(1) Let $L_1$ be the language described by the regular expression $(a \cup bbb \cup \epsilon)^*(b \cup aaa \cup \epsilon)^*$.
Let $L$ be the language \{\(w \in \{a,b\}^*: w = xx^{rev}$ with $x \in L_1\}\). (Note that $x^{rev}$ means the reverse of $x$.) Produce a pushdown automaton that recognizes the language $L$.

**Answer:** A PDA is given by the following state diagram:

From the start state a $ is pushed on the stack to mark the bottom. States 1, 2, and 3 read in a string described by $(a \cup bbb \cup \epsilon)^*$ while pushing each symbol read onto the stack. The machine nondeterministically transitions from 1 to 4 to guess when the first part of the forward direction is read. States 4, 5 and 6 read in a string described by $(b \cup aaa \cup \epsilon)^*$ while pushing each symbol read onto the stack. The machine nondeterministically transitions from 4 to 7 to guess when the forward part of the string is read. At this point the stack will have the reverse of the forward string on the stack. State 7 tries to read in the reverse while popping the corresponding symbol. If it reads in the reverse, then the stack will only have the dollar sign and the machine will transition to the final state. If there is any more input to read, this branch will terminate. Also, if the remaining string to be read does not match the reverse on the stack that computation will terminate. If the string is in $L$ then there is an accepting computation for the string.
(2) Use your PDA from question 1 and the method to convert a PDA to a CFG to form an equivalent CFG. Explain your steps.

**Answer:** First it is required that the PDA be in a specific form having the properties:
(a) The PDA has only one final state,
(b) The PDA only accepts if the stack is empty,
(c) Each transition on the PDA does exactly one of the following: push a symbol on the stack or pop a symbol off the stack.

The PDA given in (1) has a single final state and only enters if it can pop the bottom symbol from the stack (the $ pushed on at the start), so (a) and (b) are met. However, the transition from 1 to 4 and from 4 to 7 neither push nor pop. Therefore, we modify our original PDA by removing the transition from 1 to 4 and adding a state 8 with a transition from 1 to 8 pushing an $ and from 8 to 4 popping an $ while reading no input. Similarly, to fix the transition between 4 and 7 we add a state 9 and transition from 4 to 9 pushing a $ and from 9 to 7 popping a $ while reading no input. This gives the following equivalent PDA with the required form:

Now, we will transform the modified PDA into an CFG $G = (V, \{a,b\}, A_{SF}, R)$, where $V = \{A_{pq} : p,q \in Q\}$, where $Q$ is the set of states of the modified PDA. From the start state $A_{SF}$, we will first add all rules of the form $A_{pq} \rightarrow aA_{rs}b$ where $a,b \in \Sigma, (r,t) \in \delta(p,a,\epsilon)$ and $(q,\epsilon) \in \delta(s,b,t)$ for some $t \in \Gamma$, where $\delta$ is the transition function and $\Gamma$ is the stack alphabet of the modified PDA. These give the following rules:

$$
A_{SF} \rightarrow \epsilon A_{17} \epsilon \\
A_{17} \rightarrow aA_{17}a \mid bA_{27}b \\
A_{27} \rightarrow bA_{37}b \\
A_{37} \rightarrow bA_{17}b \\
A_{14} \rightarrow \epsilon A_{88} \epsilon \\
A_{47} \rightarrow bA_{47}b \mid aA_{57}a \mid \epsilon A_{99} \epsilon \\
A_{57} \rightarrow aA_{67}a \\
A_{67} \rightarrow aA_{47}a
$$

Next, we will add the rules $A_{pq} \rightarrow A_{pr}A_{rq}$ for all $p,q,r \in Q$. This gives many rules that never terminate or are redundant. Shown below is the grammar with the one rule, $A_{17} \rightarrow A_{14}A_{17}$, that results in strings added (but all such rules are added):
Finally, we add to the grammar all rules of the form $A_{pp} \to \epsilon$ for all $p \in Q$. The only two rules used in the grammar to produce strings are shown below in the final grammar (but all rules of this form are added):

Finally, we add to the grammar all rules of the form $A_{pp} \to \epsilon$ for all $p \in Q$. The only two rules used in the grammar to produce strings are shown below in the final grammar (but all rules of this form are added):

\[
\[
\]

Answer: For a contradiction, assume that $L'$ is a CFL. Let $p$ be the pumping length given by the pumping lemma. Take $w = a^pb^pa^pb^p$, then $w \in L'$ (since $a^pb^p \in L_1$) and $|w| > p$. Then from the pumping lemma $w = uvxyz$ with (1) $|vxy| \leq p$, (2) $|vy| > 0$ and (3) for all $i \geq 0$, $uv^ixy^iz \in L'$. $w_2 = uv^2xy^2z \not\in L'$ since $w_2$ is not even and hence cannot be written as $xx$ for any $x \in L_1$, contradicting (3).

