HW3CSE105W25: Sample solutions

CSE105W25 Team

Due: February 6th at 5pm, via Gradescope

In this assignment,

You will demonstrate the richness of the class of regular languages, as well as its boundaries, and explore push-down automata and their design.

Resources: To review the topics for this assignment, see the class material from Week 3 and Week 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 and Section 2.2. Chapter 1 exercises 1.28, 1.29, 1.30. Chapter 1 problem 1.53, 1.54, 1.55. Chapter 2 exercises 2.7, 2.10.

You will submit this assignment via Gradescope (https://www.gradescope.com) in the assignment called "hw3CSE105W25".

Assigned questions

1. **Static analysis** (10 points): In software engineering, static analysis is an approach to debugging and testing where the properties of a piece of code are inferred without actually running it. In the context of finite automata, we can think of static analysis as the process of inferring properties of the language recognized by a finite automaton from properties of the graph underlying its state diagram. The Pumping Lemma is one example of static analysis. In this question, you'll explore other examples of how properties of the graph underlying the state diagram of a machine can give us information about the language recognized by the machine.

(a) (*Graded for completeness*) [1] Suppose you are given an NFA N_0 over an alphabet Σ and each accepting state in N_0 is not reachable from the start state of N_0. What can you conclude about the language of the NFA?

[1] 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 expect you to 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.

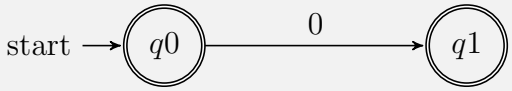
Solution: We can conclude that the language of the NFA is $\emptyset$. If each accepting state is not reachable from the start state, then it is not possible for any computation on any string over Σ to process the whole string and end in an accepting state. Therefore, the NFA N_0 rejects all strings, and $L(N_0) = \emptyset$.

(b) (*Graded for correctness*) [2] Prove or disprove: For any alphabet Σ and any DFA M over Σ, if every state in M is accepting then $L(M) = \Sigma^*$. A complete answer will clearly indicate whether the statement is true or false and then will justify with a complete and correct argument.

Solution: This statement is true. By definition of DFA, the computation of a DFA on any string over the alphabet must end in one of the states. If every state in DFA M is accepting, then the computation of M on any string over Σ ends in an accepting state. By the acceptance condition of DFA, M accepts all strings over Σ, $L(M) = \Sigma^*$.

(c) (*Graded for correctness*) Prove or disprove: For any alphabet Σ and any NFA N over Σ, if every state in N is accepting then $L(N) = \Sigma^*$. A complete answer will clearly indicate whether the statement is true or false and then will justify with a complete and correct argument.

Solution: The statement is false. Consider the following counterexample. Let $\Sigma = \{0\}$ and let N be the NFA with the state diagram below. Every state in N is accepting, but $L(N) = \{\varepsilon, 0\}$ (we accept ε since q_0 is an accepting state, and we accept 0 since we can start from q_0, read 0, and transit to q_1, which processes the whole string and ends in an accept state; all other strings over Σ cannot be fully processed, i.e. the computations will be stuck at q_1 so they are all rejected), which is not equal to $\{0\}^*$.



2. **Multiple representations** (12 points):

(a) Consider the language $A_1 = \{uw \mid u \text{ and } w \text{ are strings over } \{0,1\} \text{ and have the same length}\}$ and the following argument.

> "Proof" that A_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 A_1.
> Choose s to be the string 1^p0^p, which is in A_1 because we can choose $u = 1^p$ and $w = 0^p$ which each have length p. Since s is in A_1 and has length greater than or

[2] 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.

equal to p, if p were to be a pumping length for A_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 $i \geq 0$, $xy^iz \in A_1$. 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 A_1 because repeating y twice adds 1s to u but not to w, and strings in A_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 A_1. Since p was arbitrary, we have demonstrated that A_1 has no pumping length. By the Pumping Lemma, this implies that A_1 is nonregular.

i. (*Graded for completeness*) Find the (first and/or most significant) logical error in the "proof" above and describe why it's wrong.

