CSE 105, Fall 2019 - Homework 4 Solutions

Due: Tuesday 11/12 midnight

Instructions

Upload a single file to Gradescope for each group. All group members’ names and PIDs should be on each page of the submission. Your assignments in this class will be evaluated not only on the correctness of your answers, but on your ability to present your ideas clearly and logically. You should always explain how you arrived at your conclusions, using mathematically sound reasoning. Whether you use formal proof techniques or write a more informal argument for why something is true, your answers should always be well-supported. Your goal should be to convince the reader that your results and methods are sound.

Reading Sipser Sections 2.1 and 2.2

Key Concepts CFG, PDA.
Problem 1 (10 points)

Describe in English, or using set notations, the language generated by the following CFGs. In each case, give 2 examples each of strings which belong to the language of the CFG.

a. \( G_1 = (\{S\}, \{0,1\}, R, S) \). The set of rules \( R \) is given as:
   1. \( S \rightarrow S 0 \)
   2. \( S \rightarrow 1 S \)
   3. \( S \rightarrow \varepsilon \)

   **Solution:**
   In this case, note that any string that you have can only be generated by adding a 1 to the left of \( S \) and a 0 to the right of \( S \). Hence, all strings constructed here will be of the form 111...S000….
   Hence, the general form of the solution here will be:
   \( L_1 = \{1^n0^m \mid n, m \geq 0\} \)

   **Common mistake:**
   Some students wrote that language is the set of all strings of the form \( 0^n1^n \). However, the \( S \) generates strings that end with 0 and start with 1 and has no 1’s after 0’s.

b. \( G_2 = (\{S,A,B\}, \{a, b, c\}, R, S) \). The set of rules \( R \) is given as:
   1. \( S \rightarrow AB \)
   2. \( A \rightarrow aAb \mid \varepsilon \)
   3. \( B \rightarrow bBc \mid \varepsilon \)

   **Solution:**
   In this case, the first thing we observe is that we can still accept the empty string by setting \( A \) and \( B \) as \( \varepsilon \). Next, we construct the string using our variables \( A \) and \( B \). The variable \( A \) here ensures that we have an equal number of \( a \)’s and \( b \)’s for every occurrence of \( A \). The variable \( B \) allows us to have an equal number of \( b \) and \( c \), for every occurrence of \( B \). Hence, the number of \( b \)’s is the sum of number of \( a \)’s and \( c \)’s
   \( L_2 = \{a^ib^jc^k \mid i + k = j\} \)

   **Common mistake:**
   Some solutions mentioned that language is union of strings of the form \( a^n b^n \) and \( b^n c^n \). However, the strings in the language can have \( a, b \) and \( c \).

c. \( G_3 = (\{S,Y\}, \{a,b\}, R, S) \). The set of rules \( R \) is given as:
   1. \( S \rightarrow aY \mid bY \)
   2. \( Y \rightarrow aYa \mid bYb \mid aYb \mid bYa \mid \varepsilon \)

   **Solution:**
   In this case, the description of the language isn’t instantly obvious. We see that trivially, this language rejects the empty string, but can start with either \( a \) or \( b \). Further, the next state the
system has, Y, is independent of whether we started with an a or b. Next, also observe that after this, the string can take the form of all possible strings of even length (since all 4 combinations: aa, ab ,ba or bb are allowed, along with the empty string). Hence, this language accepts all strings which have an odd length.

\[ L_3 = \{ w \mid w \text{ is a string over } \{a, b\}^* \text{ with an odd length} \} \]

**Common mistake:**
Some students wrote that language is the set of all strings over \{a,b\}*. But, the language contains only odd length strings over \{a,b\}*. 

d. \( G_4 = (\{S,Y\}, \{a,b\}, R, S) \). The set of rules R is given as:
1. \( S \rightarrow bSa \mid bY \)
2. \( Y \rightarrow bY \mid aY \mid \epsilon \)

**Solution:**
In this case, observe that all strings generated by this grammar start with b. observe that the Rule 2, which defines the variable Y, is basically a rule which generates all strings in your language. Hence, this CFG constructs the language of all strings which start with a b.

\[ L_4 = \{ w \mid w \text{ is a string over } \{a, b\}^* \text{ which starts with } b \} \]

Note that the rule \( S \rightarrow bSa \) is redundant.

