COMS W3261 Fall 2022 Handout 7b: Midterm Review Solutions

Helen Chu and Eumin Hong

October 23, 2022

1 Problem Set A: Regular Languages

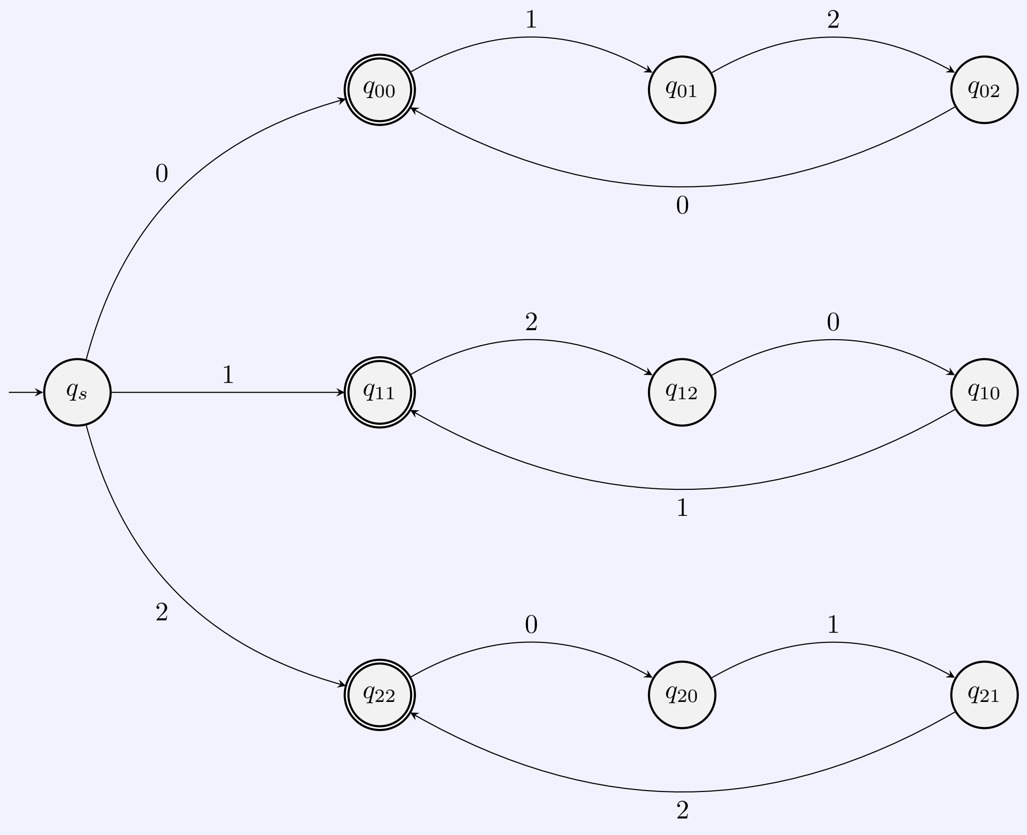
Design deterministic finite automata for the following languages. You can give the DFA by their transition diagrams. You do not need to show that they are correct.
Note: In your transition diagrams, you can use shorthand notation on the labels of the edges. For example, you can label an edge by $\Sigma \setminus \{a\}$ to indicate that the transition takes place for all input symbols except a. Make sure to specify the starting state and the accepting states in your diagrams.

1. L is the language over the alphabet $\Sigma = \{0, 1, 2\}$ consisting of all strings that:

   - Every 0 is immediately followed by a 1, every 1 is immediately followed by a 2, and every 2 is immediately followed by a 0.
   - The string starts and ends with the same symbol.
   - The string must have length at least 1.

