

Problem Set 6

This fifth problem set explores the regular languages and their properties. This will be your first foray into computability theory, and I hope you find it fun and exciting!

As always, please feel free to drop by office hours, ask on Piazza, or email the staff list if you have any questions. We'd be happy to help out.

Good luck, and have fun!

Due Friday, November 6 at the start of class.

Problem One: Constructing DFAs (8 Points)

For each of the following languages over the indicated alphabets, construct a DFA that accepts precisely the strings that are in the indicated language. Your DFA does not have to have the fewest number of states possible. You should specify your DFA as either a state-transition diagram (the graphical representation we've seen in class) or as a table.

We have an online tool you can use to design, test, and submit the DFAs in this problem. To use it, visit the CS103 website and click the “DFA/NFA Editor” link under the “Resources” header. We strongly recommend this tool, as it makes it easy to design, test, and submit your solutions. If you submit through this system, please have only one team member submit your automata, and include the name of that person in your submission.

- i. For the alphabet $\Sigma = \{a, b, c\}$, construct a DFA for the language $\{ w \in \Sigma^* \mid w \text{ contains exactly two } cs. \}$
- ii. For the alphabet $\Sigma = \{a, b\}$, construct a DFA for the language $\{ w \in \Sigma^* \mid w \text{ contains the same number of instances of the substring } ab \text{ and the substring } ba \}$. Note that substrings are allowed to overlap, so $aba \in L$ and $babab \in L$.
- iii. For the alphabet $\Sigma = \{a, b, c, \dots, z\}$, construct a DFA for the language $\{ w \in \Sigma^* \mid w \text{ contains the word “cocoa” as a substring} \}$. As a shorthand, you can specify multiple letters in a transition by using set operations on Σ (for example, $\Sigma - \{a, b\}$)*
- iv. Suppose that you are taking a walk with your dog along a straight-line path. Your dog is on a leash that has length two, meaning that the distance between you and your dog can be at most two units. You and your dog start at the same position. Consider the alphabet $\Sigma = \{y, d\}$. A string in Σ^* can be thought of as a series of events in which either you or your dog moves forward one unit. For example, the string “yydd” means that you take two steps forward, then your dog takes two steps forward. Let $L = \{ w \in \Sigma^* \mid w \text{ describes a series of steps that ensures that you and your dog are never more than two units apart} \}$. Construct a DFA for L .

Problem Two: Constructing NFAs (6 Points)

For each of the following languages over the indicated alphabets, construct an NFA that accepts precisely the strings that are in the indicated language. You should specify your NFA as either a state-transition diagram (the graphical representation we've seen in class) or as a table. Your NFA may use ϵ -transitions if you wish and does not have to have the fewest number of states possible. **We recommend designing, testing, and submitting your automata using our online system;** see above for details.

- i. For the alphabet $\Sigma = \{a, b, c\}$, construct an NFA for the language $\{ w \in \Sigma^* \mid w \text{ ends in } a, bb, \text{ or } ccc \}$.
- ii. For the alphabet $\Sigma = \{a, b, c, d, e\}$, construct an NFA for the language $\{ w \in \Sigma^* \mid \text{the last character of } w \text{ appears nowhere else in } w, \text{ and } |w| \geq 1 \}$.
- iii. For the alphabet $\Sigma = \{a, b\}$, construct an NFA for the language $\{ w \in \Sigma^* \mid w \text{ contains at least two } b\text{'s with exactly five characters between them} \}$. For example, baaaaab is in the language, as is aabaababb and abbbbbbaaaaaaaab, but bbbbbb is not, nor are bbbbab or aaabab.

* DFAs are often used to search large blocks of text for specific substrings, and several string searching algorithms are built on top of specially-constructed DFAs. The *Knuth-Morris-Pratt* and *Aho-Corasick* algorithms use slightly modified DFAs to find substrings extremely efficiently.

