Problem Set 6

This sixth 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, February 19 at the start of class.
There is no checkpoint problem.

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.

Please use our online tool to design, test, and submit your answers to this problem. Handwritten or typed solutions will not be accepted. To use the tool, visit the CS103 website and click the "DFA/NFA Editor" link under the "Resources" header. When you submit the rest of the problems for this problem set, please make a note in your submission of which member of your group submitted so that we can match the problem set to the submitted answers.

As a note, we will use an autograder to check your answers for this problem, so be sure to test your solutions before you submit!

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 cs. $\}$

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 $\}$.[*]

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 (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. **Please use our online system to design, test, and submit your automata**; see above for details. As before, **please test your submissions thoroughly!**

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.

Please use our online tool to design, test, and submit your regular expressions. Typed or handwritten solutions will not be accepted. To use it, visit the CS103 website and click the "Regex Editor" link under the "Resources" header. As before, make a note in your GradeScope submission of which team member submitted your answers to this question so that we know where to look. Also, as a reminder, please test your submissions thoroughly, since we'll be grading them with an autograder.

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 `yyddddyy` 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, $\texttt{ddydyy} \in L$. However, $\texttt{yyyyddd} \notin L$, since halfway through your walk you are three steps ahead of your dog; $\texttt{ddyd} \notin L$, because your dog ends up two steps ahead of you; and $\texttt{ddyddyyy} \notin L$, 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 Σ be an alphabet. Give a short 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 Five: 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.

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

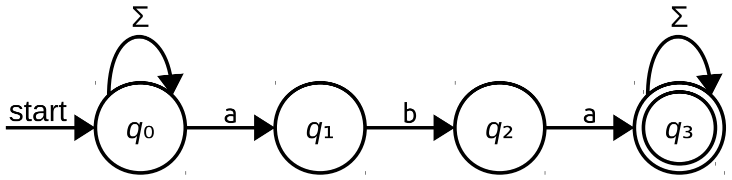
Problem Six: Cardinalities and Concatenations (3 Points)

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 $\{\varepsilon\}$).

Let L be an an arbitrary finite language (that is, a language containing a finite number of strings). Prove or disprove: for every natural number $k \geq 1$, we have $|L^k| = |L|^k$.

Problem Seven: 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.

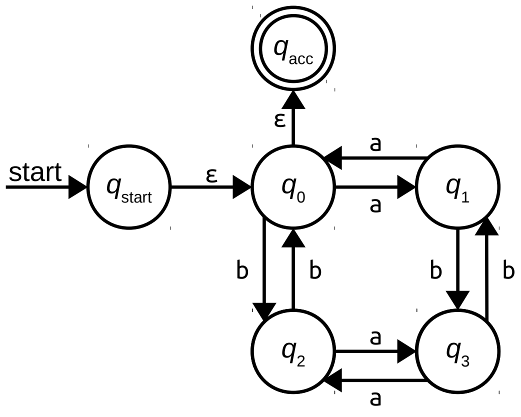
i. Using the subset construction, convert this NFA to a DFA. Do ***not*** just find a DFA that has the same language as this NFA; we want you to use the subset construction algorithm detailed in lecture. For simplicity, please handwrite or type up the resulting DFA as a transition table.

ii. 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 Eight: 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 $L = \{ w \in \Sigma^* \mid w$ 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 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_{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.
6. Make every state besides q_{start} a rejecting 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.
4. Make every other state of N a rejecting 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: Why Finite? (1 Point Extra Credit)

The term "finite" in finite automata refers to the fact that each automaton can only have finitely many states. It turns out there's a good reason for that.

We'll say that a ***deterministic infinite automaton***, or ***DIA***, is a generalization of a DFA in which the automaton has infinitely many different states. Since DIAs have infinitely many states, they can't actually be built, and so are mostly an object of purely theoretical study.

Prove that if L is an arbitrary language over an alphabet Σ, then there is a DIA that accepts L (that is, the DIA accepts every string in L and rejects every string not in L.)