COMP 2805 — Solutions Assignment 3

Question 1: (**6+7+7 marks**) Give context-free grammars that generate the following languages. In all cases, the set Σ of terminals is equal to $\{0, 1\}$. For each case, justify your answer.

1. $\{w : w$ contains at least three 1s$\}$.
2. $\{w : w$ starts and ends with the same symbol$\}$.
3. $\{w :$ the length of w is odd and its middle symbol is 0$\}$.

Solution: First, we do

$$\{w : w \text{ contains at least three 1s}\}.$$

Observe that this language is regular. Here is a DFA that accepts it: There are four states A_0, A_1, A_2, and A_3, where A_0 is the start state and A_3 is the accept state.

- When in state A_0 and read 0: go to state A_0
- When in state A_0 and read 1: go to state A_1
- When in state A_1 and read 0: go to state A_1
- When in state A_1 and read 1: go to state A_2
- When in state A_2 and read 0: go to state A_2
- When in state A_2 and read 1: go to state A_3
- When in state A_3 and read 0: go to state A_3
- When in state A_3 and read 1: go to state A_3

In class, it was shown how to convert a DFA to a context-free grammar:

- Set of variables: $\{A_0, A_1, A_2, A_3\}$
- Start variable: A_0
- Set of terminals: $\{0, 1\}$
- Rules:

$$\begin{array}{lll} A_0 \to 0A_0 & A_0 \to 1A_1 & \\ A_1 \to 0A_1 & A_1 \to 1A_2 & \\ A_2 \to 0A_2 & A_2 \to 1A_3 & \\ A_3 \to 0A_3 & A_3 \to 1A_3 & A_3 \to \epsilon \end{array}$$

Here is another explanation that this grammar generates the language:

- From A_3, we can generate all binary strings.
- From A_2, we can generate all strings that contain at least one 1.
- From A_1, we can generate all strings that contain at least two 1s.
- From A_0, we can generate all strings that contain at least three 1s.

By the way, here is another grammar that generates the same language (S is the start variable):

$$\begin{aligned} S &\rightarrow T1T1T1T \\ T &\rightarrow \epsilon|0T|1T \end{aligned}$$

From S, we can derive the string $T1T1T1T$, whereas from T, we can derive all binary strings. Hence, from S, we can derive all strings of the form $u1v1w1x$, where u, v, w, and x are binary strings. These are exactly the binary strings that contain at least three 1s.

Next, we do

$$\{w : w \text{ starts and ends with the same symbol}\}.$$

This language is also regular. So one solution is to give a DFA that accepts the language, and then to convert it to a context-free grammar. Here is a direct solution:

- Variables: S and T
- Start variable: S
- Terminals: 0 and 1
- Rules:

$$\begin{aligned} S &\rightarrow 0|1|0T0|1T1 \\ T &\rightarrow \epsilon|0T|1T \end{aligned}$$

Here is the explanation why this works: From T, we can derive all binary strings. From S, we can derive the strings 0, 1, $0T0$, and $1T1$. Hence, from S, we can derive the following strings:

- 0,
- 1,
- $0(0 \cup 1)^*0$,
- $1(0 \cup 1)^*1$.

This is exactly the language we want.

Finally, we do

$$\{w : \text{ the length of } w \text{ is odd and its middle symbol is } 0\}.$$

- Variable: S
- Start variable: S
- Terminals: 0 and 1
- Rules: $S \to 0|0S0|0S1|1S0|1S1$

Here is the explanation why this works: Without using the rule $S \to 0$, we can derive from S all strings of the form uSv, where u and v have the same length.

If we add the rule $S \to 0$, then we can only replace, in the string uSv, the symbol S by 0. So we can generate all strings of the form $u0v$, where u and v have the same length. This is exactly the language we are supposed to generate.

