COL 352 Introduction to Automata and Theory of Computation Holi 2022

Problem Set 3

Aniket Gupta 2019CS10327

Harshil Vagadia 2019CS10356

Shrey Patel 2019CS10400

1. We say that a context-free grammar G is self-referential if for some non-terminal symbol X we have $X \to^* \alpha X \beta$, where $\alpha, \beta \neq \varepsilon$. Show that a CFG that is not self-referential is regular.

   Solution: We will show that a grammar that is not self-referential generated a finite language. This finite language will be trivially regular.

   Claim: The smallest parse tree of a string s will not contain any repeated non-terminal on any path from root to leaf.
   Proof: Assume by contradiction there is a repeated symbol X in a path. Then the topmost X generates another X. If no side branches exists between these two X, then the path between X can be removed without affecting s. This is contradiction to the fact that the given parse tree was smallest. If there exists side branches, then $X \to^* \alpha X \beta$ where at least one of α, β is non empty. This is contradiction to the fact that the grammar is not self-referential.

   If there are n non-terminals in the grammar, then the height of any parse tree is restricted to $n + 1$. If the maximum number of symbols in the right hand side of all derivation rule is b, then each non terminal in the parse tree will have at most b children. Thus the maximum length of the string generated will be b^n. Thus the language generated is finite which is also regular.

2. Prove that the class of context-free languages is closed under intersection with regular languages. That is, prove that if L_1 is a context-free language and L_2 is a regular language, then $L_1 \cap L_2$ is a context-free language. Do this by starting with a DFA.

Solution: Let P be a pushdown automata that recognizes the CFL L_1 and accepts by final states and D be a DFA that recognizes the regular language L_2. We will show that $L = L_1 \cap L_2$ is context free by constructing a pushdown automata that recognizes L.

The idea is to construct a PDA P' that simulates both the PDA P and DFA D together and accepts only if both P and D accepts.

Let $P = (Q, \sum, \Gamma, q_0, \delta, \perp, F)$ be the automata recognizing L_1 and $D = (Q', \sum, q_0', F', \delta')$ be the DFA recognizing L_2.

Define $P' = (Q'', \sum', \Gamma', q_0'', \delta'', \perp', F'')$ as:

- $Q'' = Q \times Q'$
- $\sum' = \sum$
- $\Gamma' = \Gamma$
- $q_0'' = (q_0, q_0')$
- $\forall\, p \in Q, q \in Q', a \in \Sigma, A \in \Gamma \cup \{\perp\}$
  $\delta''((p,q),a,A) = \{((p',q'),A') \mid (p',A') \in \delta(p,a,A),\ \delta'(q,a) = q'\}$
  $\delta''((p,q),\varepsilon,A) = \{((p',q),A') \mid (p',A') \in \delta(p,\varepsilon,A)\}$
- $\perp' = \perp$
- $F'' = F \times F'$

Now we will show that the pushdown automata P' defined above recognizes the language $L = L_1 \cap L_2$.

Claim 1: If $x \in \Sigma^*$ lies in $L = L_1 \cap L_2$, then PDA P' accepts x.
Proof: Let us assume that after reading first $i(0 \le i \le |x|)$ characters of the input x, the PDA P is in configuration $(q, x', t\tau), \tau \in \Gamma^*, t \in \Gamma \cup \{\epsilon\}$ and the DFA D is in state q'.
Now if, in the next step, the PDA P reads ϵ from the input strip and changes to configuration $(p, x', \tau'\tau), \tau', \tau \in \Gamma^*$, then by the definition of δ'', $((p,q'),\tau') \in \delta''((q,q'),\epsilon,t)$. Thus, $((q,q'),x',t\tau) \vdash ((p,q'),x',\tau'\tau)$ in PDA P'.
If PDA P reads the next character from the input strip and changes the configuration to $(q', x'', \tau'\tau)$ and DFA changes the state to p' on reading the next character, then by definition of δ'', $((p,p'),\tau') \in \delta''((q,q'),\epsilon,t)$. Thus, $((q,q'),x',t\tau) \vdash ((p,p'),x',\tau'\tau)$ in PDA P'.
Now, since PDA P accepts input x, thus $(q_0, x, \perp) \vdash^* (f, \epsilon, \tau), f \in F, \tau \in \Gamma^*$. Also, the DFA D accepts x, thus DFA accepts the input x after reaching a state $f' \in F'$ starting from state q_0'. By repeatedly applying the two points explained in the previous paragraphs, $((q_0,q_0'),x,\perp) \vdash^* ((f,f'),\epsilon,\tau), (f,f') \in F'', \tau \in \Gamma^*$ in PDA P'.
Thus, P' accepts string x.

