Student ID: ______________________________

Collaborators: ______________________________

CS181 Winter 2018 – Problem Set 2

Due Tuesday, February 6, 11:59 pm

- Please write your student ID **and the names of anyone you collaborated with** in the spaces provided and attach this sheet to the front of your solutions. **Please do not include your name anywhere since the homework will be blind graded.**
- An extra credit of **5%** will be granted to solutions written using LaTeX. Here is one place where you can create LaTeXdocuments for free: `https://www.sharelatex.com/`. The link also has tutorials to get you started. There are several other editors you can use.
- If you are writing solutions by hand, please write your answers in a neat and readable hand-writing.
- Always explain your answers. When a proof is requested, you should provide a rigorous proof.
- 20% of the points will be given if your answer is "I don't know". However, if instead of writing "I don't know" you write things that do not make any sense, no points will be given.
- The homework is expected to take anywhere between 8 to 14 hours. You are advised to start early.
- Submit your homework online on the course webpage on CCLE. You can also hand it in at the end of any class before the deadline.

Note: *All questions in the problem sets are challenging; you should not expect to know how to answer any question before trying to come up with innovative ideas and insights to tackle the question. If you want to do some practice problems before trying the questions on the problem set, we suggest trying problems 1.17 and 1.23 from the book. Do not turn in solutions to problems from the book.*

1(a).

Theorem 1a. If the language L1 is not regular and L2 is any language then the languages shuffle(L1,L2) and shuffle(L1,∼L2) cannot both be regular.

Proof. For this problem, we suppose that both shuffle(L1,L2) and shuffle(L1,∼L2) are regular. That means that there is some DFA that accepts shuffle(L1,L2) and one that accepts shuffle(L1,∼L2). Now we focus on the differences between the two DFAs. Notice that L2 and ∼L2 are complements. In class, we briefly mentioned that to build a DFA that recognizes the complement of a language, you invert the accepting states. The five tuples of L2 and ∼L2 are as follows:

$$L2 = (Q, \Sigma, \delta, q_0, F) \tag{1}$$

$$\sim L2 = (Q, \Sigma, \delta, q_0, \sim F) \tag{2}$$

However since the final states are opposites because of the two accepting states being complements, it is impossible for both L2 and ∼L2 on the same input. This is because the δ transitions remain identical to each other. Thus for any input, both DFAs end in the same state, the only difference being the fact that one ends in an accept state and the other in a non-accepting state. As a result, we can conclude that shuffle(L1,L2) and shuffle(L1,∼L2) cannot both be regular. □

1(b).

Theorem 1b. If L1 and L2 are regular languages then shuffle(L1,L2) is regular.

Proof. To prove this theorem, I will build an NFA that accepts shuffle(L1,L2). To do this, we need to build it using two DFAs. One DFA, call it M_1 will recognize L1. The other DFA, call it M_2 will recognize L2. The idea is to alternate between the two DFAs for each new input. In this way, we can achieve our goal of recognizing a shuffled input. Assuming that Σ is {0,1}, our five-tuples are as follows:

$$M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1) \tag{3}$$

$$M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2) \tag{4}$$

With our DFAs established, we can now build our NFA N. Our five-tuple for N is:

$$N = (Q', \Sigma, \delta', q', F') \tag{5}$$

The states in the NFA consists of the states in both M_1 and M_2 along with a new start state to be defined later.

$$Q' = Q_1 \times Q_2 \cup q' \tag{6}$$

Our delta transitions involve an additional input that will keep track of what character input will come next. This input will help us alternate between the two DFAs. For example if the next character is expected to be from L1, then only the part of the state from Q_1 will change. Our δ transitions are as follows:

$$\delta'(q', \epsilon) = (q_1, q_2, x \in L1) \tag{7}$$

The equation above is the δ transition for the start state of the NFA, q' given an input of ϵ. It transitions to the start states of both DFAs and signifies with the third parameter that it expects the next character to be recognized by L1.

$$\delta'((s1, s2, x \in L1), a) = \delta'((\delta'((s1, s2, x), a), s2, y \in L2), a) \tag{8}$$

$$\delta'((s1, s2, y), a) = \delta'((s1, \delta'((s1, s2, y), a), x), a) \tag{9}$$

The start state of the NFA has ϵ transitions to both of the DFA start states and a property stating the next input type.

$$q' = (q_0, x) \tag{10}$$

The NFA will only accept if both DFAs are in accept states and the next input type is of L1.

$$F' = F_1 \times F_2 \times \{x \in L1\} \tag{11}$$

To prove the construction, we use the fact that M_1 accepts L1 and M_2 accepts L2 if and only if N accepts shuffle(L1, L2). □

2(a).

Theorem 2a. Prove that the language L2 = b* ∪ L1 satises the conditions of the Pumping Lemma.

Proof. To prove this theorem, I will prove each component of L2 individually. If I prove b* and L1 both satisfy the Pumping Lemma (PL), then the union of the two will also satisfy the lemma, since the union of the two languages consists of the strings from the two languages.

To prove b* satisfies PL, we assume that b* is regular so it has to satisfy the PL's conditions. By PL, ∃n ∈ ℕ. Next, we need x to be at least of length n.

$$x = b^{\mathrm{n}} \tag{12}$$

By PL, ∃a,b,c such that x = abc. Since $\|ab\| \leq$ n, we have the following:

$$a = b^{\alpha}, b = b^{\beta}, \beta \geq 1, c = b^{\text{n-}\alpha\text{-}\beta} \tag{13}$$

By property 1 of PL and using i = 0, we get:

$$b^{\alpha} b^{\text{n-}\alpha\text{-}\beta} = b^{\text{n-}\beta} \in b^{*} \tag{14}$$

Choosing values of i higher than 0 will result in the same conclusion. b* satisfies PL.