Solution

We can construct a DFA with an omitted garbage state that recognizes L:

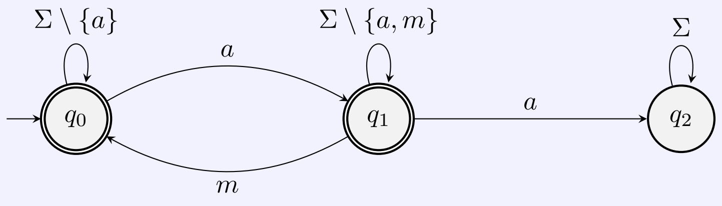


All accepted strings must have length at least 1 since the initial state is not accepting. We branch from q_s depending on the first symbol of w. WLOG, let the first symbol of w be 0. Once we take the transition from q_s to q_{00}, we return to the accepting state q_{00} if and only if we read a sequence of symbols (after the initial 0) of the form $(120)^*$, which ensures the first two conditions of the strings in the language.

2. The set of strings over the alphabet $\Sigma = \{a, b, \ldots, z\}$ that contain at least one m between any two a's in the string; for example $abc, john, mama, american$ are in the language, but $papa, panamerican$ are not.

Solution

We can construct a DFA that recognizes this language:



Whenever we see an a, if we ever see another a before seeing any ms, we reject.

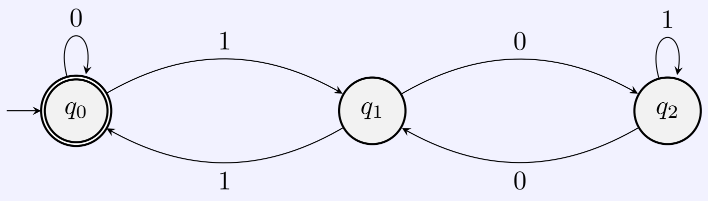
3. **Challenge:** The set of binary strings which represent in binary a number that is an integer multiple of 3 (leading zeros are allowed). For example, 00, 110 are in the language (they correspond to 0, 6 respectively), but 001, 101 are not (they correspond to 1, 5). Hint: Three states are enough. Think about what each additional symbol in a binary string does to the number; it might be helpful to write out some examples.

Solution

As the hint mentions, three states are enough for our DFA; thus, it is natural to think about how we can use the three states as a counter for the modulus operation, where 0 mod 3 is the accepting state for our binary string. We see the following relationship between the values mod 3:

curr mod 3	x	next mod 3
0	0	0
0	1	1
1	0	2
1	1	0
2	0	1
2	1	2

Thus we construct our DFA as follows:



For the following problems, if a language L is given, prove that L is regular or prove that L is nonregular.

4. $L = \{ww \mid w \in \{0,1\}^*$ and w contains at least one 0 and at least one 1$\}$ over the alphabet $\Sigma = \{0,1\}$.

Solution

Define the following language:

$$A = \{ww \mid w \in \{0,1\}^* \text{ and } w \text{ does not contain both a 0 and a 1}\}$$

We see that $A = L((00)^* \cup (11)^*)$, as every string in A must have even length (due every string in A having form ww) and must be composed of only 0s or only 1s (in order to not contain both a 0 and a 1).
We then observe that $L \cup A = B$, where

$$B = \{ww \mid w \in \{0,1\}^*\}$$

Since A can be expressed as a regular expression, A is regular. Now assume, for the sake of contradiction, that L is also regular. Since the class of regular languages is closed under the union operation, if L and A are both regular, then B must also be regular.
However, B is not context-free and therefore not regular.[a]
Since B is not regular, then due to closure properties, it cannot be the case that both L and A are regular, so L must be nonregular.

[a]Since it is true that if a language is regular, it is context-free, the contrapositive must be true. It was proved that B is not context-free in Lecture 12 (10/17).

5. Prove that the union of a regular language L_1 and nonregular language L_2, $L_1 \cup L_2$, such that $L_1 \cap L_2 = \emptyset$ results in a nonregular language.