Claim 2: If PDA P' accepts x, $x \in \Sigma^*$ then $x \in L_1 \cap L_2$.
Proof: If P' accepts string x, then $((q_0, q_0'), x, \perp) \vdash^* ((f, f'), \epsilon, \tau), (f, f') \in F'', \tau \in \Gamma^*$.
Note by definition of δ'', $((q, q'), ax', t\tau) \vdash ((p, p'), x', \tau'\tau)$ in P' implies $(q, x', t\tau) \vdash (p, x'', \tau'\tau)$ in P and $\delta'(q', a) = p'$ (here, $p, q \in Q; p', q' \in Q'; t \in \Gamma; \tau', \tau \in \Gamma^*$).
Now, if we keep applying this to every pair of consecutive configurations in the expanded form of $((q_0, q_0'), x, \perp) \vdash^* ((f, f'), \epsilon, \tau)$, we get $(q_0, x, \perp) \vdash^* (f, \epsilon, \tau)$ in P and that there is an accepting sequence a states that input x passes through in DFA D to reach the final state $f' \in F'$.
Thus, x lies in both L_1 and L_2.

Thus, by the above two claims, we proved that the PDA P' defined above accepts the language $L = L_1 \cap L_2$, where L_1 is CFL and L_2 is regular.

3. (**2 points**) Given two languages L, L', denote by

$$L||L' := \{x_1y_1x_2y_2 \dots x_ny_n \mid x_1x_2 \dots x_n \in L, y_1y_2 \dots y_n \in L'\}$$

.

Show that if L is a CFL and L' is regular, then $L||L'$ is a CFL by constructing a PDA for $L||L'$. Is $L||L'$ a CFL if both L and L' are CFLs? Justify your answer.

Solution: Let P be a pushdown automata that recognizes the CFL L and accepts by final states and D be a DFA that recognizes the regular language L'. We will show that $L'' = L||L'$ is context free by constructing a pushdown automata that recognizes L''.

The idea is to create a PDA P' that simulates the PDA P if it is reading a character at odd position and simulates the DFA D when reading a character at the even position in the input.
Let $P = (Q, \sum, \Gamma, q_0, \delta, \bot, F)$ be the automata recognizing L and $D = (Q', \sum, q_0', F', \delta')$ be the DFA recognizing L'.

Define $P' = (Q'', \sum', \Gamma', q_0'', \delta'', \bot', F'')$ as:

- $Q'' = Q \times Q' \times \{q_e, q_o\}$ (Note: here q_e and q_o represents that even and odd numbers of characters have been read so far respectively.)
- $\sum' = \sum$
- $\Gamma' = \Gamma$
- $q_0'' = (q_0, q_0', q_e)$
- $\forall\ p \in Q, q \in Q', a \in \Sigma, A \in \Gamma \cup \{\bot\}$
  $\delta''((p, q, q_e), a, A) = \{((p', q, q_o), A') \mid (p', A') \in \delta(p, a, A)\}$
  $\delta''((p, q, q_e), \varepsilon, A) = \{((p', q, q_e), A') \mid (p', A') \in \delta(p, \varepsilon, A)\}$
  $\delta''((p, q, q_o), a, A) = \{((p, q', q_e), A) \mid \delta'(q, a) = q'\}$
- $\bot'=\bot$
- $F'' = F \times F' \times \{q_e\}$

Now we will show that the pushdown automata P' defined above recognizes the language $L'' = L||L'$.