Problem Three: $\wp(\Sigma^*)$ (2 Points)

Let Σ be an alphabet. Give a brief English description of the set $\wp(\Sigma^*)$. Briefly justify your answer. (*We think that there is a single "best answer" to this question. Chances are that if you find it, you'll know.*)

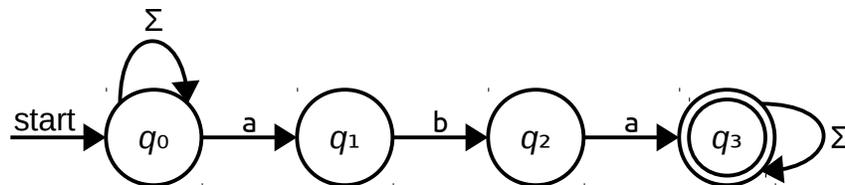
Problem Four: Cardinalities and Concatenations (3 Points)

Recall that if L is a language, L^n is the n -fold concatenation of L with itself. (As a special case, the language L^0 is defined to be $\{\epsilon\}$).

Prove or disprove: if L is a language and $k \geq 1$, then $|L^k| = |L|^k$.

Problem Five: The Subset Construction (4 Points)

Consider the following NFA over the alphabet $\{a, b\}$:



This question explores this NFA in the context of the subset construction.

- Using the subset construction, convert this NFA to a DFA. Do **not** just find a DFA that has the same language as this NFA; we want you to use the subset construction algorithm detailed in lecture. For simplicity, please represent the resulting DFA as a transition table.
- Design a DFA with the same language as this NFA, but which has fewer states than the DFA you produced with the subset construction. This shows that the subset construction can always convert an NFA into DFA, but it might not produce the optimal DFA.

Problem Six: Derivatives of Regular Languages (4 Points)

Let L be a language over Σ . For any $a \in \Sigma$, the *derivative of L with respect to a* , denoted $\partial_a L$, is the following language:

$$\partial_a L = \{ w \in \Sigma^* \mid aw \in L \}$$

Intuitively, you can think of $\partial_a L$ as follows: start with the language L , then remove from L all strings that don't start with a . This leaves behind a language consisting only of strings beginning with a . Then, delete a from the front of each remaining string. The set of strings you're left with is $\partial_a L$.

As an example, let $\Sigma = \{a, b\}$ and let $L = \{ \epsilon, a, aabb, aaab, ba, bb \}$. Then $\partial_a L = \{ \epsilon, abb, aab \}$.

In this problem, we'd like you to prove that if L is a regular language over Σ and $a \in \Sigma$, then $\partial_a L$ is regular as well. This shows that the regular languages are closed under differentiation.

- Let L be a regular language over Σ and let $a \in \Sigma$. Describe a construction that transforms a DFA for L into a DFA for $\partial_a L$.
- Prove that your construction from part (i) is correct by arguing the following: your transformed DFA accepts a string $w \in \Sigma^*$ if and only if $w \in \partial_a L$.

Problem Seven: Why the Extra State? (2 Points)

In our proof that the regular languages are closed under the Kleene closure operator (that is, if L is regular, then L^* is regular), we used the following construction:

1. Begin with an NFA N where $\mathcal{L}(N) = L$.
2. Add in a new start state q_{start} .
3. Add an ϵ -transition from q_{start} to the start state of N .
4. Add ϵ -transitions from each accepting state of N to q_{start} .
5. Make q_{start} an accepting state.

You might have wondered why we needed to add q_{start} as a new state to the NFA. It might have seemed more natural to do the following:

1. Begin with an NFA N where $\mathcal{L}(N) = L$.
2. Add ϵ -transitions from each accepting state of N to the start state of N .
3. Make the start state of N an accepting state.

Unfortunately, this construction does not work correctly.

Find a regular language L and an NFA N for L such that using the second construction does not create an NFA for L^* . Justify why the language of the new NFA isn't L^* .

We will cover the material necessary to solve the remaining problems in Monday's lecture.

Problem Eight: Designing Regular Expressions (8 Points)

Below are a list of alphabets and languages over those alphabets. For each language, write a regular expression for that language.

We have an online tool you can use to design, test, and submit the regular expressions in this problem. To use it, visit the CS103 website and click the “Regex Editor” link under the “Resources” header. We strongly suggest using this tool, as it makes it easy to design, test, debug, and submit your solutions. If you submit through this system, please have only one team member submit your regular expressions, and include the name of that person in your submission.