If $vxy$ is in the first half of $w$ only, then $w_2 = uv^2xy^2z \not\in L'$ since the expanded first half of $w$ now has length $|2p| + |vy|$, which means that first half of $w_2$ has length $|2p| + |vy|/2$. Hence, $|vy|/2$ bs from the first half are now at the start of the second half of $w_2$. Therefore, the first half of $w_2$ starts with an $a$ and the second half starts with a $b$. Hence, there can be no $x \in L_1$ such that $w_2 = xx$, contradicting (3).

If $vxy$ is only in the second half of $w$ only, then $w_2 = uv^2xy^2z \not\in L'$ since the expanded second half of $w$ now has length $|2p| + |vy|$, which means that second half of $w_2$ has length $|2p| + |vy|/2$. Hence, $|vy|/2$ as from the second half are now at the end of the first half of $w_2$. Therefore, the
first half of $w_2$ ends with an $a$ and the second half ends with a $b$. Hence, there can be no $x \in L_1$ such that $w_2 = xx$, contradicting (3).

- Therefore, $vxy$ must straddle the middle of $w$. Then by (1), $w_0 = uxz = a^p b^j a^k b^p \notin L'$. If $j > k$ then as above, the first half of the of $w_2$ starts with an $a$ and the second half starts with a $b$, so $w_0 \notin L'$. If $j < k$, then the first half of $w_2$ ends with an $a$ and the second half ends with a $b$, so $w_0 \notin L'$. Finally, if $j = k$, then both $j$ and $k$ are less than $p$ so their are more $a$s at the start of the first half of $w_0$ than at the second half. Therefore, $w_0 \notin L'$ in all cases.

Thus, in every case, $w$ cannot be pumped contradicting (3). Therefore, $L'$ is not a CFL.

(4) Suppose that we modify the notion of a PDA as follows. An ePDA is defined just like a PDA, but accepts if and only if (1) reads all of its input, and (2) has an empty stack at the end. It does not check whether it is in a final state in determining whether or not to accept.

(a) Must every ePDA accept $\epsilon$? Why?

**Answer:** Yes, every ePDA will accept $\epsilon$. By definition of a computation on a PDA (and hence ePDA) at the start of a computation the stack is empty. If additionally, the string being read is $\epsilon$, then at the start of the computation all the input is also read. Therefore, conditions (1) and (2) of the acceptance for an ePDA is met. Therefore, every ePDA must accept $\epsilon$.

(b) If $\epsilon \in L$ and $L$ is context-free, show that there is always an ePDA that recognizes $L$.

**Answer:** Since $\epsilon \in L$ and we have seen in (a) that every ePDA accepts $\epsilon$, we just have to make sure that we can construct an ePDA, $E$, that accepts every other string in $L$ and only those strings in $L$. Since $L$ is a CFL, there exists a PDA $P = (Q, \Sigma, \Gamma, \delta, q_0, F)$ such that $L(P) = L$. We will construct $E$ from $P$. To make $E$ accept only the strings in $L$, we must worry about cases where $P$ is done reading input, empties the stack, but is in a non-final state. In this case $P$ does not accept the input, but $E$ will. To fix this, add a new start state to $P$ and add a transition to the old start state without reading input, that pushes a symbol not in $\Gamma$ on the stack. Then, if $P$ ended in a non-final state with an empty stack, $E$ would end with the new symbol still on the stack. Therefore, $E$ won’t accept any strings that $P$ also will not accept.

We now need to make sure that $E$ will accept everything that $P$ does. As noted above $E$ will accept $\epsilon$. If $P$ on input $w$ ends in a final state with a non-empty stack, then $P$ would still accept; however, $E$ will not. To fix this, add a new state to $E$ and add $\epsilon$-transitions from every state in $F$ to the new state that do not change the stack. Also, add loop transitions in the new state that do not read any input, but pops every symbol on the stack, including the new symbol pushed on from the new start state above. Then, if $P$ ends in a final state after reading all the input, $E$ will have a transition to the new state that pops every symbol on the stack. Thus, $E$ accepts $w$ iff $P$ does.
The machine has a read-only input, and starts at the first character with ε-transition function and q set of states. When in state q, is a counter that is a single storage location that can hold any nonnegative integer. The if (show that any language that is recognized by a counter automaton is context-free.