Claim 1: If $x||y \in \Sigma^*$ lies in $L'' = L||L'$, then PDA P' accepts $x||y$.
Proof: If $x||y \in L||L'$, then $x \in L$ and $y \in L'$. Now, note that transition function, δ'', first takes steps according to δ until it reads the first character. Then it reads the second character according to δ'. Again it reads the third character according to δ and so on. Thus, P' reads characters at even positions in the input according to the PDA P and characters at odd positions according to the DFA D. After completely reading characters at the even positions, PDA P will be in some final state $f \in F$. Similarly, after reading characters at the odd positions, DFA D will be in some final state $f' \in F'$. Thus, after reading $x||y$, PDA P' will be in some state $(f, f') \in F''$. Thus, PDA P' accepts $x||y$.

Claim 2: If PDA P' accepts $x||y$, $x||y \in \Sigma^*$ then $x||y \in L'' = L||L'$.
Proof: If we observe closely the definition of δ'' written above, we can find that if P' is in some state (q, q') after reading even number of characters, then it changes the state and the stack only according to transition function of P.
If $(q_0, q'_0) \rightarrow (q_1, q'_0) \rightarrow ...(q_m, q'_{|y|})$ is sequence of states followed while accepting input $x||y$ in P', then consider the sequence of states in which characters at even positions are read. This sequence of states effectively reads the string y and reaches some final state. Let this sequence of states be $(q_{i1}, q'_0) \rightarrow (q_{i2}, q'_1) \rightarrow ...(q_{i|y|}, q'_{|y|})$. Now, note that $q'_0 \rightarrow q'_1 \rightarrow ...q_{|y|}$ is an accepting sequence of states for the string y in DFA D.
Similarly, if we consider the sequence of states ans stack operations that are involved in reading characters at the odd position, we will effectively get a sequence of states and stack operations that accepts string x in PDA P. Note that this is possible only because the reading the characters at even positions does not change the stack content.
Thus, $x||y \in L''$.

Thus, by the above two claims, we proved that PDA P' defined above accepts the language $L'' = L||L'$.

If both L and L' are CFLs, then $L||L'$ is not always CFL..
Here, we give an example to prove our claim.
Consider $L = \{0^{2n}1^n \; |n \geq 0\}$ and $L' = \{1^n2^{2n}|n \geq 0\}$.
Note that L is a CFL with CFG:
$S \rightarrow 00S1|\epsilon$
Similarly, L' is a CFL with CFG:
$S' \rightarrow 1S'22|\epsilon$

Then $L||L' = \{(01)^n(02)^n(12)^n|n \geq 0\}$
Now, we can see that $L||L'$ is not CFG. We can prove this using the pumping lemma. Let us assume that the pumping length be p. Then, consider the string $s = (01)^p(02)^p(12)^p$. Now, for any partition $s = uvwxy$ such that $|vx| > 0$ and $|vwx| \leq p$, vwx will either completely in any one of $(01)^p, (02)^p$ and $(12)^p$ or vwx can lie across the boundary of $(01)^p(02)^p$ or $(02)^p(12)^p$.
In the first case, if it lies completely inside any one of the three parts, then we can pump down to get uwy. The part which contained v and x will not have the same number of pairs as in the other two parts.
In the second case, if vwx lies across the boundary of two regions, then again we can pump down to get uwy. In this case, the third region which didn't contain v or x will have more number of consecutive pairs.
Thus, we can say by pumping lemma that this language is not context free.

4. For $A \subseteq \Sigma^*$, define

$$cycle(A) = \{yx \mid xy \in A\}$$

For example if $A = \{aaabc\}$, then

$$cycle(A) = \{aaabc, aabca, abcaa, bcaaa, caaab\}$$

. Show that if A is a CFL then so is $cycle(A)$

Solution: Consider the PDA $P = (Q, \sum, \Gamma, q_0, \delta, \perp, F)$ for the language A such that P accepts a string if and only if it ends on an empty stack. Any PDA can be extended to such a special PDA.