Solution

To prove that $L_1 \cup L_2$ is nonregular, we assume for the sake of contradiction that it is regular. Then, we aim to show that if $L_1 \cup L_2$ is regular, then L_2 will always be regular, which is a contradiction.
To construct an expression for L_2 in terms of $L_1 \cup L_2$ and L_1, we observe that

$$L_2 = (L_1 \cup L_2) \cap \overline{L_1}$$

In other words, L_2 is composed of everything in both L_1 and L_2 (which is $L_1 \cup L_2$) and everything that is not in L_1 (which is $\overline{L_1}$). Since L_1 is regular and the class of regular languages is closed under the complement operation, $\overline{L_1}$ must also be regular. Since (we assumed for the sake of contradiction that) $L_1 \cup L_2$ is regular and the class of regular languages is closed under the intersection operation, $(L_1 \cup L_2) \cap \overline{L_1} = L_2$ must be regular. However, this leads us to a contradiction, as L_2 was stated to be nonregular. Thus, $L_1 \cup L_2$ must be nonregular.

6. $L = \{a^i b^j c^k \mid i + j = k\}$.

Solution

L is nonregular as it does not satisfy the pumping lemma for regular languages. We choose the string $w = a^p b^p c^{2p}$, and it is clear that $w \in L$ and $|w| = 4p \geq p$. For all partitions $w = xyz$, due to the conditions $|xy| \leq p$ and $|y| > 0$, we see that y must consist of at least one a and is composed only of as, or in other words, $y = a^k, 0 < k \leq p$. We then consider the pumped-up string $xy^2z = a^{p+k} b^p c^{2p}$ and see that $xy^2z \notin L$ (as $(p+k) + p = 2p + k \neq 2p$ since $k \neq 0$); for all partitions $w = xyz$, w cannot be pumped. Since L does not satisfy the pumping lemma for regular languages, L is not a regular language.

7. $L = \{0^k u 0^k \mid k \geq 1, u \in \Sigma^*\}$.

Solution

We prove that L is regular by providing a regular expression. We observe that if $k = 1$, then L contains all strings that start and end with 0 (note that u includes all characters between the starting and ending characters; this is possible since $u \in \Sigma^*$). Due to this observation, $L = L(0\Sigma^*0)$, so L is regular.

8. $L = \{0^k 1 u 0^k \mid k \geq 1, u \in \Sigma^*\}$.

Solution

L is nonregular as it does not satisfy the pumping lemma for regular languages. We choose the string $w = 0^p10^p$, and it is clear that $w \in L$ and $|w| = 2p + 1 \geq p$. For all partitions $w = xyz$, due to the conditions $|xy| \leq p$ and $|y| > 0$, we see that y must consist of at least one 0 and is composed only of 0s, or in other words, $y = 0^k, 0 < k \leq p$. We then consider the pumped-up string $xy^2z = 0^{p+k}10^k$ and see that $xy^2zz \notin L$; for all partitions $w = xyz$, w cannot be pumped.[a] Since L does not satisfy the pumping lemma for regular languages, L is not a regular language.

[a]By pumping up, we see the difference between this language L and the previous language $L_R = \{0^k u 0^k \mid k \geq 1, u \in \Sigma^*\}$: the fixed 1 allows for the distinction between the substring of the first occurrence of 0^k and u, which in turn allows us to pump up the string w in our solution.

9. $L = \{0^i 1^j \mid i, j \geq 0, i \neq j\}$.

