q_0,\ a \in \Sigma \end{cases}$$

(a) (*Graded for correctness*) Illustrate this construction by defining a specific example NFA N and applying the construction above to create a new NFA. Your example NFA should

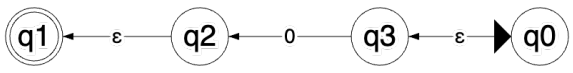
- Have exactly three states (all reachable from the start state),
- Have at least one spontaneous move (arrow labelled ε),
- Accept at least one string and reject at at least one string, and
- Not have any states labelled q_0.

Apply the construction above to create the new NFA. A complete submission will include the state diagram of your example NFA N and the state diagram of the NFA resulting from this construction.

Solution: We choose the alphabet $\{0\}$. Let the original 3-state NFA be as follows:

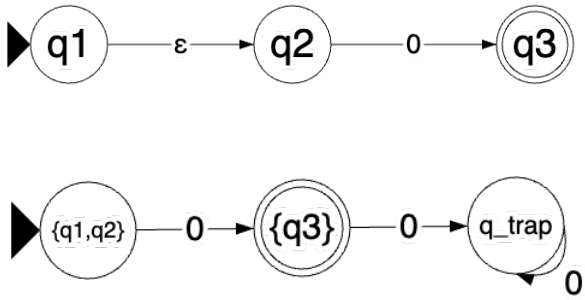


Now we apply the construction, creating q_0 and connecting it to all original final states, which is just q_3. Additionally, we reverse all the edges and make q_1 the accept state. This gives



(b) (*Graded for correctness*) Use Theorem 1.39 on page 55 of the book (see also page 7 in Week 3 notes) to construct a DFA equivalent to your example NFA N from part (a). A complete submission will include the state diagram of your example NFA N and the state diagram of the DFA resulting from this construction, with the correct state labels for this DFA. You may prune the DFA so that only the "macro-states" reachable from the start state are included.

Solution:



Where the label for $q_{trap} = \emptyset$. Note that when we begin in q_1, we also spontaneously transition to q_2, so our starting state is $\{q_1, q_2\}$. The other five macro-states labelled by elements of $\mathcal{P}(\{q1, q2, q3\})$ are not reachable from $\{q1, q2\}$ so we don't include them. The macro-state labelled $\{q3\}$ is the only accept state in the DFA because it's the only one where at least one of its elements is an accept state in the NFA.

(c) (*Graded for completeness*) Explain the relationship between N_1 and N_2 in the general construction. Give an example string that is accepted by your example NFA N and is rejected by the NFA that results from applying the general construction that illustrates this relationship, or explain why there is no such example string.

Solution: In general, this construction creates a machine whose language is the reverse of the original language. If a string s is in the original language, there is a path from the starting state to an accept state such that we read one character of s at a time. Now, if we feed s to the new machine, there will be a path from an old accept state to the old starting state reading s backward. The same argument applies if a string s is not in the original language. That said, for our particular machine above, there is not a single string that is accepted by the original machine but rejected by the new machine. This is because the only string in the language is 0 and it is its own reverse.

4. **Pumping** (8 points):

(a) (*Graded for correctness*) Give an example of a language over the alphabet $\{0, 1\}$ that has cardinality 3 and for which 5 is a pumping length and 4 is not a pumping length. A complete solution will give a clear and precise description of the language, a justification for why 5 is a pumping length, and a justification for why 4 is not a pumping length. Is this language regular?

Solution: Consider $\{0, 1, 0000\}$. This language has size 3. 4 is not a pumping length because we 0000 is an element of the set which has length 4 but is not pumpable. Indeed, no matter how you split the string into xyz with $y \neq \varepsilon$, xy^2z is not in the language because this string would have a length greater than 4 but our language does not have strings longer than 4.

On the other hand, 5 is a pumping length because there is no string of length 5 in the original language, the pumping lemma is vacuously true. In particular, to disprove that a number is a pumping length for a language, one needs to find a counterexample string of equal length or longer than the candidate pumping length 5 and show that it is not pumpable. However, we don't even have a string longer or equal to 5, let alone a counterexample to pumpability.

(b) (*Graded for completeness*) Consider the following attempted "proof" that the set

$$X = \{uw \mid u \text{ and } w \text{ are strings over } \{0,1\} \text{ and have the same length}\}$$

is nonregular.

