CS 181: Formal Languages and Automata Theory	**Spring 2021**
Homework 5 Solution	
Assigned: 27 April	*Due: 3 May, 9:00pm PDT*

Note Submission deadline is now 9:00pm.

Problem 1

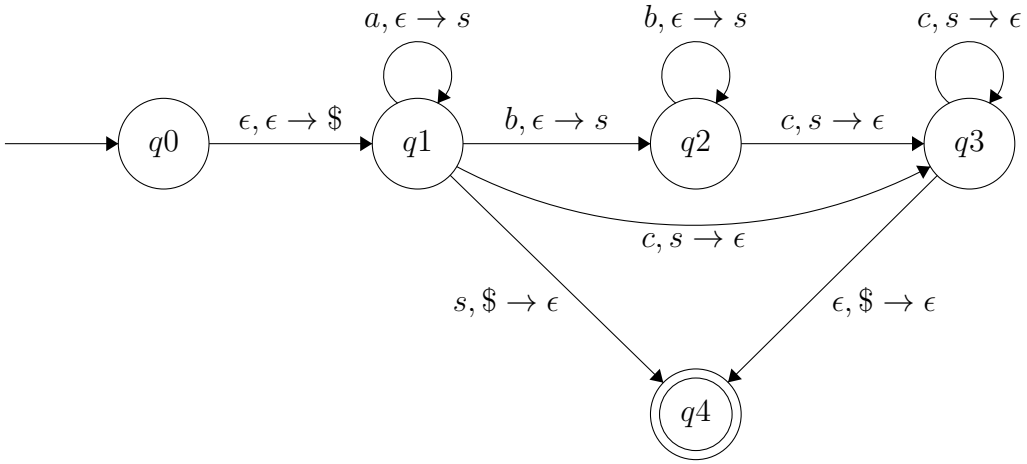
Let alphabet $\Sigma = \{$ a, b, c $\}$, and let:

$$L_{add} = \{a^i b^j c^k \mid i, j, k \geq 0 \text{ and } k = i + j\}$$

Show a PDA for this language. There are no other conditions for correctness other than what you see here, except, of course our general guidance in this class that your answer should be clear and avoid too much unnecessary complexity. Remember to include a brief description of how your PDA is designed to correctly recognize the language.

Solution:

The input alphabet $\Sigma = \{a, b, c\}$, the start state is $q0$, the set of accept states is $F = \{q4\}$. We define the stack alphabet $\Gamma = \{\$, s\}$ where the symbol s is used to count the number of a and b, and the symbol \$ is used to mark the bottom of the stack. The PDA state transition diagram is as follows:



Brief Explanation: If the input string is ϵ, the PDA will accept it by going through $q0 \to q1 \to q4$. The self-loop transition on $q1$ $a, \epsilon \to s$ is used to count the number of a's, the self-loop transition $b, \epsilon \to s$ is used to count the number of b's at state $q2$. Thereby, the number of s's in the stack equals the number of a's plus b's seen so far. Since the transition $c, s \to \epsilon$ will pop out the symbol s from the stack when reading a c, the \$ will be revealed on the top of the stack if the number of c's equals the number of a's and b's. Then the

PDA will reach $q4$ via $q3$ through the transition $\epsilon, \$ \rightarrow \epsilon$ when the empty stack marker, i.e., \$ is on the top of the stack. If there are not enough c's, the \$ will never be revealed, and it cannot reach the accepting state. If there are too many c's, then it will not be able to proceed reading input after the \$ is revealed because the pop is only allowed when the top of stack is s. Thus, it will be stuck in state $q3$ or $q4$ and unable to reach the end of the input string. Recall that if a PDA (or NFA or DFA) cannot reach the end of the input, then it is *never* an accepting computation.

Problem 2

Let $\Sigma = \{a, b, c\}$. Show using the Pumping Lemma for the Family of CFLs (and possibly other results) that the following language over Σ is not context free:

$$L_2 = \{a^i b^j c^k \mid i, j, k \geq 0 \text{ and } k = i + j \text{ and } i > j\}$$

Remember that your proof will be graded based on the four elements of a good proof using the CFL pumping lemma: appropriate choice of your string s; effective use of the constraints on the 5 substrings $uvxyz$ from the lemma; showing that you have considered every possible case for $s = uvxyz$; and clear, sound logic in each case.