Question 2: (**20 marks**) Let $G = (V, \Sigma, R, S)$ be the context-free grammar, where $V = \{A, B, S\}$, $\Sigma = \{0, 1\}$, S is the start variable, and R consists of the rules

$$\begin{array}{lcl} S & \to & 0S|1A|\epsilon \\ A & \to & 0B|1S \\ B & \to & 0A|1B \end{array}$$

Define the following language L:

$$L := \{w \in \{0,1\}^* : \begin{array}{l} w \text{ is the binary representation of a non-negative integer} \\ \text{that is divisible by three } \} \cup \{\epsilon\} \end{array}$$

Prove that $L = L(G)$. (*Hint:* The variables S, A, and B are used to remember the remainder after division by three.)

Solution: By looking at the rules, you will notice that, from the start variable S, we can derive the empty string ϵ, and strings of the form wS, wA, and wB, where w is a non-empty binary string.

For any non-empty binary string w, let n_w be the non-negative integer whose binary representation is w. For example,

- if $w = 1011$, then $n_w = 1 + 2 + 8 = 11$, and
- if $w = 0001011$, then $n_w = 1 + 2 + 8 = 11$.

We claim the following: Let w be a non-empty binary string. Then the following holds:

1. $S \stackrel{*}{\Rightarrow} wS$ if and only if $n_w \equiv 0 \bmod 3$.

2. $S \stackrel{*}{\Rightarrow} wA$ if and only if $n_w \equiv 1 \bmod 3$.

3. $S \stackrel{*}{\Rightarrow} wB$ if and only if $n_w \equiv 2 \bmod 3$.

If a non-empty string is in the language $L(G)$ of the grammar G, then, by definition, $S \stackrel{*}{\Rightarrow} w$. Since w does not contain any variable, the *last* step in the derivation of w must use the rule $S \to \epsilon$. Therefore, the derivation has the form

$$S \stackrel{*}{\Rightarrow} wS \Rightarrow w.$$

If we assume that the above claim is true, then we see that the following holds, for any non-empty binary string w:

$$\begin{array}{lll} w \in L(G) & \text{if and only if} & S \stackrel{*}{\Rightarrow} wS \Rightarrow w \\ & \text{if and only if} & S \stackrel{*}{\Rightarrow} wS \\ & \text{if and only if} & n_w \equiv 0 \bmod 3 \\ & \text{if and only if} & w \in L. \end{array}$$

In other words, if we can prove the claim made above, then we have also shown that $L = L(G)$.

It remains to prove the claim. The proof is by induction on the length of the string w. For the basis of the induction, we assume that $|w| = 1$. Then $w = 0$ or $w = 1$.

- If $w = 0$, then $n_w = 0$. We have $S \stackrel{*}{\Rightarrow} 0S = wS$ and $n_w \equiv 0 \bmod 3$.
- If $w = 1$, then $n_w = 1$. We have $S \stackrel{*}{\Rightarrow} 1A = wA$ and $n_w \equiv 1 \bmod 3$.

Hence, for the base case, the claim is true.

Let w be a non-empty binary string. The induction hypothesis is that the claim is true for w. What do we have to show?

- Let $w' = w0$, i.e., w' is the string obtained by adding 0 to the end of w. Then we have to prove that the claim holds for the string w'. Observe that $n_{w'} = 2n_w$.
- Let $w'' = w1$, i.e., w'' is the string obtained by adding 1 to the end of w. Then we have to prove that the claim holds for the string w''. Observe that $n_{w''} = 2n_w + 1$.

We next observe that

- $n_w \equiv 0 \bmod 3$ if and only if $n_{w'} \equiv 0 \bmod 3$,
- $n_w \equiv 1 \bmod 3$ if and only if $n_{w'} \equiv 2 \bmod 3$,
- $n_w \equiv 2 \bmod 3$ if and only if $n_{w'} \equiv 1 \bmod 3$,
- $n_w \equiv 0 \bmod 3$ if and only if $n_{w''} \equiv 1 \bmod 3$,

- $n_w \equiv 1 \bmod 3$ if and only if $n_{w''} \equiv 0 \bmod 3$,
- $n_w \equiv 2 \bmod 3$ if and only if $n_{w''} \equiv 2 \bmod 3$.