**Common mistake:**
Some students wrote that language is the set of all strings that start with b and end with a. However, as \( S \rightarrow bY \) and \( Y \rightarrow bY \) and \( Y \rightarrow \epsilon \), the language can have strings of the form \( b^n \epsilon \).

Some students also wrote that b’s should not appear after a’s. But such strings can also be part of the language as \( Y \rightarrow aY \) and \( Y \rightarrow bY \).

**Problem 2 (10 points)**
We have seen in Homework 3 that the set of regular languages are closed under the reversal operation. Given a language \( L \subseteq \Sigma^* \) we define its reversal is \( L^R = \{ w^R : w \in L \} \). Prove that the set of context-free languages is also closed under the reversal operation. To do this, consider a CFG given by \( G = \{ V, \Sigma, R, S \} \). Prove that \( L^R \) is context-free by constructing a CFG \( G' = \{ V', \Sigma', R', S' \} \) for \( L^R \), and proving the correctness of our construction.

**Solution:**
We are given a language L that we know is context-free, since it is represented by a CFG G. To prove that the language \( L^R \) is context-free, we must be able to define a CFG which can construct \( L^R \).

Let us say that the CFG that can construct \( L^R \) is \( G' = \{ V', \Sigma', R', S' \} \). First, we define the 4-tuple for this CFG. Here, the alphabet must be the same as that for the original language. Hence, \( \Sigma' = \Sigma \). We will now show that we can construct the new CFG with the same set of start variables as the original CFG, by introducing a new set of rules.
Observe that to reverse the strings in the CFG, we will actually have to reverse the order of each rule that we used to generate the CFG. Any derivation in your CFG must now follow the reversed rules. However, observe that while the rules must be reversed, the order of the rules should not be reversed. Hence, your start variable must now still be $S$.

Correctness:
To demonstrate the correctness of our construct, we must start by considering a string from our language, $w$. We will prove by induction that the reversal will be Context free. We should apply induction on the length of the rules for the generation rules. Let us assume that $w$ has a derivation in $G$ as $w = uxv$ using $k$ derivations in $G$. Here, $u$ and $v$ can be a combination of variables and terminals. $X$ is a variable in $G$. We assume that the reversal of $w = uxv \rightarrow w^R = v^R Xu^R$ is derivable using $G'$.

The idea behind the proof here is that every string that can be constructed from the CFG must have a certain number of derivations for it. In this case, if we want to build the reversed string, I can follow the same set of derivations, with the reversed rules, to reach each intermediate point in the derivation. If we show that after $k$ derivations, if the construction from my first CFG was at a point such that I had a combination of variables and terminals, let’s say, for example, $aBcDeFGhi$, which is a combination of variables and terminals, then I can apply one further derivation on any one rule in my original CFG, and the corresponding reversed rule, and arrive at the next intermediate step such that the two intermediate sets are reverses of each other, until we reach the state in which we have no terminals left in our construct, at which state, if the two constructs are reversal of each other, we have arrived at our proof.

Base case: When $k = 0$
In this case, your derivation must have the derivation $S \rightarrow S$. This must be the first derivation rule, to set the initial start variable. Since the reversal of this is $S^R = S$, which belongs to $G'$, this case is trivially correct.

IH: Let’s say that after $k$ derivations in $G$, we reach an intermediate step $w$, which is a combination of terminals and variables. Our assumption here is that after $k$ derivations in $G'$, we also reach the intermediate step $w'$.
If $w$ does not have any variables in it, i.e. $w$ can be obtained from your CFG $G$ using $k$ derivations, then our assumption here ensures that the reversed string can be constructed using $G'$, since $w'$ must also not have variables in it. However, if after $k$ derivations in $G$, we assume we have $w = uXv$ where $u$ and $v$ are a combination of terminals and variables, and $X$ is the next variable on which we apply our derivation. By the induction hypothesis, the reversal of $w$, $w^R = v^R Xu^R$, is derivable using $k$ derivations in $G'$, i.e. $w' = w^R$.