Solution

We will prove that L is nonregular using closure properties. Assume, for the sake of contradiction, that L is regular. Then, we see that

$$L(0^*1^*) \cap \overline{L} = \{0^n1^n \mid n \geq 0\}$$

It is important to note that while $\overline{L}$ does contain strings of the form $0^n1^n, n \geq 0$, $\overline{L}$ also contains strings that are not of the form 0^*1^* as these strings are not in L. Thus, when we take the intersection of $L(0^*1^*)$ and $\overline{L}$, the result is the set of all strings of the form 0^*1^* where the number of 0s is equal to the number of 1s. However, the resulting language $\{0^n1^n \mid n \geq 0\}$ is nonregular as it was proved in class.[a]
Since $\{0^n1^n \mid n \geq 0\}$ is nonregular, due to closure properties, it cannot be the case that both $L(0^*1^*)$ and $\overline{L}$ are regular, so $\overline{L}$ must be nonregular. Since the class of regular languages is closed under the complement operation, if $\overline{L}$ is nonregular, then L itself must be nonregular.[b]

[a]See Lecture 8 (10/3). If you would like to use results from class, you must state that you are doing so; simply stating that the resulting language is nonregular without stating that it was proved such in class is not sufficient.
[b]To see this, consider what would be true about $\overline{L}$ if L were to be regular.

10. $L = \{0^n1^n \mid 0 \leq n \leq 3\}$

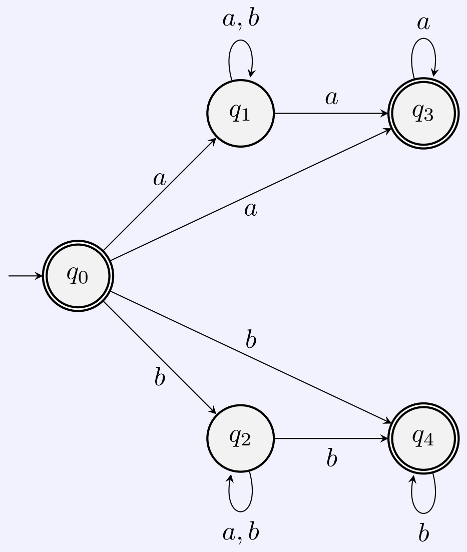
Solution

Since L is finite, L is regular.

11. **Challenge:** Let L be the language consisting of all strings of a's and b's with an equal number of occurrences of ab and ba as substrings. (The string $aabbbaa$ has one occurrence of each of the substrings ab and ba.) Is L a regular language? Prove your answer.

Solution

Yes, L is regular: In fact, L can be described equivalently as the set of strings over $\{a, b\}$ that begin and end with the same symbol. We break into two scenarios: there are no substrings of ab and ba or there are one or more substrings of ab and ba. The first case is trivial; for the second scenario, wlog, let the string start with a. Once ab is introduced to the string, the string must end with an a to balance the amount of substrings of ab and ba. If the string instead ends with b, then there are now more occurances of ab than ba. We can construct an NFA that recognizes L as follows:



Or equivalently, L can be generated by the regular expression

$$\varepsilon \ \cup \ a \ \cup \ b \ \cup \ (a(a \cup b)^* a) \ \cup \ (b(a \cup b)^* b)$$

2 Problem Set B: Context Free Languages

1. Show that the language

$$L = \{a^i b^j \mid i, j \geq 0, i \leq j \leq 3i\}$$

is context-free.

Solution

The language L is context-free because the following CFG G_3 generates it. Let $G_3 = (\{S\}, \{a, b\}, R, S)$ where $R = \{S \to aSb \mid aSbb \mid aSbbb \mid \varepsilon\}$. An informal justification is as follows: for each non-empty string (the string ε is trivially in L and generated by G), each derivation generates one a on the left and n b's on the right where $1 \leq b \leq 3$. If i a's are generated, then the minimum number of b's generated is $j = i$ and the maximum number of b's generated is $j = 3i$, so $i \leq j \leq 3i$.

2. Give a context-free grammar that generates the language

$$L = \{a^i b^j c^k : i = j \text{ or } j = k \text{ where } i, j, k \geq 0\}$$

Is your grammar ambiguous? Why or why not? (Consider the string $a^n b^n c^n$.)