Using these observations, and using the induction hypothesis, if follows that

$$\begin{aligned} S \stackrel{*}{\Rightarrow} w'S \quad & \text{if and only if} \quad S \stackrel{*}{\Rightarrow} wS \\ & \text{if and only if} \quad n_w \equiv 0 \bmod 3 \\ & \text{if and only if} \quad n_{w'} \equiv 0 \bmod 3. \end{aligned}$$

In a similar way, we obtain

$$\begin{aligned} S \stackrel{*}{\Rightarrow} w'A \quad & \text{if and only if} \quad S \stackrel{*}{\Rightarrow} wB \\ & \text{if and only if} \quad n_w \equiv 2 \bmod 3 \\ & \text{if and only if} \quad n_{w'} \equiv 1 \bmod 3. \end{aligned}$$

From these two examples, you can verify the remaining cases:

1. $S \stackrel{*}{\Rightarrow} w'B$ if and only if $n_{w'} \equiv 2 \bmod 3$,
2. $S \stackrel{*}{\Rightarrow} w''S$ if and only if $n_{w''} \equiv 0 \bmod 3$,
3. $S \stackrel{*}{\Rightarrow} w''A$ if and only if $n_{w''} \equiv 1 \bmod 3$,
4. $S \stackrel{*}{\Rightarrow} w''B$ if and only if $n_{w''} \equiv 2 \bmod 3$.

Question 3: (**6+7+7 marks**) Let A and B be context-free languages over the same alphabet Σ.

(3.1) Prove that the union $A \cup B$ of A and B is also context-free.
(3.2) Prove that the concatenation AB of A and B is also context-free.
(3.3) Prove that the star A^* of A is also context-free.

Solution: Since A is context-free, there is a context-free grammar $G_1 = (V_1, \Sigma, R_1, S_1)$ that generates A. Similarly, since B is context-free, there is a context-free grammar $G_2 = (V_2, \Sigma, R_2, S_2)$ that generates B. We assume that $V_1 \cap V_2 = \emptyset$. (If this is not the case, then we rename the variables of G_2.)

First, we show that $A \cup B$ is context-free. Let $G = (V, \Sigma, R, S)$ be the context-free grammar, where

- $V = V_1 \cup V_2 \cup \{S\}$, where S is a new variable, which is the start variable of G,
- $R = R_1 \cup R_2 \cup \{S \to S_1 | S_2\}$.

From the start variable S, we can derive the strings S_1 and S_2. From S_1, we can derive all strings of A, whereas from S_2, we can derive all strings of B. Hence, from S, we can derive all strings of $A \cup B$. In other words, the grammar G generates the union of A and B. Therefore, this union is context-free.

Next, we show that AB is context-free. Let $G = (V, \Sigma, R, S)$ be the context-free grammar, where

- $V = V_1 \cup V_2 \cup \{S\}$, where S is a new variable, which is the start variable of G,
- $R = R_1 \cup R_2 \cup \{S \to S_1 S_2\}$.

From the start variable S, we can derive the string $S_1 S_2$. From S_1, we can derive all strings of A, whereas from S_2, we can derive all strings of B. Hence, from S, we can derive all strings of the form uv, where $u \in A$ and $v \in B$. In other words, the grammar G generates the concatenation of A and B. Therefore, this concatenation is context-free.

Finally, we show that A^* is context-free. Let $G = (V, \Sigma, R, S)$ be the context-free grammar, where

- $V = V_1$,
- $S = S_1$ is the start variable (hence, we do not introduce a new start variable),
- $R = R_1 \cup \{S \to \epsilon | SS\}$.

From the start variable S, we can derive all strings S^n, where $n \geq 0$. From S, we can derive all strings of A. Hence, from S, we can derive all strings of the form $u_1 u_2 \ldots u_n$, where $n \geq 0$, and each string u_i $(1 \leq i \leq n)$ is in A. In other words, the grammar G generates the star of A. Therefore, A^* is context-free.