Next, the derivation rule exists in $G$ for $X \rightarrow a$, where $a$ is some combinations of variables and terminals (can have only one of the two, or just be the empty string). Hence, we have $w_{k+1} = uav$ in $G$. The reversal of this intermediate construct must be $w_{k+1}^R = v^R a^R u^R$. We need
to show that this derivation must exist in $G'$ as well. We know that $w^R = v^Rxu^R$ exists in $G'$. We also know that the reversed derivation rules $R'$ for variable $X$ must now go as $X \rightarrow a^R$. Hence, applying this as the next derivation rule from $G'$, on $w$, we get $w'_{k+1} = v^Rau^R$, which is the same as the reversal of the string $w_{k+1}$. Therefore, at each intermediate step for $G$, we can attain a corresponding reversed state in $G'$. This proves that $L^R \subseteq L(G')$.

Now, we must also prove that the CFG $G'$ does not generate strings outside $L^R$, i.e. $L(G') \subseteq L^R$. For this case, let us again start as before by considering a string $w'$ from $L(G')$. To prove this, we will show that $w'^R \in L$. We will again use a proof similar to the previous part, using induction, on the length of the derivation.

Base case: When $k = 0$
We have shown in the previous part that the start state can be the same for both $G$ and $G'$. In this case, your derivation must have the starting derivation $S \rightarrow S$. Since the reversal of this is $S^R = S$, which belongs to $G$, and hence consequently to $L^R$ as well, this case is trivially correct.

IH: Let us take the case when we have an intermediate construct $w'$ obtained from $k$ derivations in $L(G')$. If $w'$ does not have any variables in it, i.e. $w'$ can be obtained from your CFG $G'$ using $k$ derivations, then our assumption here ensures that the reversed string can be constructed using $G$, since $w$ must also not have variables in it.
If not, again we can assume without loss of generality, that $w'_k = uXv$. We assume that if $w'_k \in L(G') \rightarrow w_k = v^Rxu^R \in L(G)$. The next derivation rule will be on the variable $X$ here, where $X \rightarrow a$. Notice that from our definition, the equivalent rule from $L(G)$ will go as: $X \rightarrow a^R$. Therefore, $w'_{k+1} = uav$. In this case, $w_{k+1} = v^Rau^R = w'_{k+1} \in L(G)$. Therefore, at each intermediate step for $G'$, we can attain a corresponding reversed state in $G$.

This proves that all strings in $L(G')$ can be represented as a reversal of a string from $L$, and hence, $L(G') \subseteq L^R$.

Therefore, since $L^R \subseteq L(G')$ and $L(G') \subseteq L^R$, $L(G') = L^R$

Common Mistakes:
The most common mistake was that most students gave only one direction of the proof. That is, it was proved that the reverse of the language $L$ can be generated by the constructed grammar but to prove that the reverse of the language $L$ is the language corresponding to the constructed grammar, it is also necessary to prove that every string generated by the constructed grammar is the reverse of some string in the language.
Another common mistake was that students showed proof by induction but instead inducing over the number of derivation, they wrongly induced over the size of the string or the number of rules in the grammar.

Some students gave informal arguments which were valid but do not constitute a thorough mathematical proof which the question required.

Problem 3 (10 points)

In the next two questions, we will investigate the closure properties of Context Free Languages. To do so, consider two CFGs given as: $G_1 = \{V_1, \Sigma, R_1, S_1\}$ for a language $L_1$ and $G_2 = \{V_2, \Sigma, R_2, S_2\}$ for a language $L_2$. Construct a CFG that accepts:

- a) $L_1 \circ L_2$
- b) $L_1 \cup L_2$
- c) $L_1^*$

In each case, prove the correctness of your construction.

