Problem Set 5

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

In any question that asks for a proof, you **must** provide a rigorous mathematical proof. You cannot draw a picture or argue by intuition. You should, at the very least, state what type of proof you are using, and (if proceeding by contradiction, contrapositive, or induction) state exactly what it is that you are trying to show. If we specify that a proof must be done a certain way, you must use that particular proof technique; otherwise you may prove the result however you wish.

As always, please feel free to drop by office hours or send us emails if you have any questions. We'd be happy to help out.

This problem set has 125 possible points. It is weighted at 7% of your total grade. The earlier questions serve as a warm-up for the later problems, so do be aware that the difficulty of the problems does increase over the course of this problem set.

Good luck, and have fun!

Due Friday, February 15th at 12:50PM

Problem One: Constructing DFAs (24 Points)

For each of the following languages over the indicated alphabets, construct a DFA that accepts precisely those 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 https://www.stanford.edu/class/cs103/cgi-bin/nfa/nfa.php. 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 make a note of it in your problem set submission so that we know to look online for your answers.

i. For the alphabet $\Sigma = \{\texttt{0}, \texttt{1}, \texttt{2}\}$, construct a DFA for the language $L = \{ w \in \Sigma^* \mid w$ contains exactly two `2`s. $\}$

ii. For the alphabet $\Sigma = \{\texttt{0}, \texttt{1}\}$, construct a DFA for the language $L = \{ w \in \Sigma^* \mid w$ contains the same number of instances of the substring `01` and the substring `10` $\}$. Note that substrings are allowed to overlap, so `010` $\in L$ and `10101` $\in L$.

iii. For the alphabet $\Sigma = \{\texttt{a}, \texttt{b}, \texttt{c}, \ldots, \texttt{z}\}$, construct a DFA for the language $L = \{ w \in \Sigma^* \mid w$ contains the word "`cocoa`" as a substring $\}$. Remember that as a shorthand, you can specify multiple letters in a transition by using set operations on Σ (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 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 $L = \{ w \in \Sigma^* \mid w$ 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 (20 Points)

For each of the following languages over the indicated alphabets, construct an NFA that accepts precisely those 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. **As in Problem One, we recommend designing, testing, and submitting your automata using our online system.**

i. For the alphabet $\Sigma = \{\texttt{0}, \texttt{1}, \texttt{2}\}$, construct an NFA for the language $\{ w \in \Sigma^* \mid w$ ends in `0`, `11`, or `222`. $\}$

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 the string, and $|w| \geq 1 \}$

iii. For the alphabet $\Sigma = \{\texttt{0}, \texttt{1}\}$, construct an NFA for the language $\{ w \in \Sigma^* \mid w$ contains at least two `1`'s with exactly five characters between them. $\}$ For example, `1`000001 is in the language, as is `00`1`00110`1`00 and `0`1`1111101`00000001, but `11111` is not, nor are `11101` or `000101`.

---

* 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 (20 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. It's available online at https://www.stanford.edu/class/cs103/cgi-bin/simpleregex/edit.php. 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 make a note of it in your problem set submission so that we know to look online for your answers.

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 $L = \{ w \in \Sigma^* \mid w$ contains an even number of **a**'s $\}$. Write a regular expression for L.

iii. Suppose that 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, **YYDDDDYY** $\in L$, 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, and similarly **DDYDYY** $\in L$. However, **YYYDDD** $\notin L$, 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 partway through 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 $L = \{ w \in \Sigma^* \mid w \neq$ **ab** $\}$. Write a regular expression for L.

Problem Four: Finite and Cofinite Languages (16 Points)

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

i. Prove that any finite language is regular. *(Hint: Use induction.)*

ii. Prove that any cofinite language is regular.

We will cover the material necessary to solve these next two problems in Monday's lecture.

Problem Five: The Complexity of Addition (20 Points)

This problem explores the question

How hard is it to add two numbers?