Question 4: (**10+10 marks**) Give (deterministic or nondeterministic) pushdown automata that accept the following languages.

1. $\{w \in \{0,1\}^* : w \text{ contains more 1s than 0s}\}$.
2. $\{w \in \{0,1\}^* : w \text{ is a palindrome}\}$. (A string w is a palindrome if $w = w^R$, i.e., reading w from left to right gives the same result as reading w from right to left. The empty string ϵ is a palindrome.)

Solution: I will give two deterministic pushdown automata for the language

$$\{w \in \{0,1\}^* : w \text{ contains more 1s than 0s}\}.$$

Here is the first solution. The idea is to walk along the input string from left to right, and use the stack to keep track of the number of 1s minus the number of 0s seen so far. (How do

we do this: for each 1, push a symbol S onto the stack; for each 0, pop the top symbol S.) The problem is that even though if the input string is in the language, an initial segment of the string may have more 0s than 1s. In this case, the stack would become empty before we have seen the entire string. So this doesn't work. The next idea is to use the stack to keep track of the number of 0s minus the number of 1s seen so far. This leads to a similar problem as above. The trick is to combine these two ideas.

- Tape alphabet $\Sigma = \{0, 1\}$.
- Stack alphabet $\Gamma = \{\$, S\}$.

We use three states:

- q_0: It is the start state. If we are in this state, then
  - the number of 0s seen is at least the number of 1s seen, and
  - the number of S-symbols on the stack is equal to the number of 0s seen minus the number of 1s seen.
- q_1: If we are in this state, then
  - the number of 1s seen is at least the number of 0s seen, and
  - the number of S-symbols on the stack is equal to the number of 1s seen minus the number of 0s seen.
- q_2: If we are in this state, then we have read the entire input string (so the tape head is on the blank symbol immediately to the right of the input string). If we are in this state, then we know that the input string has more 1s than 0s (so the string belongs to the language). What remains to be done is to pop all symbols from the stack.

The instructions are as follows.

- $q_0 0\$ \to q_0 R\$S$
- $q_0 0S \to q_0 RSS$
- $q_0 1\$ \to q_1 R\$S$
- $q_0 1S \to q_0 R\epsilon$
- $q_0 \square\$ \to q_0 N\$$
  - Explanation: We have read the entire string; since the stack does not contain any S-symbols, the string contains as many 1s as it contains 0s. Therefore, the string is not in the languge. In this instruction, we don't make any changes, so we loop forever and don't accept the string.

- $q_0 \square S \to q_0 N S$
  - Explanation: We have read the entire string; since we are in state q_0 and the stack contains at least one S-symbol, the string contains more 0s than 1s. Therefore, the string is not in the languge. In this instruction, we don't make any changes, so we loop forever and don't accept the string.
- $q_1 0 \$ \to q_0 R \$ S$
- $q_1 0 S \to q_1 R \epsilon$
- $q_1 1 \$ \to q_1 R \$ S$
- $q_1 1 S \to q_1 R S S$
- $q_1 \square \$ \to q_1 N \$$
  - Explanation: We have read the entire string; since the stack does not contain any S-symbols, the string contains as many 1s as it contains 0s. Therefore, the string is not in the languge. In this instruction, we don't make any changes, so we loop forever and don't accept the string.
- $q_1 \square S \to q_2 N \epsilon$
  - Explanation: We have read the entire string; since we are in state q_1 and the stack contains at least one S-symbol, the string contains more 1s than 0s. Therefore, the string is in the languge. We switch to state q_2.
- $q_2 \square S \to q_2 N \epsilon$
- $q_2 \square \$ \to q_2 N \epsilon$
  - Explanation: Now the stack is empty, so we accept the string.

Here is the second solution. The idea is similar, but now we use only two states. In the first solution, we used S-symbols to keep track of the absolute value of the difference in 1s and 0s seen so far. We used two states to indicate whether we had seen more 1s than 0s, or more 0s than 1s.