Solution:

a) For the case of concatenation, let us assume that our new string is derived from the CFG $G' = \{V', \Sigma', R', S'\}$. In this case, the alphabet remains the same, so $\Sigma' = \Sigma$. To obtain the concatenation, we just need to change one rule in $R$, by adding a new starting variable $S$, with the transitions defined as:

$$S \rightarrow S_1 S_2$$

The rest of the rules from $S_1$ and $S_2$ onwards is the same as the rules for $G_1$ and $G_2$. Therefore:

$$V' = \{S\} \cup V_1 \cup V_2$$

$$R' = \{S \rightarrow S_1 S_2\} \cup R_1 \cup R_2$$

$$S' = S$$

Correctness: Let us say that you have $w = w_1 w_2$, $w_1 \in L_1$ and $w_2 \in L_2$. In this case, we must have a set of derivation rules for $w_1$ from $S_1$ (the start state of $G_1$) $S_1 \Rightarrow_{G_1} w_1$, and another set of derivation rules for $w_2$ from $S_2$ (the start state of $G_2$) $S_2 \Rightarrow_{G_2} w_2$. Since our start state of $G'$ is $S$, and the first derivation rule is $S \rightarrow S_1 S_2$, we can use the same set of derivation rules $S_1 \Rightarrow_{G_1} w_1$ to first get the string $w_1 S_2$ and then, follow the derivation rules $S_2 \Rightarrow_{G_2} w_2$ to get $w_1 w_2$. This proves that $L_1 \circ L_2 \subseteq L(G')$

Next, we go to the reverse direction. Let’s say that we consider a string $w \in L(G')$. We know that this string must come from the starting rule $S \rightarrow S_1 S_2$. This means that we can have one set of derivations starting from $S_1$, which would lead to the first half of the string, leading to a string $w_1 \in L_1$, and the transition from $S_2$, leading to a string $w_2 \in L_2$. Therefore, $w = w_1 w_2$, which means all strings within your language belong to $L_1 \circ L_2$. Therefore, $L(G') \subseteq L_1 \circ L_2$.
This consequently allows us to have \( L_1 \circ L_2 = L(G') \)

b) For the case of union, let us assume that our new string is again derived from a CFG \( G' = \{ V', \; \Sigma', \; R', \; S' \} \). In this case, the alphabet remains the same, so \( \Sigma' = \Sigma \). To obtain the concatenation, we just need to change one rule in \( R \) by adding a new starting variable \( S \), with the transitions defined as:

\[
S \rightarrow S_1 | S_2
\]

The rest of the rules from \( S_1 \) and \( S_2 \) onwards is the same as the rules for \( G_1 \) and \( G_2 \). Therefore:

\[
V' = \{ S \} \cup V_1 \cup V_2
\]

\[
R' = \{ S \rightarrow S_1 | S_2 \} \cup R_1 \cup R_2
\]

\[
S' = S
\]

Correctness: Let us say that you have \( w \). We can have two cases here: \( w \in L_1 \) or \( w \in L_2 \). In this case, if we have \( w \in L_1 \) we must have a set of derivation rules for \( w \) from \( S_1 \) (the start state of \( G_1 \)). \( S_1 \Rightarrow G_1 w \). The start state of \( G' \) can then make the transition \( S \rightarrow S_1 \) to lead to an intermediate state in the construction \( w = S_1 \). Since we have retained all other construction rules of the CFG \( G_1 \), we can still make the transitions \( S_1 \Rightarrow G_1 w \) to get \( w \). Similarly, if \( w \in L_2 \), we can proceed as above, and have a set of derivation rules for \( w \) from \( S_2 \) (the start state of \( G_2 \)). \( S_2 \Rightarrow G_2 w \). The start state of \( G' \) can then make the transition \( S \rightarrow S_2 \) to lead to an intermediate state in the construction \( w = S_2 \). Since we have also retained all other construction rules of the CFG \( G_2 \), we can still make the transitions \( S_2 \Rightarrow G_2 w \) to get \( w \). This proves that \( L_1 \cup L_2 \subseteq L(G') \).

This completes the first half of the proof. For the 2nd half, we need to prove that any string that can be generated using \( G' \) must lie in \( L_1 \cup L_2 \). For this, assume that \( w \in L(G') \). Thus, in this case, \( w \) must have its origin in either \( S_1 \) or \( S_2 \). If the first transition here is \( S \rightarrow S_1 \), then we effectively replace our variable \( S \) with \( S_1 \), which leads to \( w \in L_1 \). If the first transition here is \( S \rightarrow S_2 \), then we effectively replace our variable \( S \) with \( S_2 \), which leads to \( w \in L_2 \). Since there are no further transitions possible for \( S \), we must have \( w \in L_1 \) or \( w \in L_2 \). Therefore, \( L(G') \subseteq L_1 \cup L_2 \).