> **Solution:** The part that "If we let $i = 2$, we get a string xy^iz that is not in A_1" is incorrect because the new string can still have even length, in which case we can reassign u and w to be the new first half and second half so we can see that the new string is still in A_1. For example, when we choose integer $p = 2$, then for any string s in A_1 of length at least 2, s can be divided into three pieces $s = xyz$ where we let $x = \varepsilon$ and let y be the first two characters of s which satisfies $|xy| \leq p$. In this case, for each $i \geq 0$, $xy^iz \in A_1$ since xyz is even length and y is also even length, which makes xy^iz also even length.

ii. (*Graded for completeness*) Prove that the set A_1 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:** Restating the definition of A_1,
>
> $$\begin{aligned} A_1 &= \{uw \mid u, w \in \{0,1\}^* \text{ and } |u| = |w|\} \\ &= \{z \mid z \in \{0,1\}^* \text{ and } z \text{ can be split into equal length strings}\} \end{aligned}$$
>
> is the set of strings over $\{0, 1\}$ with even lengths. Thus the regular expression $((0 \cup 1)(0 \cup 1))^*$ can describe A_1, therefore A_1 is regular.

(b) Consider the language $A_2 = \{u1w \mid u \text{ and } w \text{ are strings over } \{0,1\} \text{ and have the same length}\}$ and the following argument.

"Proof" that A_2 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 A_2.
Choose s to be the string $1^{p+1}0^p$, which is in A_2 because we can choose $u = 1^p$ and $w = 0^p$ which each have length p. Since s is in A_2 and has length greater than or equal to p, if p were to be a pumping length for A_2, 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 $i \geq 0$, $xy^iz \in A_2$. When $x = \varepsilon$ and $y = 1^{p+1}$ and $z = 0^p$, we

have satisfied that $s = xyz$, $|y| > 0$ (because p is positive) and $|xy| \leq p$. If we let $i = 0$, we get the string $xy^iz = 0^p$ that is not in A_2 because its middle symbol is a 0, not a 1. 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 A_2. Since p was arbitrary, we have demonstrated that A_2 has no pumping length. By the Pumping Lemma, this implies that A_2 is nonregular.

i. (*Graded for completeness*) Find the (first and/or most significant) logical error in the "proof" above and describe why it's wrong.

> **Solution:** The part that "When $x = \varepsilon$ and $y = 1^{p+1}$ and $z = 0^p$, we have satisfied that $s = xyz$, $|y| > 0$ (because p is positive) and $|xy| \leq p$" is incorrect. Firstly, here by the choices, $|xy| = p + 1 > p$ so it does not satisfy $|xy| \leq p$. But the more significant problem is that we want to prove that there is no way to split $s = xyz$ such that $|y| > 0$, $|xy| \leq p$, and for each $i \geq 0$, $xy^iz \in A_2$, therefore we should not only prove that one particular choice of x, y, z does not work. Instead, we should prove that for all possible splits $s = xyz$ s.t. $|y| > 0$, $|xy| \leq p$, there exists some $i \geq 0$, where $xy^iz \notin A_2$.

ii. (*Graded for completeness*) Prove that the set A_2 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:** A_2 is non-regular. We can fix the proof:
>
> Let p be an arbitrary positive integer. We will show that p is not a pumping length for A_2. Choose s to be the string $1^{p+1}0^p$, which is in A_2 because we can choose $u = 1^p$ and $w = 0^p$ which each have length p. Since s is in A_2 and has length greater than or equal to p, if p were to be a pumping length for A_2, 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 $i \geq 0$, $xy^iz \in A_2$. **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 $i = 0$, we get the string xy^iz that is not in A_2 because given $|y| > 0$, either the new string becomes even length which do not have a middle symbol, or the new string has odd length but the middle symbol is a 0, not a 1.** 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 A_2. Since p was arbitrary, we have demonstrated that A_2 has no pumping length. By the Pumping Lemma, this implies that A_2 is nonregular.

3. **Pumping** (10 points):

(a) (*Graded for correctness*) Give an example of a language over the alphabet $\{a, b\}$ that has cardinality 3 and for which 5 is a pumping length and 4 is not a pumping length. Is this

language regular? A complete solution will give (1) a clear and precise description of the language, (2) a justification for why 5 is a pumping length, (3) a justification for why 4 is not a pumping length, (4) a correct and justified answer to whether the language is regular.