Solution:

Suppose for contradiction that the language were a CFL. Let p be the number in the pumping lemma, and choose $s = a^{p+1}b^p c^{2p+1}$, which is in the language since $2p + 1 = (p + 1) + p$ and $(p + 1) > p$. Since $|s| > p$, we can write $s = uvxyz$ according to the lemma. And since each block of like symbols is of length at least p and $|vxy| \leq p$, we know that vxy cannot straddle all three blocks; so there are five cases to consider for where substring vxy could be in s:

Case 1: Suppose the pump-able parts of vxy straddle the ab boundary, meaning that v by itself or y by itself contains both types of symbols or that v contains at least one a and y contains at least one b. Note that these two sub-cases are not mutually exclusive. We consider these two sub-cases separately:
In the first sub-case where v or y straddles by itself, pumping up ($n > 1$) will cause $s' = uv^n xy^n z$ to contain a's and b's out of order, since there will be two consecutive copies of both v and y in s'. (Notice that we will use "n" for the pumping factor, rather than "i" to avoid confusion with the "i" in the definition of the language).
In the second sub-case, pumping up or down will increase or decrease (respectively) the number of a's and/or b's, which will change the value "$i + j$" in the definition of the language without changing k, contradicting the definition of the language.

Case 2: Suppose that vxy straddles the bc boundary. If v and/or y contain a symbol b, pumping up will increase the number of b's without increasing the number of a's and cause s' to violate the condition $i > j$ because they differ by exactly 1 in s. Otherwise, neither v nor y contains a b; so v and/or y must contain at least one symbol c. Then, pumping

up or down will change k without changing $i+j$, contradicting the definition of the language.

Note about cases 1 and 2: Depending on where v and y are, pumping up in some of these cases and sub-cases could also result in the symbols being out of order, but we already know that changing the numbers of symbols here is sufficient to derive our contradiction; so we do not need to account for that as part of our proof.

Cases 3,4,5: The remaining three cases are when vxy is entirely contained within a single block of the same symbol. In all three cases, pumping up or down will change one of the values i, j, or k in the definition of the language without changing the other two, contradicting the "$k = i + j$" condition.

By the way we have used "straddle" and "the pump-able parts straddle", this covers all possible ways that s could be parsed into $uvxyz$.

Since all cases of $s = uvxyz$ lead to contradictions, we conclude that the language cannot be a CFL.

Problem 3

Let A and B be two languages over the same alphabet, Σ. Assume that neither language contains ϵ. Then let the **perfect shuffle** of A and B be the language $\{\omega \mid \omega = a_1b_1...a_kb_k$ where $a_1...a_k \in A$ and $b_1...b_k \in B$ each $a_i, b_i \in \Sigma\}$.
Show that the class of finite state (aka regular) languages is closed under perfect shuffle by showing how to construct a DFA for the perfect shuffle of two FSLs.

You do not have to formally prove that your constructed finite automaton is correct. However, as always in this class, your response must include a brief explanation of how your answer correctly represents the solution.

Solution:

Suppose that A and B are finite state languages. Then we can assume there is a DFA for each language: $D_A = \{Q_A, \Sigma, \delta_A, q_A, F_A\}$ and $D_B = \{Q_B, \Sigma, \delta_B, q_B, F_B\}$ to represent language A and language B respectively. Then we construct a new DFA, M_{ps}, using the above mentioned DFAs to represent the perfect shuffle of A and B. Recall, a DFA is defined as a 5-tuple. Therefore, we need to define all 4 new elements of $M_{ps} = (Q_{ps}, \Sigma, \delta_{ps}, q_{ps}, F_{ps})$.
Here is the construction definition:

- $Q_{ps} = Q_A \times Q_B \times \{0, 1\}$
- For all $x \in \Sigma$, $a \in Q_A$ and $b \in Q_B$ and $pos \in \{0, 1\}$, define

$$\delta_{ps}((a, b, pos), x) = \begin{cases} (\delta_A(a, x), b, 1), & \text{if } pos = 0 \\ (a, \delta_B(b, x), 0), & \text{if } pos = 1 \end{cases}$$

- $q_{ps} = (q_A, q_B, 0)$
- $F_{ps} = F_A \times F_B \times \{0\}$

Brief explanation: We use the states of M_{ps} to keep track of the two machines D_A and D_B as we execute each one symbol-by-symbol. After we process one input symbol using machine D_A, we switch to machine D_B. We process one input symbol using machine D_B, and then we continue to switch back and forth. We use an additional marker added as a third component of the states of M_{ps} to indicate which machine we are currently using right now and to direct M_{ps} to correctly switch back and forth. The accepting states, F_{ps}, are defined so that the machine accepts only if both machines end in an accepting state and only if the last machine executed was D_B.