Since \( L_1 \cup L_2 \subseteq L(G') \) and \( L(G') \subseteq L_1 \cup L_2 \), we have the bi-directional proof allowing \( L(G') = L_1 \cup L_2 \).

c) For the Kleene star operation, we again define a CFG \( G' = \{ V', \; \Sigma', \; R', \; S' \} \). In this case, the alphabet remains the same, so \( \Sigma' = \Sigma \). To obtain Kleene star, we need to modify the start state to allow multiple repetitions of the language:

\[
S \rightarrow SS_1 | \varepsilon
\]
We retain the remaining set of rules from $G_1$. Therefore, we have the definition as:

$$V' = \{S\} \cup V_1$$
$$R' = \{S \rightarrow SS_1 | \varepsilon\} \cup R_1$$
$$S' = S$$

Correctness: Let us say that you have $w$. In this case, $w$ has 2 cases: $w = \varepsilon$ or $w = w_1w_2...w_n$, $n > 0$, $w_i \in L_1 \forall i \leq n$. In the first case, we can use the derivation $S \rightarrow \varepsilon$.

In this case, we can observe that each of the strings $w_1...w_n$ will have a set of derivations in $G_1$ starting from $S_1$. Then, we can start from the new start state of $G'$, $S \rightarrow SS_1$, and repeat the derivation $n$ times, to give $S \rightarrow SS_1S_1...S_1$, where we have $n$ occurrences of $S_1$.

We can repeat this for all values of $n$, and hence, we can have a derivation from each of the occurrences of $S_1$ to the strings $w_1...w_n$. Therefore, $L_1^* \subseteq L(G')$.

Let us look at the reverse direction for the two languages. Let’s say we start with a string $w$ in $L(G')$. In this case, the start state can either transition to the empty string, or it make the transition $S \rightarrow SS_1$. If we make the transition $S \rightarrow \varepsilon$, then we have $w = \varepsilon$. Let’s say $w$ was made by $n$ transitions of $S \rightarrow SS_1$, after which it terminates with $S \rightarrow \varepsilon$. This leaves us with $S \rightarrow S_1S_1...S_1$ ($n$ times). Each $S_1$ allows us to make a string $w_k = w_1w_2...w_n$. Here, we observe that consequently $w \in L_1^*$. Therefore, $L(G') \subseteq L_1^*$. Therefore, from the bi-directional proof result, we have $L(G') = L_1^*$.

Common Mistakes:
Many groups have just included the extra rule that needs to be introduced in the new grammar and forgot to add the existing rules for languages $L_1$ and $L_2$ in the new grammar.
Many groups gave proofs in just one direction for bidirectional proofs.
Many groups have written that CFGs “recognize” language. CFGs don’t recognize languages. They generate or derive them. PDAs recognize languages.

Problem 4 (10 points)
We have seen earlier that the Pumping Lemma can be used to show that a language is not regular. Interestingly, with some modifications, the Pumping Lemma can also be used to prove whether a language is context-free. The Pumping Lemma for Context Free Languages is defined as follows:

For any context free language $L$, there exists a number $p$, called the pumping length, such that any string $w \in L$ which has length greater than $p$ can be written in the form $w = uvxyz$, so that either $v$ or $y$ is non-empty, and $uv^ixy^iz \in L$, for all $i \geq 0$. (For reference, see Theorem 2.34 in the book.) That is,

1. $|xy| \leq p$
2. $|y| > 0$
3. $uv^ixy^iz \in L$, $\forall i \geq 0$

Consider the two CFGs:
$G_1 = (\{A, B, C\}, \{a,b,c\}, R, S)$. The set of rules $R$ is given as:

1. $S \rightarrow AC$
2.  \[ A \rightarrow aAb | \varepsilon \]
3.  \[ C \rightarrow cC | \varepsilon \]

\[ G_2 = (\{S, B\}, \{a,b,c\}, R, S) \]. The set of rules \( R \) is given as:

1.  \[ S \rightarrow aSc | B | \varepsilon \]
2.  \[ B \rightarrow bB | \varepsilon \]

a) Describe, in English, the languages \( L_1 \) and \( L_2 \) defined by \( G_1 \) and \( G_2 \).

b) Let \( L = L_1 \cap L_2 \). Use the pumping lemma for context-free languages to prove that \( L \) is not context-free. Hence, prove that the class of context-free languages is not closed under the intersection operation.

c) Using the fact that you proved in (b) that context-free languages are not closed under intersection, prove that the class of context-free languages are not closed under complementation.

