Homework 1 – Due Thursday, September 15, at 11:59 PM

Reminder Collaboration is permitted, but you must write the solutions *by yourself without assistance*, and be ready to explain them orally to the course staff if asked. You must also identify your collaborators and write "Collaborators: none" if you worked by yourself. Getting solutions from outside sources such as the Web or students not enrolled in the class is strictly forbidden.

Problems There are 5 required problems and one bonus problem.

1. For each of the following languages, (i) give a plain English description of the language, and (ii) give two examples of strings in the language and two examples of strings that are in Σ^* but outside the language.

   (a) $L_1 = \{w \in \{0,1\}^* \mid w = w^R\}$

   (b) $L_2 = \{x\mathsf{aa}y \mid x, y \in \{\mathsf{a},\mathsf{b}\}^*\} \cup \overline{\{x\mathsf{bb}y \mid x, y \in \{\mathsf{a},\mathsf{b}\}^*\}}$

   (c) $L_3 = \{x\#y\#z \mid x, y, z \in \{0,1\}^* \text{ and } x + y = z\}$. (Here, you should view x, y, z as the binary representations of non-negative integers, with the most significant bit on the left.)

2. For each of the following languages, (i) describe the language using set-builder notation and union/intersection/complement/reverse/concatenation operations (the notation used in Problem 1), and (ii) give two examples of strings in the language and two examples of strings that are in Σ^* but outside the language.

   (a) L_4 = the set of all strings over alphabet $\{\mathsf{a},\mathsf{b},\mathsf{c}\}$ that have length at least 3 and have b as their third symbol.

   (b) L_5 = the set of all strings over alphabet $\{0,1\}$ that either start with 0 and have odd length, or start with 1 and have even length.

   (c) L_6 = the set of all strings over alphabet $\{\mathsf{a},\mathsf{b}\}$ that contain neither aaab nor baaa as substrings.

3. Which of the following statements are true or false, for all alphabets Σ? For each, provide either a proof or a counterexample.

   (a) For all strings $x, y, z \in \Sigma^*$, we have $|x \circ (yz)^R| = |x| + |y| + |z|$. (Recall that $\circ$ denotes concatenation.)

   (b) For all languages $L_1, L_2 \subseteq \Sigma^*$, we have $(L_1 \circ L_2)^R = L_2^R \circ L_1^R$.

   (c) For all languages $L \subseteq \Sigma^*$, we have $L \circ \{\varepsilon\} = L \circ \emptyset$.

   (d) For all languages $L_1, L_2, L_3 \subseteq \Sigma^*$, we have $L_1 \circ (L_2 \cap L_3) = (L_1 \circ L_2) \cap (L_1 \circ L_3)$.

4. Let N denote the set of "alphanumeric" characters (i.e., upper and lower case English alphabet symbols together with the numbers 0-9). Let P denote the set of "printable" characters, which includes N together with other symbols such as $-, +, \$, \ldots$, but crucially, not the @ symbol. A well-formed email address is a nonempty string of printable characters, followed by an @ symbol, followed by two or more nonempty strings of alphanumeric characters that are separated by .'s.

For example, cs.332@computer.science.boston.university.edu and i$pi11ed_+heM!lk@d0.n0t.cry are well-formed email addresses, but rick@.roll.edu is not, because the substring between @ and the next . is empty. Let L = the set of all well-formed email addresses.

(a) Describe the language L using set-builder notation. It's fine if your description includes one or more "..."'s; just make sure the person reading your submission can tell the difference between those dots and the .'s in an email address.

(b) Give pseudocode for a program that recognizes the language L. That is, your program should take as input a string x and return "accept" if and only if x is a well-formed email address. Your may write things like "$a \in N$" to test if a character is in the set N.

5. Let L be a language over an alphabet Σ and let $a \in \Sigma$ be an alphabet symbol. Define the language $\partial_a L$ (read: "the derivative of L with respect to a") by $\partial_a L = \{x \mid ax \in L\}$.

(a) Let $\Sigma = \{0, 1\}$. Let $A = \{\varepsilon, 0, 01, 11, 011\}$. Let $B = \{x \in \Sigma^* \mid x \text{ starts with } 001\}$. What are $\partial_0 A$ and $\partial_0 B$?

(b) Use the pigeonhole principle to prove that if L is a *finite* language, then for every alphabet symbol $a \in \Sigma$, we have $|\partial_a L| \leq |L|$.

6. **Bonus Problem.** In this problem, you'll explore how to formally define and analyze properties of strings. Let Σ be an alphabet. A *string* over alphabet Σ is defined recursively as follows. It is either the empty string ε (base case) or takes the form ax where $a \in \Sigma$ and x is itself a string (recursive case). The *length* of the string x can then be defined as follows:

$$|x| = \begin{cases} 0 & \text{if } x = \varepsilon \\ 1 + |z| & \text{if } x = az \text{ for } a \in \Sigma \text{ and } z \in \Sigma^*. \end{cases}$$

Similarly, the concatenation of two strings x, y can be defined as

$$xy = \begin{cases} y & \text{if } x = \varepsilon \\ a(zy) & \text{if } x = az \text{ for } a \in \Sigma \text{ and } z \in \Sigma^*. \end{cases}$$

These definitions let us give inductive proofs of properties of strings. For instance, consider the following claim, which says that the length of the concatenation of two strings is the sum of the lengths of those strings.

Claim. For any two strings x, y, we have $|xy| = |x| + |y|$.

To prove this, let x and y be arbitrary strings. We will prove this by induction on the length $n = |x|$. As our base case, suppose $n = 0$. Then $x = \varepsilon$, so

$$\begin{aligned} |xy| &= |\varepsilon y| && \text{(assumption on } x\text{)} \\ &= |y| && \text{(definition of concatenation)} \\ &= |\varepsilon| + |y| && \text{(definition of length)} \\ &= |x| + |y| && \text{(assumption on } x\text{)} \end{aligned}$$

as we wanted. Now assume as our inductive hypothesis that the claim is true for length n; we want to show it is true for length $n+1$. In this case, we have $x = az$ for some string z of length n. So

$$\begin{aligned} |xy| &= |a(zy)| && \text{(definition of concatenation)} \\ &= 1 + |zy| && \text{(definition of length)} \\ &= 1 + |z| + |y| && \text{(inductive hypothesis)} \\ &= |x| + |y| && \text{(definition of length).} \end{aligned}$$

(a) Given a string $x \in \{0,1\}^*$, let $\text{sort}(x)$ denote the string obtained by sorting the characters of x so that all 0's appear before all 1's. For example, $\text{sort}(10110) = 00111$. Give a recursive definition of the sort function along the lines of what we did with length above.

(b) Give an inductive proof that $|\text{sort}(x)| = |x|$ for every string $x \in \{0,1\}^*$.

(c) Prove that $\text{sort}(\text{sort}(x)) = \text{sort}(x)$ for every string $x \in \{0,1\}^*$. Hint: Instead of trying to prove this directly by induction, it might be useful to introduce some auxiliary recursively-defined functions and prove statements about those.