Solution

An approach is to consider L as the union of two languages where $i = j$ and $j = k$, and construct the CFG accordingly:

$$\begin{aligned}
S &\to S_1 \mid S_2 \\
S_1 &\to UV \\
U &\to aUb \mid \varepsilon \\
V &\to cV \mid \varepsilon \\
S_2 &\to XY \\
X &\to aX \mid \varepsilon \\
Y &\to bYc \mid \varepsilon
\end{aligned}$$

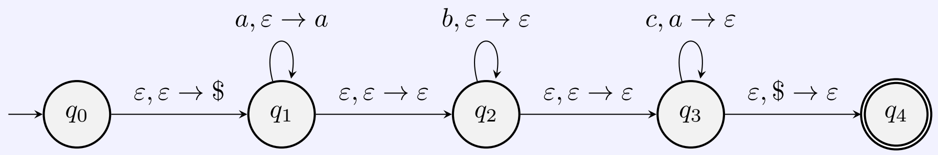
An informal justification: S_1 derives the strings where $i = j$, because the rule $U \to aUb$ ensures that $|a| = |b|$, and the rule $S_1 \to UV$ ensures the order of abc. S_2 derives the strings where $j = k$, because the rule $Y \to bYc$ ensures that $|b| = |c|$, and the rule $S_2 \to XY$ ensures the order of abc.
This grammar is ambiguous; Given the discussion at the beginning, $w = a^n b^n c^n$ satisfies both $i = j$ and $j = k$ and thus, can be derived from both S1 and S2. This gives two parse trees for w.

3. Construct a pushdown automata for the language

$$L = \{a^n b^k c^n \mid n, k \geq 0\}$$

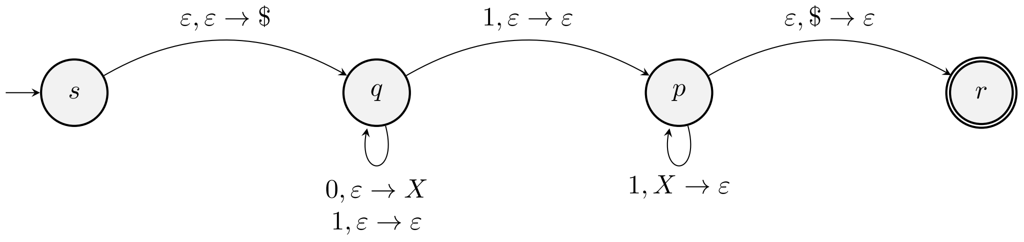
Solution

We construct PDA P as follows:


$a, \varepsilon \to a$
$b, \varepsilon \to \varepsilon$
$c, a \to \varepsilon$
$\varepsilon, \varepsilon \to \$$
$\varepsilon, \varepsilon \to \varepsilon$
$\varepsilon, \varepsilon \to \varepsilon$
$\varepsilon, \$ \to \varepsilon$
q_0
q_1
q_2
q_3
q_4

The equivalence proof is similar to that of HW3 question 2.
First, we show that if $w = a^n b^k c^n$ such that $n, k \geq 0$, then it is accepted by P. When running P on input w, first \$ is pushed to the stack and we enter state q_1. Then we loop on q_1 n times such that n a's are pushed to the stack and we enter state q_2, where all k b's are read. In q_3, we read n c's, popping n a's from the stack. This leaves us with just \$ on the stack, so P transitions to the accept state.
Then, we show if P accepts a string w, $w \in L$. P first pushes a \$ to the stack. Then it either reads and pushes $n > 0$ a's or moves to state q_2 immediately. From q_2, the only way for the computation path to reach state q_3 with only c's remaining is if all b's occur before the c's. From q_2, we have to read exactly as many d's as a's and pop that many a's from the stack. This way, we transition to the accept state popping the final \$ on the stack.