More formally (but not required). Define E = (Q′ = Q \cup \{q′_0, q_f\}, Σ, Γ′ = Γ \cup \{\$, q′_0, F\}), where $ \notin Γ and δ′ is defined:

\[ δ'(q, a, b) = \begin{cases} δ(q, a, b) & \text{if } q ∈ Q, a ∈ Σ, b ∈ Γ \varepsilon \\ \{(q_0, \$)\} & \text{if } q = q′_0, a = ε, b = ε \\ \{(q_f, ε)\} & \text{if } q ∈ F, a = ε, b = ε \\ \{(q_f, ε)\} & \text{if } q = q_f, a = ε, b ∈ Γ \\ \emptyset & \text{otherwise} \end{cases} \]

If w ∈ L then there is an accepting computation of P, r_0, …, r_m with r_0 = q_0 and r_m ∈ F and s_0 = ε and s_m = x for some x = x_1 … x_n ∈ Γ*. Then r′_0, r_0, …, r_m, r_{m+1}, …, r_{m+2+n} with r′_0 = q′_0, s′_0 = ε, r_0 = q_0, s_0 = $, r_m ∈ F, s_m = x$, and r_{m+i} = q_f for all 1 ≤ i ≤ n, s_{m+i} = x_i … x_n$ for 1 ≤ i ≤ n, s_{m+n+1} = $, s_{m+n+2} = ε is an accepting computation of w for E. Therefore, if P accepts w, then E does. Conversely, assume w \notin L, then for any computation of w on P, say r_0, …, r_n, we have that r_n \notin F and s_n = x for some x ∈ Γ*. Let q′_0, r_0, …, r_n, r_{n+1}, …, r_{n+k} with s_n = x$. Then r_0, …, r_n is a computation on P and hence r_n \notin F. Therefore, r_{n+i} = $ for all 1 ≤ i ≤ k (by definition of δ′). Therefore, when E ends its computation, it at least has $ on the stack and hence will not accept. Therefore, if P doesn’t accept w then neither will E. Thus, L(P) = L(E) and we see that L is recognized by an ePDA.

(5) **not to be graded** A counter automaton is a 6-tuple (Q, Σ, C, δ, q_0, F) where Q is a finite set of states, q_0 ∈ Q is the start state, F ⊆ Q is the set of final states, Σ is the input alphabet, and C is a counter that is a single storage location that can hold any nonnegative integer. The transition function δ maps Q × (Σ ∪ {ε}) to P(Q × {+, −, 0}).

The machine has a read-only input, and starts at the first character with C = 0 and in state q_0. When in state q,

- if (q′, S) ∈ δ(q, ε), it can go to state q′ without reading any input, and
  - add 1 to C if S = +;
  - leave C unchanged if S = 0;
  - subtract 1 from C if S = −, unless C is already 0, in which case the machine cannot apply this transition.

- if (q′, S) ∈ δ(q, a) for a ∈ Σ, if it can read an a, it can go to state q′, and
  - add 1 to C if S = +;
  - leave C unchanged if S = 0;
  - subtract 1 from C if S = −, unless C is already 0, in which case the machine cannot apply this transition.

If there is any sequence of transitions that take the machine from the starting configuration to a final state with all input read, the machine accepts the input; otherwise it does not. Show that any language that is recognized by a counter automaton is context-free.

5
**Answer:** Let $L$ be recognized by the counter automaton $M = (Q, \Sigma, C, \delta, q_0, F)$. We construct a PDA $P = (Q, \Sigma, \Gamma = \{1\}, \delta', q_0, F)$ where $\delta'$ is defined as

$$\delta'(q, a, b) = \begin{cases} 
\{(q', 1) : (q', +) \in \delta(q, a)\} \cup \{(q', \epsilon) : (q', 0) \in \delta(q, a)\} & \text{if } q \in Q, a = \Sigma, b = \epsilon \\
\{(q', \epsilon) : (q', -) \in \delta(q, a)\} & \text{if } q \in Q, a = \Sigma, b = 1 \\
\emptyset & \text{otherwise}
\end{cases}$$

In other words, if the transition of $M$ increases the counter, then the transition of $P$ pushes another 1 on the stack. If the transition of $M$ leaves the counter unchanged, then the transition of $M$ doesn’t change the stack. Finally, if the transition of $M$ decreases the counter, then the transition of $M$ pops a 1 from the stack. Thus at every point the stack has the unary representation of the counter on the stack. Thus, if at some point in the computation of $w$ using $M$, $M$ is in state $q$ with counter $c$, then $P$ would also be in state $q$ with the unary representation of $c$ on the stack at the same point in its computation of $w$. Therefore, $M$ accepts $w$ iff $P$ accepts $w$ and we see $L$ is recognized by a PDA. Thus, $L$ is a CFL.

Note, the converse of this is not true. $\{xx^{rev} : x \in \{a, b\}^*\}$ is a CFL, but no counter automaton will recognize it since it cannot keep track of what order it reads the two symbols in.