In this second solution, the stack will store only 0s (plus \$ at the bottom), if we have seen more 0s than 1s. It will store only 1s (plus \$ at the bottom), if we have seen more 1s than 0s.

- Tape alphabet $\Sigma = \{0, 1\}$
- Stack alphabet $\Gamma = \{\$, 0, 1\}$.

We use two states:

- q: This is the start state. If we are in this state, then the following holds: Let a be the number of 1s read so far, let b be the number of 0s read so far.

  If $a \geq b$, then the stack contains $a - b$ many 1s (plus \$ at the bottom); the stack does not contain any 0s.

  If $a \leq b$, then the stack contains $b - a$ many 0s (plus \$ at the bottom); the stack does not contain any 1s.

- q': If we are in this state, then we have read the entire input string w, and we know that w has more 1s than 0s. The tape head is on the blank symbol immediately to the right of w. The stack contains only 1s plus the \$ at the bottom. The purpose of this state is to make the stack empty (because we want to accept the string).

Here are the instructions:

- $q0\$ \to qR\$0$
- $q00 \to qR00$
- $q01 \to qR\epsilon$
- $q1\$ \to qR\$1$
- $q10 \to qR\epsilon$
- $q11 \to qR11$
- $q\Box\$ \to qN\$$
  - Explanation: The string contains equal number of 1s and 0s, so we loop forever.
- $q\Box 0 \to qN\$$
  - Explanation: The string contains more 0s than 1s, so we loop forever.
- $q\Box 1 \to q'N\epsilon$
- $q'\Box 1 \to q'N\epsilon$
- $q'\Box\$ \to q'N\epsilon$
  - Explanation: The stack is empty now, so we terminate and accept.

Finally, we give a nondeterministic pushdown automaton for the language

$$\{w \in \{0,1\}^* : w \text{ is a palindrome}\}.$$

We have to be careful:

- If a palindrome is of odd length, then it can be written as $u0v$ or as $u1v$, where v is the reverse of u.
- If a palindrome is of even length, then it can be written as uv, where v is the reverse of u.

At the start, we "guess" whether the input string has odd or even length.

If we guessed for odd length, then we do the following: While walking along the string from left to right, we "guess" that we have reached the middle symbol. All symbols to the left of the middle are pushed onto the stack. After we have reached the middle, we check if the contents of the stack is the same as the remaining part of the input string.

If we guessed for even length, then we do the following: While walking along the string from left to right, we "guess" that we have just entered the second half of the input string. All symbols in the first half are pushed onto the stack. After we have entered the second half, we check if the contents of the stack is the same as the remaining part of the input string.

- tape alphabet $\Sigma = \{0, 1\}$ and
- stack alphabet $\Gamma = \{\$, 0, 1\}$.

We will use five states:

- q_0: start state.
- q_1: we have guessed that the input string has odd length; we have not guessed yet the position of the middle symbol.
- q_2: we have guessed that the input string has odd length; we have guessed already the position of the middle symbol.
- q_3: we have guessed that the input string has even length; we have not guessed yet that we have entered the second half of the input string.
- q_4: we have guessed that the input string has even length; we have guessed already that we have entered the second half of the input string.

Here are the instructions:

- $q_0 0\$ \to q_1 N\$$
- $q_0 0\$ \to q_3 N\$$
- $q_0 1\$ \to q_1 N\$$
- $q_0 1\$ \to q_3 N\$$
- $q_0 \square\$ \to q_0 N\epsilon$ (input string is empty, therefore accept)