4. Describe the language that the following PDA recognizes:


$\varepsilon, \varepsilon \to \$$
$1, \varepsilon \to \varepsilon$
$\varepsilon, \$ \to \varepsilon$
s
q
p
r
$0, \varepsilon \to X$
$1, \varepsilon \to \varepsilon$
$1, X \to \varepsilon$

Solution

$$L = \{xy \mid x \in \{0,1\}^*, y \in \{1\}^*, \text{the number of 0's in } x < |y|\}$$

We prove equivalence through proving an if-and-only-if relationship.
We first show if a string w is accepted by the PDA, then $w \in L$. We transition from state s to state q by pushing a \$ marker to the stack, then we loop on state q where, for every 0 we see, we push an X to the stack. For an input of 1 we have two choices - to stay on q, or nondeterminiscally choose to transition to state p, where for every 1 we see, we pop an X off the stack. Finally, we transition to state r by popping \$ off the stack. Thus we see that state q generates $\{0,1\}^*$ while the stack counts the number of 0's, the transition to state p takes in a 1, and we must loop on state p until we've popped all of the X's off the stack before we can transition to the accept state, thus the string must have a suffix of 1's at least as long as the number of 0's in the string.
We then show if $w \in L$ then w is accepted by the PDA. Let the number of 0's in w be called x. Because of the nondeterministic transitions from state q, the PDA is guaranteed to accept any string such that the length of its suffix of 1's is greater than x, since we can loop on q and take the 1 transition to p at the $x + 1$th character from the end of the string.
Thus L is equal to the language recognized by the PDA.

5. **Challenge:** For any language L, let $\mathsf{SUFFIX}(A) = \{v \mid uv \in L \text{ for some string } u\}$. Show that the class of context-free languages is closed under the SUFFIX operation.

Solution

To show that the class of context-free languages is closed under the SUFFIX operation, we show that if L is context-free, $\mathsf{SUFFIX}(L)$ is also context-free. Let L be a context-free language recognized by PDA $M = (Q, \Sigma, \Gamma, \delta, q_0\, F)$; to show that $\mathsf{SUFFIX}(L)$ is context-free, we construct a PDA $M' = (Q', \Sigma, \Gamma, \delta', q_0', F)$ that recognizes $\mathsf{SUFFIX}(L)$. We define the new set of states Q', transition function δ', and start state q_0':

- $Q' = Q \cup \hat{Q}$, where $\hat{Q} = \bigcup_{q \in Q}\{\hat{q}\}$. In other words, we add a new state $\hat{q}$ that corresponds (the relationship will be defined in the transition function) to each existing state q; the new and existing states make up the set of states of M'. We will call $\hat{q}$ the corresponding state of q for all $q \in Q$.
- $\delta' : Q' \times \Sigma_\varepsilon \times \Gamma_\varepsilon \to \mathcal{P}(Q \times \Gamma_\varepsilon)$ is defined as:
  - For all $q \in Q$, $a \in \Sigma_\varepsilon$, and $b \in \Gamma_\varepsilon$, $\delta'(q, a, b) = \delta(q, a, b)$. In other words, the original transition function is copied over to our new transition function.
  - For all $q \in Q$, $a \in \Sigma_\varepsilon$, and $b \in \Gamma_\varepsilon$, $\delta'(\hat{q}, \varepsilon, b) = \{(\hat{r}, c) \mid (r, c) \in \delta(q, a, b)\}$. In other words, for each transition from state q that reads a, pushes b onto the stack, goes to state r, and pops c from the stack, the corresponding state $\hat{q}$ does not consume any input symbol, pushes b onto the stack, goes to the corresponding state $\hat{r}$, and pops c from the stack. This effectively performs what M would do upon seeing symbol a, but without actually consuming such symbol.
  - For all $q \in Q$, $a \in \Sigma_\varepsilon$, and $b \in \Gamma_\varepsilon$, $\delta'(\hat{q}, \varepsilon, \varepsilon) = \{(q, \varepsilon)\}$. In other words, the transition from the corresponding state $\hat{q}$ to the existing state q can be performed without consuming a symbol or pushing/popping from the stack.
- $q_0' = \hat{q_0}$. In other words, the new start state is the corresponding state of q_0.