- i. Let $\Sigma = \{a, b\}$ and let $L = \{ w \in \Sigma^* \mid w \text{ does not contain } ba \text{ as a substring} \}$. Write a regular expression for L .
- ii. Let $\Sigma = \{a, b\}$ and let $L = \{ w \in \Sigma^* \mid w \text{ does not contain } bb \text{ as a substring} \}$. Write a regular expression for L .
- iii. Suppose you are taking a walk with your dog on a leash (as in Problem 1.iv). As in that problem, your leash has length two. Let $\Sigma = \{y, d\}$ and let $L = \{ w \in \Sigma^* \mid w \text{ represents a walk with your dog on a leash where you and your dog both end up at the same location} \}$. For example, the string $yydddyy$ is in L because you and your dog are never more than two steps apart and both of you end up four steps ahead of where you started; similarly, $ddyddy \in L$. However, $yyyyddd \notin L$, since halfway through your walk you are three steps ahead of your dog; $ddy \notin L$, because your dog ends up two steps ahead of you; and $ddyddy \notin L$, because at one point in your walk your dog is three steps ahead of you. Write a regular expression for L .
- iv. Let $\Sigma = \{a, b\}$ and let $L = \{ w \in \Sigma^* \mid w \neq ab \}$. Write a regular expression for L .

Problem Nine: Finite and Cofinite Languages (3 Points)

A language L is called *finite* if L contains finitely many strings. More precisely, a language L is a finite language if $|L|$ is a natural number. A language L is called *cofinite* if its complement is a finite language; that is, L is cofinite if $|\bar{L}|$ is a natural number.

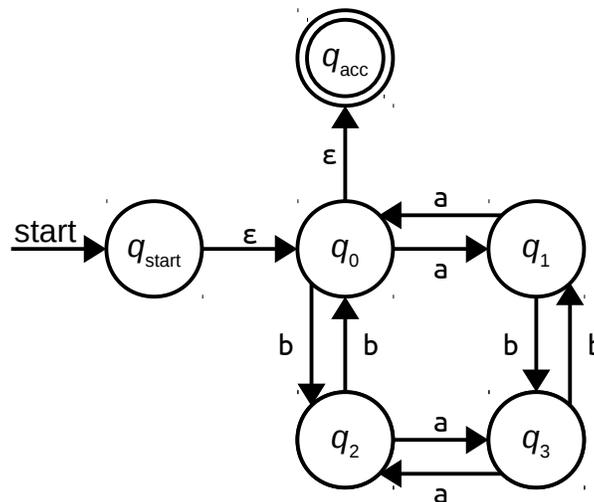
- i. Prove that any finite language is regular.
- ii. Prove that any cofinite language is regular.

These results are important. You'll see them come up later in the course.

Problem Ten: State Elimination (3 Points)

The state elimination algorithm gives a way to transform a finite automaton (DFA or NFA) into a regular expression. It's a really beautiful algorithm once you get the hang of it, so we thought that we'd let you try it out on a particular example.

Let $\Sigma = \{a, b\}$ and let $L = \{w \in \Sigma^* \mid w \text{ has an even number of } a\text{'s and an even number of } b\text{'s}\}$. Below is a finite automaton for L that we've prepared for the state elimination algorithm by adding in a new start state q_{start} and a new accept state q_{acc} :



We'd like you to use the state elimination algorithm to produce a regular expression for L .

- i. Run two steps of the state elimination algorithm on the above automaton. Specifically, first remove state q_1 , then remove state q_2 . Show your result at this point.
- ii. Finish the state elimination algorithm. What regular expression do you get for L ?

Extra Credit Problem: Cyclic Shifting (1 Point Extra Credit)

Let $w \in \Sigma^*$ be a string. A string z is called a *cyclic shift* of w if there exist $x, y \in \Sigma^*$ such that $xy = w$ and $yx = z$. If L is a language over Σ , then the *cyclic shift* of L , denoted $Cycle(L)$, is defined as follows:

$$Cycle(L) = \{z \in \Sigma^* \mid \text{there is some } w \in L \text{ where } z \text{ is a cyclic shift of } w\}$$

Prove that if L is a regular language over Σ , then $Cycle(L)$ is also regular.