- $q_1 0\$ \rightarrow q_1 R\$0$
- $q_1 0\$ \rightarrow q_2 R\$$
- $q_1 1\$ \rightarrow q_1 R\$1$
- $q_1 1\$ \rightarrow q_2 R\$$
- $q_1 00 \rightarrow q_1 R00$
- $q_1 00 \rightarrow q_2 R0$
- $q_1 01 \rightarrow q_1 R10$
- $q_1 01 \rightarrow q_2 R1$
- $q_1 10 \rightarrow q_1 R01$
- $q_1 10 \rightarrow q_2 R0$
- $q_1 11 \rightarrow q_1 R11$
- $q_1 11 \rightarrow q_2 R1$
- $q_1 \square\$ \rightarrow q_1 N\$$ (loop forever)
- $q_1 \square 0 \rightarrow q_1 N0$ (loop forever)
- $q_1 \square 1 \rightarrow q_1 N1$ (loop forever)
- $q_2 0\$ \rightarrow q_2 N\$$ (loop forever)
- $q_2 1\$ \rightarrow q_2 N\$$ (loop forever)
- $q_2 00 \rightarrow q_2 R\epsilon$
- $q_2 01 \rightarrow q_2 N1$ (loop forever)
- $q_2 10 \rightarrow q_2 N0$ (loop forever)
- $q_2 11 \rightarrow q_2 R\epsilon$
- $q_2 \square\$ \rightarrow q_2 N\epsilon$ (accept)
- $q_2 \square 0 \rightarrow q_2 N0$ (loop forever)
- $q_2 \square 1 \rightarrow q_2 N1$ (loop forever)
- $q_3 0\$ \rightarrow q_3 R\$0$
- $q_3 1\$ \rightarrow q_3 R\$1$

- $q_3 00 \rightarrow q_3 R00$
- $q_3 00 \rightarrow q_4 N0$
- $q_3 01 \rightarrow q_3 R10$
- $q_3 01 \rightarrow q_4 N1$
- $q_3 10 \rightarrow q_3 R01$
- $q_3 10 \rightarrow q_4 N0$
- $q_3 11 \rightarrow q_3 R11$
- $q_3 11 \rightarrow q_4 N1$
- $q_3 \Box\$ \rightarrow q_3 N\$$ (loop forever)
- $q_3 \Box 0 \rightarrow q_3 N0$ (loop forever)
- $q_3 \Box 1 \rightarrow q_3 N1$ (loop forever)
- $q_4 0\$ \rightarrow q_4 N\$$ (loop forever)
- $q_4 1\$ \rightarrow q_4 N\$$ (loop forever)
- $q_4 00 \rightarrow q_4 R\epsilon$
- $q_4 01 \rightarrow q_4 N1$ (loop forever)
- $q_4 10 \rightarrow q_4 N0$ (loop forever)
- $q_4 11 \rightarrow q_4 R\epsilon$
- $q_4 \Box\$ \rightarrow q_4 N\epsilon$ (accept)
- $q_4 \Box 0 \rightarrow q_4 N0$ (loop forever)
- $q_4 \Box 1 \rightarrow q_4 N1$ (loop forever)

Question 5: (**10+10 marks**) Prove that the following languages are not context-free:

1. $\{a^n b^n a^n b^n : n \geq 0\}$. The alphabet is $\{a, b\}$.
2. $\{w\#x :$ w is a substring of x; and $w, x \in \{a, b\}^*\}$.

   The alphabet is $\{a, b, \#\}$. The string *aba#abbababbb* is in the language, whereas the string *aba#baabbaabb* is not in the language.

Solution: First, we prove that the language

$$A = \{a^n b^n a^n b^n : n \geq 0\}$$

is not context-free.

Assume that A is context-free. By the pumping lemma, there is an integer $p \geq 1$, such that for all strings $s \in A$ with $|s| \geq p$, the following holds: We can write $s = uvxyz$, where

1. vy is non-empty,
2. vxy has length at most p,
3. the string $uv^i xy^i z$ is in A, for all $i \geq 0$.

Consider the pumping length p. We choose $s = a^p b^p a^p b^p$. Then s is a string in A, and the length of s is $4p$, which is at least p. So we can write $s = uvxyz$ such that 1., 2., and 3. above hold. Since $|vxy| \leq p$, there are three cases:

Case 1: vxy lives in the leftmost half of s, that is, it lives in $a^p b^p$.