**Solution:**

a) The CFG \( G_1 \) is a construct which basically introduces the two variables \( A \) and \( C \) from the start state. The variable \( A \) is used to generate the terminals \( a \) and \( b \). The number of \( a \)'s and \( b \)'s will be equal, and of the form \( aa...bb... \).

The number of \( c \)'s in this case can be any number, since there are no restrictions on the variable \( C \).

Therefore: \( L_1 = \{a^i b^j c^i | i, j \geq 0 \} \)

The CFG \( G_2 \) is a construct which introduces an equal number of \( a \)'s and \( c \)'s at the start, while also allowing us to keep the variable \( B \). Hence, the language here is:

\[ L_1 = \{a^i b^j c^j | i, j \geq 0 \} \]

**Common Mistakes:**

Groups did not give constraints for the values of \( i \) and \( j \) for both languages in their English descriptions, \( i.e \ i, j \geq 0 \)

b) Therefore, the intersection of the two languages is:

\[ L = \{a^i b^j c^i | i \geq 0 \} \]

Using the pumping lemma, we will prove that the intersection of two context-free languages \( L_1 \) and \( L_2 \), will not be context-free.

As usual, we begin by assuming that the language is context-free. Hence, it must have a context-free grammar. Assume the pumping length is \( p \). Next, we consider a string of the form:

\[ w = a^p b^p c^p \]

Clearly, \( w \in L \) and \( |w| \geq p \)

Therefore, by the pumping lemma, we must be able to split it into a string of the form \( uvxyz \) such that \( uv^ixy^iz \) is also in \( L \).
This string will always violate the pumping lemma. To prove this consider the definition of the string and of \( vxy \). The maximum length of \( vxy \) is \( p \), which means that \( v, x \) and \( y \) can be either homogeneous (consisting entirely of \( a \)'s, \( b \)'s or \( c \)'s), or have a combination of \( a \)'s and \( b \)'s, or a combination of \( b \)'s and \( c \)'s, but never all 3 alphabets. This is because \( |vxy| \) has to be less than or equal to \( p \). And since we have a sequence of \( p \) alphabets of each kind: \( a, b \) and \( c \), we cannot have a combination of all 3. Note that in this case, unlike the case with the pumping lemma for regular languages, there is no condition on the first string of the split, \( u \).

For this proof, we will hence have multiple cases we need to consider. Remember, the pumping lemma requires us to check all possible splits for the string \( s \).

**Case 1:** Let us consider the case when \( v, x \) and \( y \) consists entirely of one alphabet. In this case, when we pump down, i.e., we consider \( uv^0xyz^0z \), there will be at least one alphabet less than the other two, since at least one of \( v \) or \( y \) must have length 1. This means that \( uv^0xyz^0z \) will not lie in \( L \), and hence this particular combination of \( u, v, x, y \) and \( z \) cannot be pumped.

**Case 2:** Let us consider the case when \( vxy \) is made up of a combination of \( a \)'s and \( b \)'s, or \( b \)'s and \( c \)'s. Let us assume that \( vxy = a^m b^n \), i.e a combination of \( a \)'s and \( b \)'s. Note, that the same case will hold if \( vxy \) consists of \( b \)'s and \( c \)'s. In this case, either \( v \) or \( y \) must consist of at least one \( a \), or one \( b \). So, when we pump this string, say take the string: \( uv^2xy^2z \), the number of \( a \)'s or \( b \)'s, or both will increase to at least \( p+1 \). However, the number of \( c \)'s still remains \( p \). Therefore, this string does not belong to the language \( L \) either. In this case, if \( v \) or \( y \) is a combination of \( a \) and \( b \)'s, pumping them would lead to strings of the form \( a^k b^l a^k b^l \), which means that this string won’t belong to \( L \) in any case. The same argument can hold for the case where \( v \) or \( y \) is a combination of \( b \) and \( c \)'s.