> "Proof" that X_1 is not regular using the Pumping Lemma: Let p be an arbitrary positive integer. We will show that p is not a pumping length for X_1.
> Choose s to be the string 1^p0^p, which is in X_1 because we can choose $u = 1^p$ and $w = 0^p$ which each have length p. Since s is in X_1 and has length greater than or equal to p, if p were to be a pumping length for X_1, s ought to be pump'able. That is, there should be a way of dividing s into parts x, y, z where $s = xyz$, $|y| > 0$, $|xy| \leq p$, and for each $i \geq 0$, $xy^iz \in X_1$. Suppose x, y, z are such that $s = xyz$, $|y| > 0$ and $|xy| \leq p$. Since the first p letters of s are all 1 and $|xy| \leq p$, we know that x and y are made up of all 1s. If we let $i = 2$, we get a string xy^iz that is not in X_1 because repeating y twice adds 1s to u but not to w, and strings in X_1 are required to have u and w be the same length. Thus, s is not pumpable (even though it should have been if p were to be a pumping length) and so p is not a pumping length for X_1. Since p was arbitrary, we have demonstrated that X_1 has no pumping length. By the Pumping Lemma, this implies that X_1 is nonregular.

Find the (first and/or most significant) logical error in the "proof" and describe why it's wrong. Then, either prove that the set is actually regular (by finding a regular expression that describes it or a DFA/NFA that recognizes it, and justifying why) **or** fix the proof so that it is logically sound.

Solution: The first mistake happens when the proof claims after setting $i = 2$, the new string is not in the language. This is not necessarily true because the new string we end up might still have even length, in which case we just reassign u and v to be the first half and second half of the new string, noting that the new string is back in X. This language is actually regular. The regular expression for it is $(\Sigma\Sigma)^*$

(c) (*Graded for completeness*) In class and in the reading so far, we've seen the following examples of nonregular sets:

$\{0^n1^n \mid n \geq 0\}$	$\{0^n1^m \mid 0 \leq m \leq n\}$	$\{0^n1^m0^n \mid n, m \geq 0\}$
$\{0^n1^n \mid n \geq 2\}$	$\{0^i1^{2i} \mid 0 \leq i\}$	$\{w \in \{0,1\}^* \mid w = w^R\}$
$\{0^n1^m \mid 0 \leq n \leq m\}$	$\{0^i1^{i+1} \mid 0 \leq i\}$	$\{ww^R \mid w \in \{0,1\}^*\}$

Modify one of these sets in some way and use the Pumping Lemma to prove that the resulting set is still nonregular.

Solution: Consider $\{ww \mid w \in \{0,1\}^*\}$. Suppose it is regular with a pumping length p. We see that 0^p10^p1 is in the language because it is 0^p1 repeated twice. Moreover, it is longer than p. However, pick any splitting $0^p10^p1 = xyz$ with $|xy| \leq p$ and $y \neq \varepsilon$. We must have y consists of one or more 0s. Now, we will show that $xy^0z = xz \neq xyz$ will not be in the language. Suppose it is, then it consists of two copies of some string w. Since xz ends in 1, w ends in 1. But xz only has two 1's, so $w = 0^p1$. But that's a contradiction because $xz \neq xyz = ww$. This establishes that any p is not a pumping length and completes the proof that the set is not regular.

5. **Regular and nonregular languages** (8 points): In Week 2's review quiz, we saw the definition that a set X is said to be **closed under an operation** if, for any elements in X, applying to them gives an element in X. For example, the set of integers is closed under multiplication because if we take any two integers, their product is also an integer .

Prove or disprove each closure claim statement below about the class of regular languages and the class of nonregular languages. Your arguments may refer to theorems proved in the textbook and class, and if they do, should include specific page numbers and references (i.e. write out the claim that was proved in the book and/or class).

Recall the definitions we have:

For languages L_1, L_2 over the alphabet $\Sigma_1 = \{0,1\}$, we have the associated sets of strings

$$SUBSTRING(L_1) = \{w \in \Sigma_1^* \mid \text{there exist } a, b \in \Sigma_1^* \text{ such that } awb \in L_1\}$$

and

$$L_1 \circ L_2 = \{w \in \Sigma_1^* \mid w = uv \text{ for some strings } u \in L_1 \text{ and } v \in L_2\}$$

(a) (*Graded for completeness*) The set of regular languages over $\{0,1\}$ is closed under set-wise concatenation.

Solution: True
It has been shown that if there is a NFA N_1 such that $L(N_1) = A_1$ and NFA N_2 such that $L(N_2) = A_2$, then there is another NFA, let's call it N such that $L(N) = A_1 \circ A_2$.
Let L_1, L_2 be two regular languages over the alphabet $\Sigma_1 = \{0,1\}$. Since they are regular, there is a NFA that recognizes each of them. Call those NFA N_1, N_2 respectively. According to the claim above, there is a NFA N such that $L(N) = L_1 \circ L_2$. So the original claim is true.

(b) (*Graded for completeness*) The set of nonregular languages over $\{0,1\}$ is closed under set-wise concatenation.