Solution: Consider the language $L = \{aaa, aabb, bbbb\}$. This language has exactly 3 distinct elements.

We will prove that 5 is a pumping length of L: To prove this, we want to show that for each string $s \in L$ and $|s| \geq 5$, there are strings x, y, z such that $s = xyz$ satisfies for each $i \geq 0$, $xy^iz \in L$; $|y| > 0$; and $|xy| \leq 5$. All strings in L have length less than 5. Thus, there is no string in L with length at least 5. So the statement above is vacuously true. Therefore, 5 is a pumping length of L.

We will now prove that 4 is not a pumping length of L: To prove this, we want to find a counterexample string $s \in L, |s| \geq 4$ such that for any x, y, z where $s = xyz$, $|y| > 0$, and $|xy| \leq 4$, there exists $i \geq 0$ such that $xy^iz \notin L$. Take $s = aabb \in L$ and we will show it is a counterexample. First, it is a string of length 4. Now, consider any x, y, z where $s = xyz$, $|y| > 0$, and $|xy| \leq 4$. Let $i = 5$ so $xy^iz = xy^5z$. Since $|y| > 0$, we have $|xy^5z| \geq 5$. However, all strings in L have length less than 5, therefore the string that we get when $i = 5$, xy^5z, is not in L. Thus we proved that 4 is not a pumping length of L.

The language L is regular, since it can be described by the regular expression $aaa \cup aabb \cup bbbb$. (This also follows from the observation that every finite language is regular.)

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

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

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

Solution: Consider the language $A = \{0^n1^{3n} \mid n \geq 0\}$. We will prove that this language is nonregular using the Pumping Lemma.

Let p be an arbitrary positive integer. We will show that p is not a pumping length for A. Choose s to be the string 0^p1^{3p}. Since s is in A and has length greater than or equal to p, if p were to be a pumping length for A, 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 $i \geq 0$, $xy^iz \in A$. Since the first p letters of s are all 0 and $|xy| \leq p$, we know that x and y are made up of all 0s. Denote $x = 0^m, y = 0^n, z = 0^{p-m-n}1^{3p}$, where $m \geq 0, n > 0, m+n \leq p$. If we let $i = 0$, we get the string $xy^iz = xz = 0^{p-n}1^{3p}$ that is not in A because given $|y| > 0, n > 0$, we have $3p > 3(p-n)$. 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 A. Since p was arbitrary, we have demonstrated that A has no pumping length. By the Pumping Lemma, this implies that A is nonregular.

4. **Regular and nonregular languages** (12 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 .

Recall the definitions of operations we've talked about that produce new languages from old: for each language L over an alphabet Σ, we have the associated sets of strings (also over Σ)

$$L^* = \{w_1 \cdots w_k \mid k \geq 0 \text{ and each } w_i \in L\}$$

and

$$SUBSTRING(L) = \{w \in \Sigma^* \mid \text{there exist } x, y \in \Sigma^* \text{ such that } xwy \in L\}$$

and

$$EXTEND(L) = \{w \in \Sigma^* \mid w = uv \text{ for some strings } u \in L \text{ and } v \in \Sigma^*\}$$

Also, recall the set operations union and intersection: for any sets X and Y

$$X \cup Y = \{w \mid w \in X \text{ or } w \in Y\}$$

$$X \cap Y = \{w \mid w \in X \text{ and } w \in Y\}$$

(a) (*Graded for correctness*) Use the general constructions that we developed to prove the closure of the class of regular languages under various operations to produce the state diagram of a NFA that recognizes the language

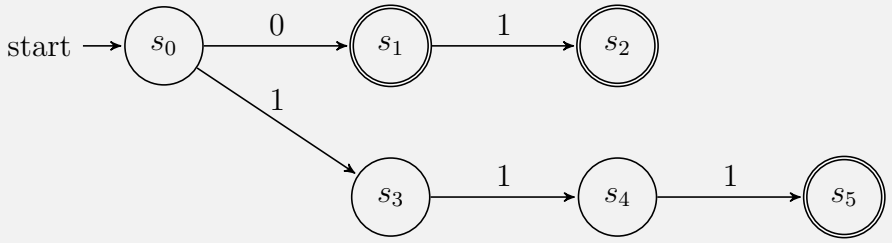
$$(\ SUBSTRING(\{0, 01, 111\})\ )^*$$

Hint: Question 4 from Homework 2 might be helpful.