Since $xy \in A$, after reading xy, the stack will be empty (and the PDA will be in accepting state). Now we are starting with y. The PDA starts with an empty stack and does not know about the stack that will be left after reading y. So we create a new PDA P' which initially "guesses" the stack left by x. It then runs exactly as P on y. Then it confirms the guesses by running on x.

More formally, P' has a marker symbol (say #) which separates the guesses from the actual stack. Start the stack only with #. Now guess a symbol (say A) by using an epsilon transition. Replace the top # with $A\#A$ (A is arbitrary). This can be done by adding the transitions $(q, \epsilon, \#, q, t\#t), \forall q \in Q, t \in \Gamma$ in the transition function of P i.e., δ. We make two copies of the guessed symbol above and below the marker symbol. The above copy will be used by y and the bottom copy is for confirming the guesses.
Also, we would have to guess the state in which the P will be after reading x. To make this guess, we add transitions $(q_0, \epsilon, \perp, q, \perp), \forall q \in Q$.
First, we make an epsilon transition to some state $q \in Q$ according to one of the transitions added above to guess the state in which the original automata P will be after reading x.

Now, P' starts consuming the string exactly as P. When we hit #, it implies that we have consumed the previously guessed symbol. This could mean one of the following two things:

(a) We need to make more guesses. For this add an ϵ transition that replaces # with $B\#B$ (B is arbitrary). Continue consuming the string. Note that the bottom copy of the guesses is still preserved and it is in the reverse order of the stack generated by x (supposedly).

(b) We may have reached end of y in which case we need to confirm the guesses. Note that we do not need to check if y ends in an accepting state or not because we defined P to accept on empty state. Also, we would have to make a guess (using an epsilon transition) if y has been read. In this case, the following steps are followed.

   i. After guessing the end of y, we make a transition to initial state q_0 using an epsilon transition to start reading x and confirming the initial guesses for the stack content. For this, we add the transitions $(q, \epsilon, \#, q_0, \#), \forall q \in Q$.

   ii. The only thing remaining is to confirm the guesses. This can be done by reading the remaining string and popping a symbol instead of pushing it if the symbol match with the symbol below # on the stack. For example, if we have to push $A \in \Gamma$ on the stack and the stack content is $\#A\gamma, \gamma \in \Gamma^*$, then we replace $\#A$ with $\#$ on the top of stack.

iii. If it does not, push the symbol on the stack. If P pops a symbol, P' also pops the symbol.

iv. If we end up with only $\#$ on the stack after reading the complete input, the run is accepted.

If the string ends with only $\#$ in the stack, then the string is accepted.

Claim: P' accepts $cycle(A)$
Proof: $\Leftarrow$ If $s = yx \in cycle(A)$ such that $xy \in A$, then there exists a run which guesses the stack left by x, then runs on y, reaches the marker symbol at the end of y, confirms the guesses by reading x and accepts on empty stack. Hence $cycle(A) \subseteq L(P')$.
$\Rightarrow$ If s is accepted by P', then there must be an accepting run. Partition $s = yx$ on this run such that P' starts confirming guesses after reading y. Then $xy \in A$ by the construction of P'. This means $s \in cycle(A)$ and $L(P') \subseteq cycle(A)$.

5. Let
$$A = \{wtw^R \mid w, t, \in \{0,1\}^* \text{ and } |w| = |t|\}$$
. Show that A is not a CFL.

Solution: We can prove this by using contrapositive of pumping lemma. Given any number p, consider the string $s = 0^{2p}(01)^p 0^{2p}$. Clearly $|s| \geq p$ and $s \in A$. Now consider any partition of $s = uvxyz$ such that $|vy| > 0$ and $|vxy| < p$. If $|vy|$ is not a multiple of 3, the string uv^2xy^2z will not be in A (as string cannot be divided into three equal parts). Hence we will consider $|vy| = 3r$. Consider the following cases on the relative position of vxy w.r.t s.