I will prove $a^{i}b^{p}$ using the same method. Again I assume that $a^{i}b^{p}$ satisfies PL. I set x as follows:

$$x = a^{n}b^{2} \tag{15}$$

Now we have x=abc and I set the three variables as follows:

$$a = a^{\alpha}, b = a^{\beta}, \beta \geq 1, c = a^{n-\alpha-\beta}b^{2} \tag{16}$$

You might be tempted to use the following abc:

$$a = a^{n}b^{\alpha}, b = b^{\beta}, c = b^{n-\alpha-\beta} \tag{17}$$

However this setup violates property 2 of PL which states that $\|ab\| \leq$ n.
Using our correct setup, we apply property 1 of PL and use i = 0 and we get:

$$a^{\alpha}a^{n-\alpha-\beta}b^{2} = a^{n-\beta}b^{2} \in L1 \tag{18}$$

Using i > 0 will still yield the same conclusion. Because both individual components of L2 satisfy PL, then the union of such components will satisfy PL.

□

2(b).

Theorem 2b. Prove the general form of the Pumping Lemma.

Proof. To prove this version of the PL, we can use a modified version of the proof of the old PL. Recall the FSM that we built in class to prove the old PL. We will be using that again here. Suppose our pumping length is p. Let the start state equal q_0 and let q_1, q_2 ..., q_p be the states in the FSM. Let us assume that an input xyz is of length p. That means that the set of visited states to process the input will be of length p+1. By the pigeonhole principle, one of the states is repeated and there is a loop somewhere in the FSM. We can call the input string of the looped portion of the FSM y. Since the machine will match the string with or without y portion, our old PL is proven. To prove our new PL, we simply tack on some non-empty string to the front and back of our input string xyz, e.g. axyzb where a and b are non-empty strings. Now that we have a new input string for the FSM, notice the fact that the inputs axyzb and $axy^{i}zb$ will get the machine to the same state. Thus if the FSM accepts axyzb, then it must also accept $axy^{i}zb$. Thus the general form of the PL is proven. □

2(c).

Theorem 2c. Prove that the language L2 is not regular.

Proof. Suppose that L2 is regular. Then L2 will satisfy our new PL proven in 2b. Let d be the following:

$$d = a^{\mathrm{n}}b^{\mathrm{p}} \tag{19}$$

By PL, ∃xyz such that d = xyz. My setup is the following:

$$x = a^{\mathrm{n}}, y = b^{\mathrm{p}\text{-}\alpha}, z = b^{\alpha} \tag{20}$$

By PL, ∃abc such that y = abc (y is from xyz). My setup is the following:

$$a = b^{\beta}, b = b^{\gamma}, \gamma > 0, c = b^{\mathrm{p}\text{-}\alpha\text{-}\beta\text{-}\gamma} \tag{21}$$

Multiplying the b term by i per the PL, we get:

$$b^{\beta}b^{\mathrm{i}\gamma}b^{\mathrm{p}\text{-}\alpha\text{-}\beta\text{-}\gamma} = b^{\mathrm{p}\text{-}\alpha\text{-}\gamma\text{-}\mathrm{i}\gamma} = b^{\mathrm{p}\text{-}\alpha+(\mathrm{i}+1)\gamma} \tag{22}$$

Setting i = p+1, we get:

$$b^{\mathrm{p}\text{-}\alpha+(\mathrm{p}+1\text{-}1)\gamma} = b^{\mathrm{p}\text{-}\alpha+\mathrm{p}\gamma} \tag{23}$$

Combining the previous result with x and z from xyz, we get:

$$a^{\mathrm{n}}b^{\mathrm{p}\text{-}\alpha+\mathrm{p}\gamma}b^{\alpha} = a^{\mathrm{n}}b^{\mathrm{p}+\mathrm{p}\gamma} = a^{\mathrm{n}}b^{\mathrm{p}(1+\gamma)} \tag{24}$$

The result is a contradiction since p(1+γ) is not prime. Since PL is not satisfied, L2 is not regular.

□

3.

Theorem 3. If L is regular, then $\mathrm{L}_{\frac{1}{3}\text{-}\frac{1}{3}}$ need not be regular.

Proof. If L is regular, then L satisfies PL and ∃D where D is a DFA such that it accepts L. Since there is a DFA that accepts L, then the DFA recognizes xyz ∈ L. If you remove the y portion of the string xyz, resulting in the string xz, then it is not necessarily a given that xz ends in an accept state. Referring to the proof of PL, xyz and xz end in the same state if and only if y is the input string of the looped portion of the DFA.

For example take the string 000212121, which is $0^3 21^3 \in 0^*21^*$. We can use the following setup for the example string (taking into account that $\|x\| = \|y\| = \|z\|$) :

$$x = 000, y = 212, z = 121 \tag{25}$$

If we remove the y portion of the string, then this is the resulting string:

$$xz = 000121 \notin 0^{*}21^{*} \tag{26}$$

The language that could accept the string xz is 0*1*21*. Our 3 closure properties for regular languages are union, concatenation and star. However, we cannot get to 0*1*21* from 0*21* using our closure properties. Since we cannot use the closure properties in our example, it is not necessarily the case that $L_{\frac{1}{3}\text{-}\frac{1}{3}}$ can be reached from L via the closure properties and so it is not the case that $L_{\frac{1}{3}\text{-}\frac{1}{3}}$ must be regular if L is regular.

$\square$