For full credit, submit (1) a state diagram of an NFA that recognizes $\{0, 01, 111\}$, (2) a state diagram of an NFA that recognizes $SUBSTRING(\{0, 01, 111\})$, and (3) a state diagram of an NFA that recognizes $(\ SUBSTRING(\{0, 01, 111\})\ )^*$, and a brief justification of each state diagram that references the language being recognized or the general constructions being used.

Solution:

(1) a state diagram of an NFA that recognizes $\{0, 01, 111\}$:



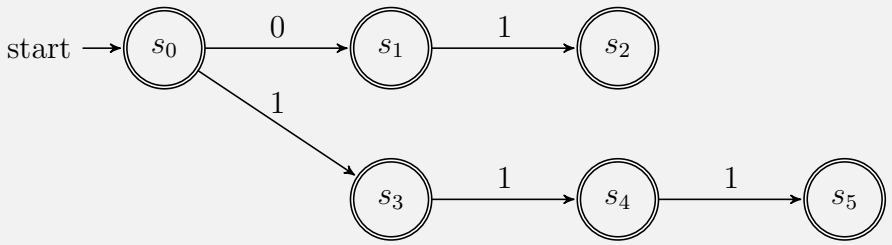
All strings in the language $L = \{0, 01, 111\}$ are accepted by the above NFA N_1:

- There exists an accepting computation of this NFA on input string 0: $s_0 \xrightarrow{0} s_1$
- There exists an accepting computation of this NFA on input string 01: $s_0 \xrightarrow{0} s_1 \xrightarrow{1} s_2$
- There exists an accepting computation of this NFA on input string 111: $s_0 \xrightarrow{1} s_3 \xrightarrow{1} s_4 \xrightarrow{1} s_5$

On the other hand, for all other strings, there either does not exist a computation that can process the whole string, or there exists a computation that processes the whole string but does not end in an accepting state. Therefore, we reject all strings not in L.

(2) a state diagram of an NFA that recognizes $SUBSTRING(\{0, 01, 111\})$:

By the definition of $SUBSTRING$, $SUBSTRING(\{0, 01, 111\})$ contains all substrings of strings in $\{0, 01, 111\}$ so $SUBSTRING(\{0, 01, 111\}) = \{\varepsilon, 0, 1, 01, 11, 111\}$. The following NFA N_2 accepts exactly those six strings in $SUBSTRING(\{0, 01, 111\})$ and reject all other strings:

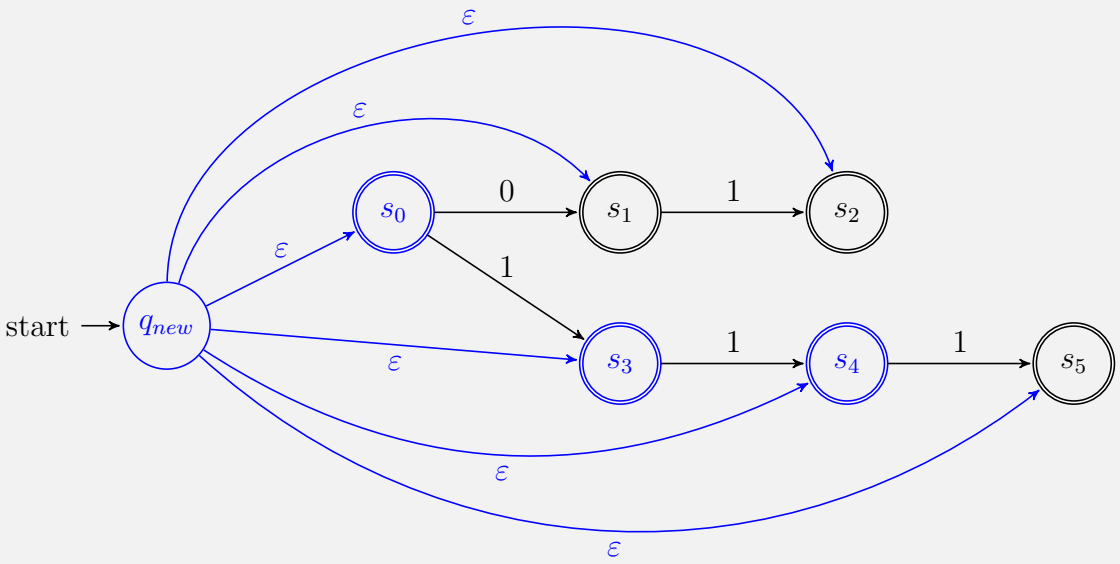


Alternatively, we can slightly modify the N_γ construction from Homework 2 to build an NFA that also recognizes $SUBSTRING(\{0, 01, 111\})$. Let's define the modified construction: Take an **NFA** $N = (Q, \Sigma, \delta, q_0, F)$. We can build a new NFA $N_{new} = (Q \cup \{q_{new}\}, \Sigma, \delta_{new}, q_{new}, F_{new})$ where $q_{new} \notin Q$ and $F_{new} = \{q \in Q \mid$

$\exists w \in \Sigma^*(\delta^*((q,w)) \in F)\}$, and

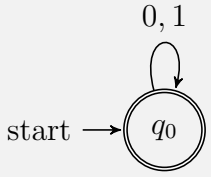
$$\delta_{new}((q,a)) = \begin{cases} \delta((q,a)) & \text{if } q \in Q,\ a \in \Sigma_\varepsilon \\ \{q' \in Q \mid \exists w \in \Sigma^*(\delta^*((q_0,w)) = q')\} & \text{if } q = q_{new},\ a = \varepsilon \\ \emptyset & \text{if } q = q_{new},\ a \in \Sigma \end{cases}$$

There are spontaneous moves from the new start state to any state reachable in the original NFA from the original start state. The new set of accepting states is the collection of states in the original NFA from which an accepting state (in the original NFA) is reachable. With these definitions, N_{new} accepts all strings that are each a substring of some string in $L(N)$, and rejects all other stringsm thus $L(N_{new}) = SUBSTRING(L(N))$. Let's use this construction on N_1 (where $L(N_1) = \{0, 01, 111\}$) to create NFA N_3 such that $L(N_3) = SUBSTRING(L(N_1)) = SUBSTRING(\{0, 01, 111\})$:

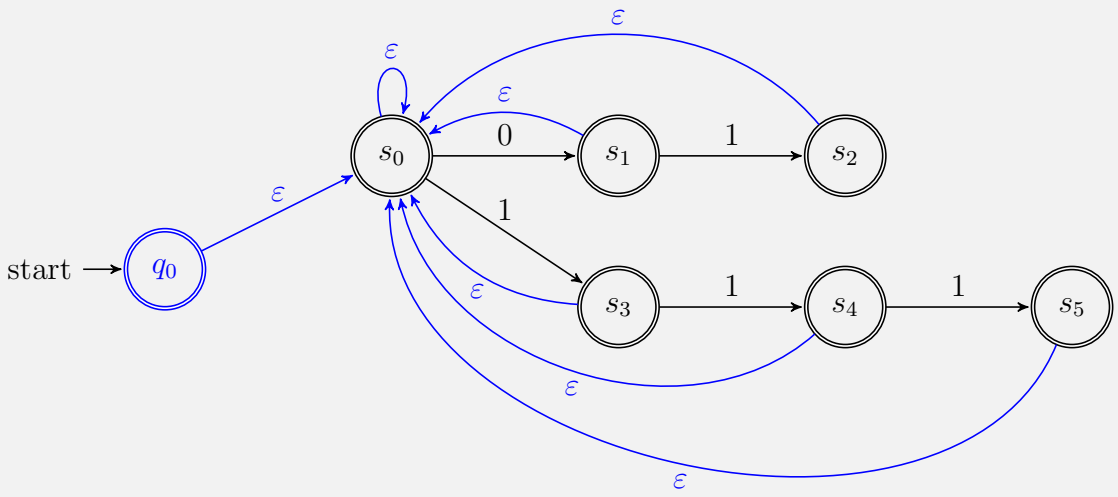


(3) a state diagram of an NFA that recognizes ($SUBSTRING(\{0, 01, 111\})$)*:

Note that $SUBSTRING(\{0, 01, 111\}) = \{\varepsilon, 0, 1, 01, 11, 111\}$. Since 0 and 1 are both elements of this set, when we apply the Kleene star operation, each "slot" can be filled with either a 0 or 1. Thus, ($SUBSTRING(\{0, 01, 111\})$)* = $\{0, 1\}^*$. Thus, an NFA N_4 that recognizes the language can be:



Alternatively, let's apply the construction in week 3 notes page 8 on N_2 (we can also apply it to N_3, but let's use N_2 for simplicity of the diagram) to get NFA N_5 where $L(N_5) = L(N_2)^* = (\ SUBSTRING(\{0, 01, 111\})\ )^*$:



(b) (*Graded for completeness*) Prove that the class of nonregular languages over $\{0, 1\}$ is not closed under the $SUBSTRING$ operation by giving an example language A that is nonregular but for which $SUBSTRING(A)$ is regular. A complete solution will give (1) a clear and precise description of the language, (2) a justification for why it is nonregular (either by proving this directly or by referring to specific examples from the class or textbook), (3) a description of the result of applying the $SUBSTRING$ operation to the language, and (4) a justification for why this resulting language is regular.

Solution: Consider example language $A = \{0^n1^n \mid n \geq 0\}$. A is a nonregular language as proved in week 4 notes page 5. By the definition that $SUBSTRING(L)$ consists of all strings that are each a substring of some string in L, we get that $SUBSTRING(L) = \{0^m1^n \mid m \geq 0, n \geq 0\}$ (erasing some of the 0s from the beginning and/or the 1s from the end). Thus $SUBSTRING(L)$ can be described by regular expression 0^*1^*, which means that $SUBSTRING(L)$ is a regular language (Sipser, Theorem 1.54: "A language is regular if and only if some regular expression describes it"). Therefore, the set of nonregular languages over $\{0, 1\}$ is not closed under the $SUBSTRING$ operation.

(c) (*Graded for correctness*) Prove that the class of nonregular languages over $\{0, 1\}$ is not closed under the $EXTEND$ operation by giving an example language B that is nonregular but for which $EXTEND(B)$ is regular. A complete solution will give (1) a clear and precise description of the language, (2) a justification for why it is nonregular (either by proving this directly or by referring to specific examples from the class or textbook), (3) a description

of the result of applying the $EXTEND$ operation to the language, and (4) a justification for why this resulting language is regular.

Solution: Consider example language $B = \{0^n1^n \mid n \geq 0\}$. B is a nonregular language as proved in week 4 notes page 5. We get that $EXTEND(L) = \{0,1\}^*$ since by definition, $EXTEND(L)$ consists of all strings that are each the result of taking a string in L and concatenating it with a string over $\{0,1\}$. Since $\varepsilon \in L$ (setting $n = 0$), we can get any string over $\{0,1\}$ in $EXTEND(L)$ (as the result of concatenating ε with this string). Thus $EXTEND(L)$ can be described by regular expression Σ^*, which means that $EXTEND(L)$ is a regular language (Sipser, Theorem 1.54: "A language is regular if and only if some regular expression describes it"). Therefore, the set of nonregular languages over $\{0,1\}$ is not closed under the $EXTEND$ operation.

(d) (*Graded for completeness*) Prove that the class of nonregular languages over $\{0,1\}$ is not closed under the Kleene star operation by giving an example language C that is nonregular but for which C^* is regular. A complete solution will give (1) a clear and precise description of the language, (2) a justification for why it is nonregular (either by proving this directly or by referring to specific examples from the class or textbook), (3) a description of the result of applying the Kleene star operation to the language, and (4) a justification for why this resulting language is regular.

