# Problem Set 5

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.

This problem set has 38 possible points. It is weighted at 5% of your total grade.

Good luck, and have fun!

**There is no checkpoint for this assignment.**

**Due Monday, May 11th 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 $\Sigma = \{\texttt{a}, \texttt{b}, \texttt{c}\}$, construct a DFA for the language $\{ w \in \Sigma^* \mid w$ contains exactly two `c`s. $\}$

ii. For the alphabet $\Sigma = \{\texttt{a}, \texttt{b}\}$, construct a DFA for the language $\{ w \in \Sigma^* \mid w$ contains the same number of instances of the substring `ab` and the substring `ba` $\}$. Note that substrings are allowed to overlap, so `aba` $\in L$ and `babab` $\in L$.

iii. For the alphabet $\Sigma = \{\texttt{a}, \texttt{b}, \texttt{c}, \ldots, \texttt{z}\}$, construct a DFA for the language $\{ w \in \Sigma^* \mid w$ contains the word "`cocoa`" as a substring $\}$. As a shorthand, you can specify multiple letters in a transition by using set operations on $\Sigma$ (for example, $\Sigma - \{\texttt{a}, \texttt{b}\}$) [*]

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 $\Sigma = \{\texttt{y}, \texttt{d}\}$. A string in $\Sigma^*$ 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 $L = \{ w \in \Sigma^* \mid w$ 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 $\Sigma = \{\texttt{a}, \texttt{b}, \texttt{c}\}$, construct an NFA for the language $\{ w \in \Sigma^* \mid w$ ends in `a`, `bb`, or `ccc` $\}$.

ii. For the alphabet $\Sigma = \{\texttt{a}, \texttt{b}, \texttt{c}, \texttt{d}, \texttt{e}\}$, construct an NFA for the language $\{ w \in \Sigma^* \mid$ the last character of $w$ appears nowhere else in $w$, and $|w| \geq 1 \}$.

iii. For the alphabet $\Sigma = \{\texttt{a}, \texttt{b}\}$, construct an NFA for the language $\{ w \in \Sigma^* \mid w$ contains at least two `b`'s with exactly five characters between them $\}$. For example, $\underline{\texttt{b}}\texttt{aaaaa}\underline{\texttt{b}}$ is in the language, as is $\texttt{aabaa}\underline{\texttt{b}}\texttt{aaabb}\underline{\texttt{b}}$ and $\texttt{a}\underline{\texttt{b}}\texttt{bbbba}\underline{\texttt{b}}\texttt{aaaaaaab}$, but `bbbbb` is not, nor are `bbbab` 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: 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 $\Sigma = \{\texttt{a}, \texttt{b}\}$ and let $L = \{ w \in \Sigma^* \mid w$ does not contain `ba` as a substring $\}$. Write a regular expression for $L$.

ii. Let $\Sigma = \{\texttt{a}, \texttt{b}\}$ and let $L = \{ w \in \Sigma^* \mid w$ does not contain `bb` 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 $\Sigma = \{\texttt{y}, \texttt{d}\}$ and let $L = \{ w \in \Sigma^* \mid w$ 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, `ddydyy` $\in L$. However, `yyyyddd` $\notin L$, since halfway through your walk you are three steps ahead of your dog; `ddyd` $\notin L$, because your dog ends up two steps ahead of you; and `ddyddyyy` $\notin L$, because at one point in your walk your dog is three steps ahead of you. Write a regular expression for $L$.

iv. Let $\Sigma = \{\texttt{a}, \texttt{b}\}$ and let $L = \{ w \in \Sigma^* \mid w \neq \texttt{ab} \}$. Write a regular expression for $L$.

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

Let $\Sigma$ be an alphabet. Give a brief English description of the set $\wp(\Sigma^*)$. Briefly justify your answer.

## Problem Five: Cardinalities and Concatenations (2 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 $\{\varepsilon\}$).