Case 2: vxy lives in the rightmost half of s, that is, it lives in $a^p b^p$.

Case 3: vxy lives in the two middle quarters of s, that is, it lives in $b^p a^p$.

Since all these cases are very similar, I will only consider Case 1. (You should go through the other cases yourself.)

So we assume that vxy lives in the leftmost half of s. Consider the string $s' = uvvxyyz$. By the pumping lemma, s' is contained in A. The string s' contains the rightmost half of s. That is, s' ends with $a^p b^p$. The part of s' that is to the left of the rightmost half of s is not equal to $a^p b^p$. Therefore, the string s' is not in A. This is a contradiction. It follows that the language A is not context-free.

Next we prove that the language

$$B = \{w\#x : \ w \text{ is a substring of } x; \text{ and } w, x \in \{a, b\}^*\}$$

is not context-free. Remember that the alphabet for this language is equal to $\{a, b, \#\}$.

Observe the following: if w and x are equal, then the string $w\#x$ is in the language B.

Assume the language B is context-free. By the pumping lemma, there is an integer $p \geq 1$, such that for all strings s in B with $|s| \geq p$, the following holds: We can write $s = uvxyz$, where

1. vy is non-empty,
2. vxy has length at most p,
3. the string $uv^i xy^i z$ is in B, for all $i \geq 0$.

Consider the pumping length p. We choose $s = a^p b^p \# a^p b^p$. Then s is a string in B, and the length of s is $4p+1$, which is at least p. So we can write $s = uvxyz$ such that 1., 2., and 3. above hold.

Case 1: vxy is completely to the left of the #-symbol.

Consider the string $uvvxyyz$. In this string, the part to the left of the #-symbol is longer than the part to the right of the #-symbol. Hence, the left part cannot be a substring of the right part. Therefore, $uvvxyyz$ is not in B. But, by the pumping lemma, it must be in B. This is a contradiction.

Case 2: vxy is completely to the right of the #-symbol.

Consider the string uxz. In this string, the part to the right of the #-symbol is shorter than the part to the left of the #-symbol. Hence, the left part cannot be a substring of the right part. Therefore, uxz is not in B. But, by the pumping lemma, it must be in B. This is a contradiction.

Case 3: v or y contains the #-symbol.

Then the string $uvvxyyz$ contains two #-symbols, hence, it is not in B. But, by the pumping lemma, it must be in B. This is a contradiction.

Alternatively, the string uxz contains no #-symbol, hence, it is not in B. But, by the pumping lemma, it must be in B. This is a contradiction.

Case 4: x contains the #-symbol. (This is the remaining case.)

Since vxy has length at most p, this implies that v contains only b's, and y contains only a's. So we can write

$$v = b^k \text{ for some } k \geq 0$$

and

$$y = a^\ell \text{ for some } \ell \geq 0.$$

Since vy is not empty, at least one of k and ℓ must be strictly positive. Consider the string uxz. This string is equal to

$$uxz = a^p b^{p-k} \# a^{p-\ell} b^p.$$

By the pumping lemma, this string is in B, so $a^p b^{p-k}$ must be a substring of $a^{p-\ell} b^p$. This implies that $\ell = 0$ (otherwise, there are more a's in $a^p b^{p-k}$ than there are a's in $a^{p-\ell} b^p$).

Consider the string $uvvxyyz$. This string is equal to

$$uvvxyyz = a^p b^{p+k} \# a^{p+\ell} b^p.$$

By the pumping lemma, this string is in B, so $a^p b^{p+k}$ must be a substring of $a^{p+\ell} b^p$. This implies that $k = 0$ (otherwise, there are more b's in $a^p b^{p+k}$ than there are b's in $a^{p+\ell} b^p$).

So we have shown that both k and ℓ are zero. This is a contradiction.

Conclusion: In each of the four cases, we get a contradiction. Therefore, the language B is not context-free.