Solution: Consider example language $C = \{w \in \{0,1\}^* \mid w = w^{\mathcal{R}}\}$. Let's prove C is nonregular using the Pumping Lemma: Let p be an arbitrary positive integer. We will show that p is not a pumping length for C. Choose s to be the string 0^p110^p. Since s is in C and has length greater than or equal to p, if p were to be a pumping length for C, 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 $i \geq 0$, $xy^iz \in C$. Since the first p letters of s are all 0 and $|xy| \leq p$, we know that x and y are made up of all 0s. Denote $x = 0^m, y = 0^n, z = 0^{p-m-n}110^p$, where $m \geq 0, n > 0, m + n \leq p$. If we let $i = 0$, we get the string $xy^iz = xz = 0^{p-n}110^p$ that is not in C because given $|y| > 0, n > 0$, we have $p > p - n$, so $0^{p-n}110^p \neq (0^{p-n}110^p)^{\mathcal{R}} = 0^p110^{p-n}$. 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 C. Since p was arbitrary, we have demonstrated that C has no pumping length. By the Pumping Lemma, this implies that C is nonregular.

The result of applying Kleene star to C is $C^* = \{0,1\}^*$: According to the definition of Kleene star operation, $C^* = \{w_1...w_k \mid k \geq 0 \text{ and each } w_i \in C\}$. Since $0 \in C$ and $1 \in C$, we know that C^* contains all strings that are zero or more 0's and 1's concatenated together, which means that $\{0,1\}^* \subseteq C^*$. Since C^* should be a language over the alphabet $\{0,1\}$, we also have $C^* \subseteq \{0,1\}^*$. Therefore we proved that $C^* = \{0,1\}^*$. We know that C^* is regular because there is a regular expression $(0 \cup 1)^*$ that describes it.

5. **Regular and nonregular languages and Push-down automata (PDA)** (6 points): On page 7 of the week 4 notes, we have the following list of languages over the alphabet $\{a, b\}$

$$\{a^n b^n \mid 0 \leq n \leq 5\} \quad \{b^n a^n \mid n \geq 2\} \quad \{a^m b^n \mid 0 \leq m \leq n\}$$
$$\{a^m b^n \mid m \geq n+3, n \geq 0\} \quad \{b^m a^n \mid m \geq 1, n \geq 3\}$$
$$\{w \in \{a, b\}^* \mid w = w^{\mathcal{R}}\} \qquad \{ww^{\mathcal{R}} \mid w \in \{a, b\}^*\}$$

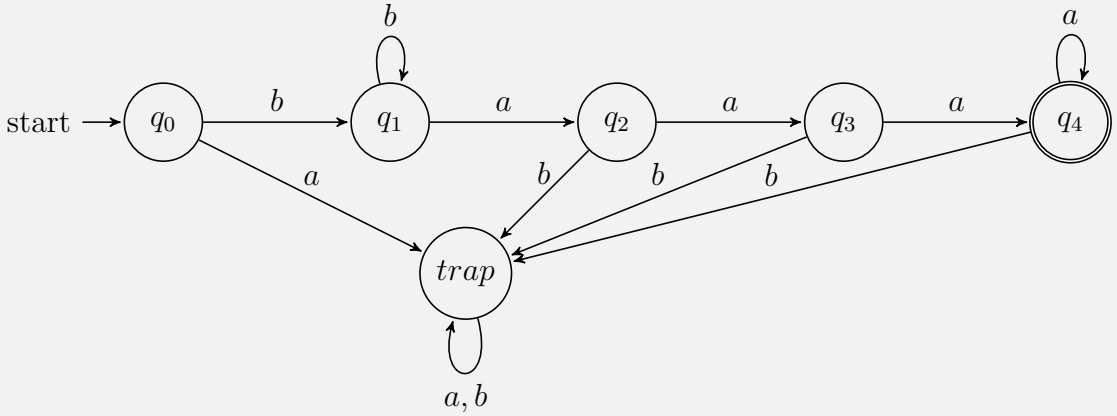
(a) (*Graded for completeness*) Pick one of the regular languages and design a regular expression that describes it and a DFA that recognizes it. Briefly justify your regular expression by connecting the subexpressions of it to the intended language and referencing relevant definitions. Briefly justify your DFA design 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.

Solution: We pick the language,

$$L = \{b^m a^n \mid m \geq 1, n \geq 3\}$$

and will prove that it is regular. A regular expression that describes this language is $R = bb^*aaaa^*$. By definition of regular expressions, R describes the set of strings starting with one or more b's followed by three or more a's, which is exactly the language L.