The core of the proof here is that in no case can we simultaneously increase or decrease the number of \( a \)'s, \( b \)'s and \( c \)'s. Therefore, this string will violate pumping lemma for all possible \( u, v, x, y \) and \( z \). Therefore \( L \) does not satisfy the pumping lemma, and hence the language \( L \) is not context-free.

We have shown already that there exists a CFG that can describe the two languages, and hence \( L_1 \) and \( L_2 \) are context free.

The intersection of \( L_1 \) and \( L_2 \), the language \( L \), given as: \( L = \{a^i b^j c^k \mid i = j = k \} \), is not context free.

**Common Mistakes:**

Groups did not cover the multiple cases of the different ways \( S \) can be split into \( uvxyz \) and therefore did not have a complete proof.

c) We prove by contradiction. Let us assume that the set of context-free languages is closed under complementation. Let us say we have two context-free languages \( L_1 \) and \( L_2 \). Now, we know that the set of CFLs is closed under the union operation, as was proved in Q3. From the previous part of this question, we also know that the set of CFLs are not closed under intersection.
Let us consider the language $L_1 \cap L_2$.
We know from the De-Morgan’s law that: $L_1 \cap L_2 = ((L_1)^C \cup (L_2)^C)^C$
Now, if our assumption of closure of CFLs under complementation holds, then
$((L_1)^C \cup (L_2)^C)^C$ must be context-free, since we already know that CFLs are closed
under union. However, we have shown in (b) that there exist two CFLs $L_1 and L_2$ such
that $L_1 \cap L_2$ is not context-free. Therefore, our assumption above is incorrect, and we
have a contradiction. Therefore, at least one of $(L_1)^C or (L_2)^C$ is not context-free, and
hence CFLs are not closed under complementation.

Common Mistakes:
Groups did not take advantage of DeMorgan’s Law or the fact that CFL’s are closed
under Union to prove that CFLs are not closed under complementation.

Problem 5 (10 points)
Construct the CFG and PDA for the languages given by:

a) $L = \{0^i1^j2^k | i + j = k\}$
b) $L = \{ab^i | i \neq j\}$

For the PDA, you must show the state diagram for your construction. You do not need to provide
the formal definition or the proof of correctness.

Solution:

a) $L = \{0^i1^j2^k | i + j = k\}$

CFG for $L$:
$G : \{ \{S, B\}, \{0, 1, 2\}, R, S \}$
$S \rightarrow 0S2 | B$
$B \rightarrow 1B2 | \varepsilon$

PDA:
Common Mistakes

- Did not specify which variable to start from in the CFG
- Created a CFG for $0^*1^*2^*$
- Created a CFG for $(0^*1^*)^*2^*$
- The CFG did not allow for zero 0s or zero 1s
- The CFG did not generate the empty string
- Did not specify the start state for the PDA
- The PDA did not accept the empty string
- The PDA did not allow for zero 0s or zero 1s

b) $L = \{a^ib^j \mid i \neq j\}$

CFG for $L$:

$G : \{ \{S, A, B\}, \{a, b\}, R, S \}$

$S \rightarrow aA \mid Bb$

$A \rightarrow aAb \mid aA \mid \varepsilon$

$B \rightarrow aBb \mid Bb \mid \varepsilon$

PDA:
Notes

- Verify that your PDA accepts / your CFG generates at least the following strings
  - a, aa, aab, aaabb, b, bb, abb, abbb, aabbb
- Verify that your PDA rejects / your CFG cannot generate at least the following strings
  - empty string, ab, aabb, ba, bbbaa

Common Mistakes

- Did not specify which variable to start from in the CFG
- CFG generated some version of a*b* (i.e. it didn’t guarantee that there were more/fewer a’s than b’s)
- CFG could not generate a* or b*
- Did not specify the start state for the PDA
- PDA could not accept strings that had two or more b’s than a’s
- PDA accepted strings with more a’s than b’s but not more b’s than a’s
  - Usually this was because there was a loop missing to read in the remaining b’s