Max Shi
CS334
Professor Bhatt
October 24, 2019
I pledge my honor that I have abided by the Stevens Honor System.

1. Let the language be context free. Thus, it should satisfy the pumping lemma. If we let pumping length be p, let string s be $a^pb^pc^pd^p$. This $s \in L$, and is a valid candidate for the pumping lemma. Using the pumping lemma for CFGs, we can split this string into uvxyz, with pumping window comprised of vxy. Because $|vxy| <= p$, the pumping window can either remain enclosed in each separate character, or straddle the gap between two consecutive characters only, such as a and b or c and d. When the pumping window is enclosed in one character, pumping the string will result in extra instances of that string, but not the same extra instances of that associated string, as in a with c and b with d. If the window straddles the strings, pumping the string can either yield extra instances of one character (e.g. $a^pb^{p+i}c^pd^p$, $0<i<=p$), extra instances of two consecutive characters into the length of the two consecutive characters(e.g. $a^pb^{p+i}c^{p+j}d^p$, $0<i$, $0<j$, $i+j<=p$), substrings of two consecutive characters(e.g. $a^pb^p(b^ic^j)^+c^pd^p$, $0<i$, $0<j$, $i+j<=p$), or substrings and adding extra characters (e.g. $a^pb^p(b^ic^j)^+c^pc^kd^p$, $0<i$, $0<j$, $0<k$, $i+j+k<=p$). In each of these cases, the associated characters can never be pumped at the same time, and their lengths will not be the same. Thus, no matter where the window is, the string cannot be pumped and stay within the language. The pumping lemma is violated, and this language cannot be context free.
2. Let w_i be the current character the automata is reading. Let there be four states – "reading a", "reading b", "reading c", "reading d". Start with both stacks, stack A and stack B, with one element "$" in each stack. Start with state "reading A."
    a. State "reading A" – for every A read, push A onto stack A, and stay in this state.
        i. If a B is read, advance to state "reading B"
        ii. If a C is read, advance to state "reading C"
        iii. If a D is read, reject.
    b. State "reading B" – for every B read, push B onto stack B, and stay in this state.
        i. If a "C" is read, advance to state "reading C"
        ii. If a "D" is read, advance to state "reading D"
        iii. If anything else is read, reject.
    c. State "reading C" – For every C read, pop A from stack A.
        i. If there are no more A's and a $ is popped, reject.
        ii. If a D is read, advance to state "reading D"
        iii. If anything else is read, reject.
    d. State "reading D" – For every D read, pop B from stack B.
        i. If there are no more B's and a $ is popped, reject.
        ii. If any other character is read, reject.
        iii. If the input is empty and both stacks are empty, accept.
        iv. Otherwise, reject. (This catches non-popped A's from stack A)
3. a. L_{add} :
   S -> AC
   A-> aAb | ε

C -> bCc | ε

b. Let L_{mult} be language M. Assuming M is a regular language, it should fulfill the pumping lemma. Let s = $a^p b^{p^2} c^p$, so that s ∈ M. With pumping window |vxy| <= p, the pumping window can either be all in the a's, all in the b's, all in the c's, or straddle the a's and b's or b's and c's. In the cases where the pumping window is confined to one character, pumping this string would result in escaping the language, as the number of other characters has not changed. When the pumping window straddles two characters, for every a or c we add, we need to add p number of b's. Adding 0<i<p number of a's would change the middle expression from p^2 to (p+i)*p, which simplifies to p^2 + ip. However, when adding i number of a's, we can only add up to p-i number of b's to the string. Because p-i is at maximum p-1, where i is 1, the quantity p^2 + p-1 is strictly less than p^2 + p. Setting i any greater only increases the gap, e.g. setting i = (p-1) yields p^2 + 1 much less than p^2 + p^2 – p. This same logic applies to the other side with straddling b's and c's. Thus, pumping the string in any of these cases escapes the language. Finally, it is also possible that v or y = $a^i b^j$ or $b^i c^j$ where 0<i, 0<j, i+j<=p, however, pumping this would switch between a and b or b and c multiple times, escaping the language. Thus, the pumping lemma is violated and this language is not context free.

4.

a. Let there be two states, "reading A" and "reading B". Start with the empty stack with a $ pushed into it. Start with state "reading A."
   i. State "reading A" – for every "a" read, push "a" into the stack and stay in the same state.
      1. If a b is read, move to state "reading B."
   ii. State "reading B" – For every "b" read, pop an "a" from the stack.
      1. If an "a" is read, accept.
      2. If a "$" is popped from the stack, accept.
      3. If the input is empty, and an "a" is popped from the stack, accept
      4. If the input is empty and a "$" is popped from the stack, reject.

b. Let this be a non-deterministic PDA that starts at the start of the input. Let it begin with a "$" pushed into the stack. Let it pass over the input to make sure it has a "#", reject if it doesn't.
   Now, non-deterministically move the head to every other string in the input by stopping on each "#". The PDA should also have a clone which does not move. Repeat the following process for every string:
   i. Push the first string into the stack, by pushing each character until a "#" is read.
   ii. For every string following the first string, non-deterministically move the head of the input to each "#" in the rest of the input.
   iii. For every now non-deterministic PDA that is reading each "#" in the input after the first string pushed into the stack, read the next character.
      1. If the character is "a" or "b", pop the stack and compare the input with the popped character on the stack (Includes "$"). If they do not match, reject. Repeat this step by reading the next character and making the comparison.

2. If the character is a “#” or an empty character, check if the stack is empty. If it is empty, accept. If it is not empty, reject.

This non-deterministic PDA checks every string and then checks every other string after that string for the $x=x^R$ relationship, and therefore accepts the language.