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

## Problem Six: 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 $|\overline{L}|$ is a natural number.

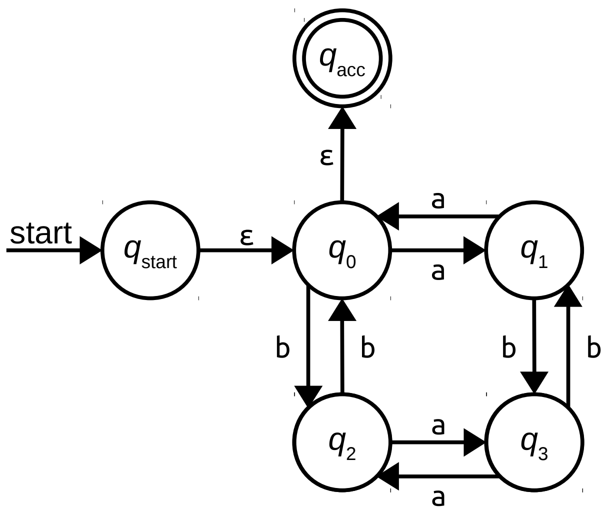
i. Prove that any finite language is regular.

ii. Prove that any cofinite language is regular.

## Problem Seven: 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 Σ = {a, b} and let $L = \{ w \in \Sigma^* \mid w$ has an even number of a's and an even number of b'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 $L$?

## Problem Eight: Derivatives of Regular Languages (4 Points)

Let $L$ be a language over Σ. For any a ∈ Σ, 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 $L = \{ \varepsilon$, a, aabb, aaab, ba, bb }. Then $\partial_a L = \{ \varepsilon$, 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.

i. Let $L$ be a regular language over Σ and let a ∈ Σ. Describe a construction that transforms a DFA for $L$ into a DFA for $\partial_a L$.

ii. 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 Nine: 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 $\mathscr{L}(N) = L$.
2. Add in a new start state $q_{\text{start}}$.
3. Add an ε-transition from $q_{\text{start}}$ to the start state of $N$.
4. Add ε-transitions from each accepting state of $N$ to $q_{\text{start}}$.
5. Make $q_{\text{start}}$ an accepting state.

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

1. Begin with an NFA $N$ where $\mathscr{L}(N) = L$.
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 $L^*$.

## Extra Credit Problem: Semilinear Languages (1 Point Extra Credit)

This week's extra credit problem might has a bit more new notation in it than usual, but it's a really cool problem. If you have some extra time to work through this problem, I think you'll find that the payoff is huge and that you'll have a *much* deeper understanding of the regular languages.

A ***linear set*** is a set of natural numbers of the form $\{ an + b \mid n \in \mathbb{N} \}$ where $a$ and $b$ are natural numbers. For example, the set $\{ 7, 12, 17, 22, 27, \dots \}$ is a linear set because it is equal to $\{ 5n + 7 \mid n \in \mathbb{N} \}$. Additionally, the set $\{137\}$ is a linear set because it's equal to $\{ 0n + 137 \mid n \in \mathbb{N} \}$.

We'll say that a set $S$ is a ***semilinear set*** if $S$ is the union of finitely many linear sets. As an example, the set $\{ 5n + 7 \mid n \in \mathbb{N} \} \cup \{ 3n + 0 \mid n \in \mathbb{N} \} \cup \{ 0n + 11 \mid n \in \mathbb{N} \}$ is a semilinear set. The empty set technically counts as a semilinear set, since it's the union of zero linear sets.

Finally, if $L$ is a language, define $\Psi(L)$ to be the set

$$\Psi(L) = \{ n \in \mathbb{N} \mid \text{there is a string in } L \text{ of length } n \}$$

In other words, $\Psi(L)$ is the set of all numbers that are the lengths of strings in $L$. As examples, we have that $\Psi(\{\varepsilon, \texttt{baa}, \texttt{abba}, \texttt{babba}\}) = \{0, 3, 4, 5\}$, that $\Psi(\Sigma^*) = \mathbb{N}$, and that $\Psi(\texttt{(ab)*}) = \{ n \in \mathbb{N} \mid n \text{ is even} \}$.

Let $\Sigma = \{\texttt{a}\}$ be an alphabet consisting of a single character and let $L$ be an arbitrary language over $\Sigma$. Prove that $L$ is regular if and only if $\Psi(L)$ is semilinear. *(Hint: This is a great time to use the fact that every regular language can be expressed as a regular expression.)*