Suppose that we want to check whether $x + y = z$, where x, y, and z are all natural numbers. If we want to phrase this as a problem as a question of strings and languages, we will need to find some way to standardize our notation. In this problem, we will be using the ***unary number system***, a number system in which the number n is represented by writing out n `1`'s. For example, the number 5 would be written as `11111`, the number 7 as `1111111`, and the number 12 as `111111111111`. Given the alphabet $\Sigma = \{$`1`, `+`, `=`$\}$, we can encode $x + y = z$ by writing out x, y, and z in unary. For example:

4 + 3 = 7 would be encoded as `111+1111=1111111`

7 + 1 = 8 would be encoded as `1111111+1=11111111`

0 + 1 = 1 would be encoded as `+1=1`

Consider the language $ADD = \{ \texttt{1}^m\texttt{+1}^n\texttt{=1}^{m+n} \mid m, n \in \mathbb{N} \}$. That is, *ADD* consists of strings encoding two unary numbers and their sum. Prove that *ADD* is not a regular language.

Problem Six: The Complexity of String Searching (20 Points)

The problem explores the question

How hard is it to search a string for a substring?

A common task in computer programming is to search a string to see if some other string appears as a substring. This task arises in computational biology (searching an organism's genome for some particular DNA sequence), information storage (finding all copies of some phrase in the full text of a book), and in spam filtering (searching for some key words in an email).

More formally, we can define the substring search problem as follows. The **string search problem** is given a string to search for (called the **pattern**) and a string in which the search should be conducted (called the **text**), to determine whether the pattern appears in the text. To encode this as a language problem, let $\Sigma = \{$`0`, `1`, `?`$\}$. We can then encode instances of the string search problem as the string `pattern?text`. For example:

"Does `0110` appear in `1110110` ?" would be encoded as `0110?1110110`

"Does `11` appear in `0001` ?" would be encoded as `11?0001`

"Does ε appear in `1100` ?" would be encoded as `?1100`

Let the language $SEARCH = \{ p\texttt{?}t \mid p, t \in \{\texttt{0}, \texttt{1}\}^* \text{ and } p \text{ is a substring of } t \}$. Prove that *SEARCH* is not regular, which means that no DFA, NFA, or regular expression is powerful enough to describe *SEARCH*.

Problem Seven: Course Feedback (5 Points)

We want this course to be as good as it can be, and we'd really appreciate your feedback on how we're doing. For a free five points, please answer the following questions. We'll give you full credit no matter what you write (as long as you write something!), but we'd appreciate it if you're honest about how we're doing.

i. How hard did you find this problem set? How long did it take you to finish? Does that seem unreasonably difficult or time-consuming for a five-unit class?

ii. Did you attend the midterm review session? If so, did you find it useful?

iii. How is the pace of this course so far? Too slow? Too fast? Just right?

iv. Is there anything in particular we could do better? Is there anything in particular that you think we're doing well?

Submission instructions

There are three ways to submit this assignment:

1. Hand in a physical copy of your answers at the start of class. This is probably the easiest way to submit if you are on campus.
2. Submit a physical copy of your answers in the filing cabinet in the open space near the handout hangout in the Gates building. If you haven't been there before, it's right inside the entrance labeled "Stanford Engineering Venture Fund Laboratories." There will be a clearly-labeled filing cabinet where you can submit your solutions.
3. Send an email with an electronic copy of your answers to the submission mailing list (cs103-win1213-submissions@lists.stanford.edu) with the string "[PS5]" somewhere in the subject line. You should receive automatic confirmation of your submission. If you submit electronically, please submit your assignment as a single PDF if at all possible. Sending multiple files makes it harder to print out and grade your submission.

Extra Credit Problem: Splitting Regular Languages (5 Points)

Let L be an arbitrary infinite regular language. Prove that there exist languages L_1 and L_2 such that

1. $L_1 \cup L_2 = L$,
2. $L_1 \cap L_2 = \varnothing$,
3. L_1 and L_2 are infinite, and
4. L_1 and L_2 are regular.

This shows that it is always possible to split an infinite regular language into two other infinite regular languages. *(Hint: Use the pumping lemma, along with the fact that regular languages are closed under subtraction. That is, if L_1 and L_2 are regular, then $L_1 - L_2$ is regular.)*