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 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.

Good luck, and have fun!

Due Monday, November 4th at 2:15PM

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/edit.php. We strongly recommend this tool, as it makes it easy to design, test, 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 $\}$. 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 and does not have to have the fewest number of states possible. **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, `1000001` is in the language, as is `0010011001` and `0111110100000001`, 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 (24 Points)

> *Depending on how quickly we cover material on Friday, we might not explore regular expressions by the time this problem set goes out. If not, we will cover the material necessary to solve this problem on Monday.*

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 $L = \{ w \in \Sigma^* \mid w$ 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, **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, the string **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 $L = \{ w \in \Sigma^* \mid w \neq$ **ab** $\}$. Write a regular expression for L.

Problem Four: Finite and Cofinite Languages (20 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.

Problem Five: Testing Universality (20 Points)

Suppose you have a magic machine (an *oracle*) that takes as input two DFAs over the same alphabet Σ and returns whether every string is accepted by exactly one of the two DFAs. In other words, given two DFAs D_1 and D_2, the oracle reports whether $\mathscr{L}(D_1) = \Sigma^* - \mathscr{L}(D_2)$. You have no insight into how this oracle operates; from your perspective, it takes in two DFAs and then magically produces an answer. You cannot make any assumptions about how it works. What else could you do with this machine?

i. Suppose you have a DFA D over an alphabet Σ and want to determine whether $\mathscr{L}(D) = \Sigma^*$. Describe, in plain English, a procedure that uses the oracle to answer this question.

ii. Formally prove that your procedure from (i) is correct by proving the following: your procedure reports that $\mathscr{L}(D) = \Sigma^*$ iff it's actually the case that $\mathscr{L}(D) = \Sigma^*$. That is, prove that if $\mathscr{L}(D) = \Sigma^*$, then your procedure returns true and that if $\mathscr{L}(D) \neq \Sigma^*$, then your procedure returns false.

Problem Six: Why the Extra State? (12 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_{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 $\mathscr{L}(N) = L$.
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^*.

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 online survey questions for this problem set, which are available at https://docs.google.com/forms/d/1QUhvPyVbgei4mXIH9p6O3cvmbI-foabHTB6yKz0HyEQ/viewform.

Extra Credit Problem: Multiples of Seven (5 Points Extra Credit)

Let Σ = {`0`, `1`, `2`, `3`, `4`, `5`, `6`, `7`, `8`, `9` } and let *MULTIPLE7* be the language over Σ defined as follows:

$$MULTIPLE7 = \{ w \in \Sigma^* \mid w \text{ is a base-10 representation of a multiple of 7 } \}$$

For example, `7` ∈ *MULTIPLE7*, `959` ∈ *MULTIPLE7*, `294` ∈ *MULTIPLE7*, and `343` ∈ *MULTIPLE7*. However, ε ∉ *MULTIPLE7*, `1` ∉ *MULTIPLE7*, and `7897987` ∉ *MULTIPLE7*.

Design a DFA for *MULTIPLE7*.