We have the following DFA recognizing this language.



Let's justify our DFA by explaining the role of each state

- q_0: We shouldn't accept the empty string because we need at least one b and three a's, so q_0 as the start state is not an accept state.
- q_1: At this state, we know the string starts with a b and can continue to process the following b's.
- q_2, q_3: q_2 means we have just read one a after the b's, and q_3 means we have just read two consecutive a's after the b's. If we see a b after reaching q_2, q_3, we

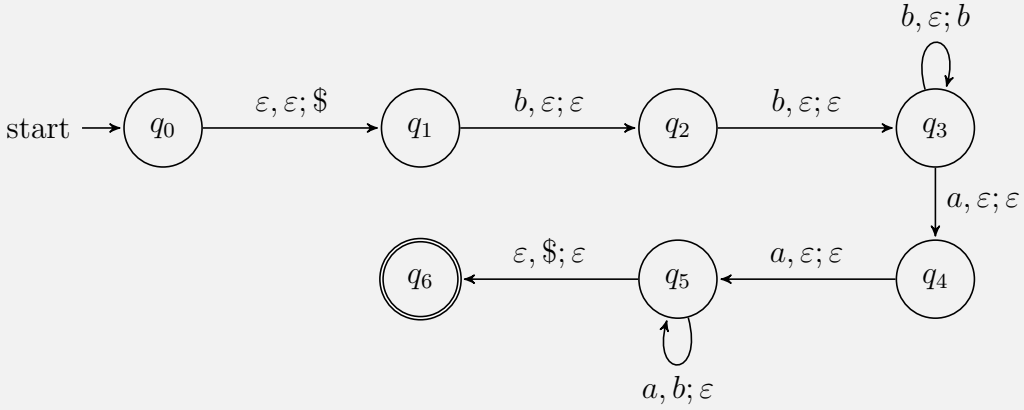
transition to our trap state.

- q_4: We've seen the string starting with at least one b followed by at least three a's so we can accept. If we see more a's we can continue to accept, but we should transition to the trap state if we see another b.
- *trap*: We don't want to accept strings that get to this state since they violate the format of the strings that should be accepted.

This DFA would process the string from start to finish, starting from q_0. An accepting computation would run from q_0 to q_4, possibly looping at q_1 and/or q_4. A rejecting computation could end in any of the other non-accept states. If it ends in *trap*, then the string breaks the pattern in the language. Otherwise, the computation ends in q_0, q_1, q_2 or q_3 so the string is missing some ending characters to be accepted.

(b) (*Graded for completeness*) Pick one of the nonregular languages and design a PDA that recognizes it. Draw the state diagram of your PDA. Briefly justify your design 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.

Solution: Let's choose the language $\{b^n a^n \mid n \geq 2\}$. The state diagram of the PDA that recognizes the language is the picture as follows.



- q_0: Start state
- q_1: At this state, our stack has \$, which marks the bottom of the stack.
- q_2: At this state, we've seen the first b. Note that there must be at least two bs in a row for a string this language because $n \geq 2$.
- q_3: At this state, we've seen two or more bs. After seeing the first two bs, any additional bs in the string will mean a b is pushed to the stack as a counter.

- $q4$: At this state, we've seen the first a after the bs. Note that at least two as must be in the string for the same reason there are at least two bs.
- $q5$: At this state, we've seen two or more as after the bs. After seeing the first two as, any additional as will result in popping a b off of the stack unless there are no more bs to pop.
- $q6$: At this state, we accept a string if there are no more characters left to read. Note that we intended to reach an empty stack to ensure that the number of bs and as are the same.

If a string is accepted by this PDA, it means that there are at least two bs followed by at least two as, and the number of as is the same as the number of bs. We made sure there are least two of each character, which means the condition $n \geq 2$ is satisfied. And only when the number of as is equal to the number of bs, we know we have reached the bottom of the stack and can accept, which is exactly what our language specifies.

On the other hand, if a string is rejected by the machine, then that means that the string either does not have the same amount of each character in that order or doesn't have at least two of each character. They will be "rejected" because we either can't read the whole string, or because we cannot land in an accept state. Therefore, all strings rejected by the machine are not in the language.