Solution: False
Claim: Let $L_1 = \{0^n1^n | n \geq 1\}$, and $L_2 = \Sigma_1^* \backslash L_1$. These languages are nonregular.
Proof: the proof that shows L_1 is nonregular is identical to that of $L = \{0^n1^n | n \geq 0\}$. L_2 can be shown to be nonregular with a proof by contrapositive. Note L_1 is the complement

of L_2. It has been shown that if a language L is regular, its complement would also be regular. Thus, by contrapositive, if a language L's complement is nonregular, then L will also be nonregular.
Claim: Let $L_A = L_1 \cup \{\varepsilon\}$ and $L_B = L_2 \cup \{\varepsilon\}$. These languages are nonregular.
Proof: $L_A = \{0^n1^n | n \geq 1\} \cup \varepsilon = \{0^n1^n | n \geq 1\} \cup \{0^n1^n | n = 0\} = \{0^n1^n | n \geq 0\}$, which has been shown to be nonregular. $L_B = L_2$, since L_2 already includes the empty string
Claim: $L_A \circ L_B = \Sigma_1^*$.
Proof: Let w be a string in Σ_1^*. By construction, L_1 and L_2 are complements of one another. So $w \in L_1$ or $w \in L_2$. WLOG, consider $w \in L_1$. Then $w \in L_A \circ L_B$ by letting $u = w \in L_1$ and $v = \varepsilon \in L_2$. Then $uv = w\varepsilon = w$. Thus $w \in L_A \circ L_B$. The case that $w \in L_2$ is similar. Then it has been shown that $\Sigma_1^* \subseteq L_A \circ L_B$. Since $L_A \circ L_B$ is a language over Σ_1, $L_A \circ L_B \subseteq \Sigma_1^*$. Subset inclusion has been shown in both directions, so the claim is proven.
Conclusion: A counterexample has been provided by showing there exists nonregular languages such that the concatenation of them results in a regular language. So the claim is false.

(c) (*Graded for completeness*) The set of regular languages over $\{0, 1\}$ is closed under the $SUBSTRING$ operation.

Solution: True
Let L be a regular language, then there is an NFA N that recognizes it. We give a construction of an NFA N such that $L(N) = SUBSTRING(L)$, thus proving that language is regular. Below is the construction:

For regular language L given an NFA N recognizing L, we assume WLOG that all states are reachable from the start state. Now, we create q_{start} and q_{end} both not in the set of original states. Connect q_{start} to all the original states with epsilon transitions. Additionally, connect all original states that have a path to an old accept state to q_{end} with epsilon transitions. Let q_{start} be the start state and q_{end} be the only accept state. We claim that this modified NFA N' recognizes $SUBSTRING(L)$.

Given any string $s \in SUBSTRING(L)$, by definition there exist a, b such that $asb \in L$. This means there exists a sequence of transitions in N labelled by the characters of asb (potentially with spontaneous moves) taking us from the start state in N to an accept state. Now, in the new machine N', since we have an epsilon transition from q_{start} to every other state, we can transition to wherever N is after consuming a. Then, since we have not deleted any transition from N and we know asb is accepted, there must be some active computation branches in N' after reading s. But this means s is accepted by N' because we can take an epsilon transition to q_{end}.

For the reverse direction, suppose N' accepts some s. This means have taken some accepting path by reading s. Let's say the path is $q_{start}, q_i, \ldots, q_j, q_{end}$. By construction, q_i is reachable from the original state by consuming some string a. Moreover, we can reach one of the original final states from q_j by consuming some string b. This means asb is accepted by the original NFA N, so $asb \in L$ and $s \in SUBSTRING(L)$

(d) (*Graded for completeness*) The set of nonregular languages over $\{0, 1\}$ is closed under the $SUBSTRING$ operation.

Solution: False.

Consider the canonical non-regular language $\{0^n1^n \mid n \geq 0\}$. I claim $SUBSTRING(\{0^n1^n \mid n \geq 0\}) = \{0^n1^m \mid n, m \geq 0\} = L(0^*1^*)$, a regular language.

Take any string $s \in \{0^n1^m \mid n, m \geq 0\}$, WLOG, s has more 1s than 0s. Then, we can prepend s with a and append s with ε to make it a string inside $\{0^n1^n \mid n \geq 0\}$. a consists of exactly the number of extra 0s needed to match the number of 1s. This shows $s \in SUBSTRING(\{0^n1^n \mid n \geq 0\})$

Take any string $s \in SUBSTRING(\{0^n1^n \mid n \geq 0\})$. We can find a, b such that $asb \in \{0^n1^n \mid n \geq 0\}$. s must have all its 0's before 1's, so it belongs to $\{0^n1^m \mid n, m \geq 0\}$.

This shows equality.