At a high level, we are creating a duplicate version of M where each transition does not read in an input symbol, and creating a transition from each state in the duplicate M to its corresponding state in the original M. Now, we show that $v \in \mathsf{SUFFIX}(L)$ if and only if M' accepts v.

- If $v \in \mathsf{SUFFIX}(L)$, then there is some string u such that $uv \in L$. Since M is the PDA that recognizes L, then M accepts uv; there exists an accepting computation path from q_0 that ends up in an accepting state after reading uv. To show that M' accepts v, we show that there exists some computation path that results in the acceptance of v. From the start state $q_0' = \hat{q_0}$, we can take the transitions $\delta'(\hat{q}, \epsilon, b)$ that correspond to the transitions that would occur in the previously mentioned accepting computation path after reading in u and push/pop from the stack accordingly (without reading any input symbols). After taking these transitions, the state of our current computation is the following: while our stack is identical to the resulting stack if M were to run on u along the accepting computation path, we are currently at the corresponding state $\bar{q}$ to the state q that M would currently be at. We then take the transition $\delta'(\hat{q}, \varepsilon, \varepsilon)$ to reach the existing state q. From here, we can read in the input string v and take the transitions $\delta'(q, a, b)$ that are along the accepting computation path. Therefore, if $v \in \mathsf{SUFFIX}(L)$, M' accepts v.
- If M' accepts v, then some computation path must end up in an accepting state. Each computation path starts by running some number transitions from the corresponding states before eventually getting to an existing state $q \in Q$; upon arriving at this existing state q, the computation path follows the transitions of the original PDA M. This means that there exists some string u such that if we read u first in M and then read in v, then M will accept. Therefore, $uv \in L$ and $v \in \mathsf{SUFFIX}(L)$.

6. **Challenge:** Show that the complement of the language

$$L = \{ww \mid w \in \{a,b\}^*\}$$

is context-free.

Solution

First note that $\overline{L}$ is the union of two languages L_1, L_2 where $L_1 = \{xy \mid x, y \in \{a,b\}, |x| = |y|, x \neq y\}$ and $L_2 = \{w \mid |w| \equiv 1 \bmod 2\}$ (i.e., the string is of odd length). We show L_1 and L_2 are context free through defining CFGs G_1 and G_2 where $L(G_1) = L_1$ and $L(G_2) = L_2$.

G_1:

$$\begin{aligned} S_1 &\to AB \mid BA \\ A &\to a \mid aAa \mid aAb \mid bAa \mid bAb \\ B &\to b \mid aBa \mid aBb \mid bBa \mid bBb \end{aligned}$$

We show that $L(G_1) = L_1$ through an iff relation.
We first show that if $w \in L(G)$, then $w \in L$. We can see that

$$L(G) = \{w_1 x w_2 v_1 y v_2 \mid |w_1| = |w_2| = k, |v_1| = |v_2| = l, x \neq y\}$$

directly as a result of how the rules are defined. We can see that, x and y will be at the same place in the first and second half of the string, thus $w \in L$.
We then show that if $w \in L$ then $w \in L(G)$. Since we have $w = xy$ where $x \neq y$, x and y must differ at at least one character; we derive the rest of the string using the production rules $A \to aAa \mid aAb \mid bAa \mid bAb$ and $B \to aBa \mid aBb \mid bBa \mid bBb$, and derive the distinct character through the rules $A \to a$ and $B \to b$.

G_2:

$$\begin{aligned} S_2 &\to aX \mid bX \\ X &\to aS_2 \mid bS_2 \mid \varepsilon \end{aligned}$$

Informal justification: the grammar only generates strings of odd length since we alternate between containing the variables S_2 and X, and only X can end the derivation (deriving ε. Thus, we can union together the two CFLs L_1 and L_2 to yield $\overline{L}$, which proves that $\overline{L}$ is context-free.