(a) If vxy lies to the left of the rightmost 1 in s, we can pump up the string (say by 2). In the string uv^2xy^2z, the $2p+1$th symbol from the left will be 0, but the $2p+1$th symbol from the right will be 1. Hence the pumped up string will not be in A.

(b) If vxy lies entirely to the right of the rightmost 1 in s, we can pump down the string (say to 0). In the string uv^0xy^0z, the $2p+1-|vy|$th symbol from the left will be 0, but the $2p+1-|vy|$th symbol from the right will be 1. Hence the pumped down string will not be in A.

(c) If vxy overlaps with the rightmost 1 in s, we have the following two cases:

   i. If y contains atleast one 0 after the rightmost 1 in s, we can pump up the string (say by 3). In the string uv^3xy^3z, the $2p+2$th symbol from the left will be 1, but the $2p+2$th symbol from the right will be 0. Hence the pumped up string will not be in A.

   ii. If y ends exactly at the rightmost 1 in s, we can pump up the string (say by 2). In the string uv^2xy^2z, the $2p+1$th symbol from the left will be 0, but the $2p+1$th symbol from the right will be 1. Hence the pumped up string will not be in A.

All partition of s satisfy one of the above cases. The above cases are exhaustive and in each of the cases there exists a i such that uv^ixy^iz is not in A. Hence by contrapositive of pumping lemma, A is not a context free language.

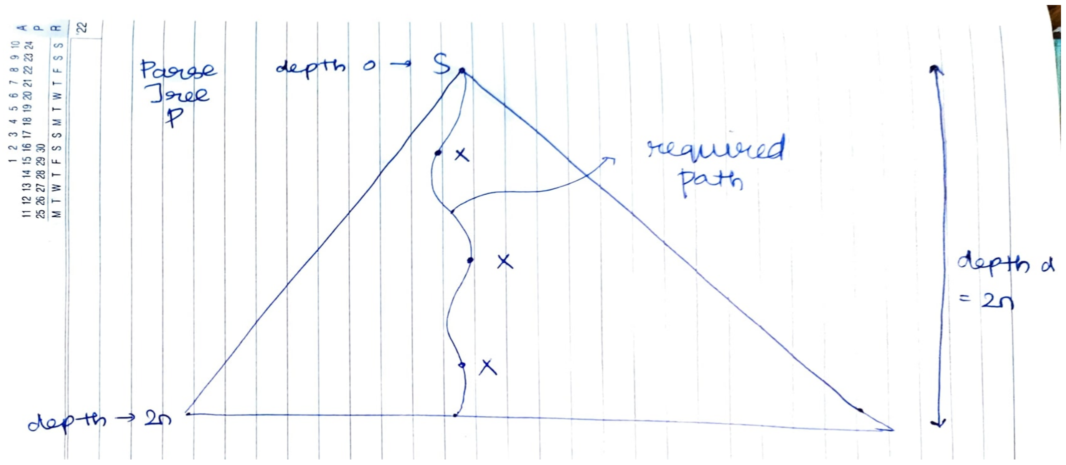
6. Prove the following stronger version of pumping lemma for CFLs: If A is a CFL, then there is a number k where if s is any string in A of length at least k then s may be divided into five pieces $s = uvxyz$, satisfying the conditions:

   - for each $i \geq 0$, $uv^ixy^iz \in A$
   - $v \neq \varepsilon$, and $y \neq \varepsilon$, and
   - $|vxy| \leq k$.

   Solution:
   Let A be a CFL and suppose G be its CFG in Chomsky Normal Form. Chomsky Normal form assures that every production can form at most two symbols (two variables) on its RHS. So, the branching factor of the parse tree of this grammar is 2. Let the number of non-terminals in G be n. Starting from depth 0(which contains the start symbol S of the CFG), the maximum length of the string which can be constructed from a parse tree of depth d is 2^d. In other words, to construct a string of length 2^d, the smallest depth of parse tree required is d.

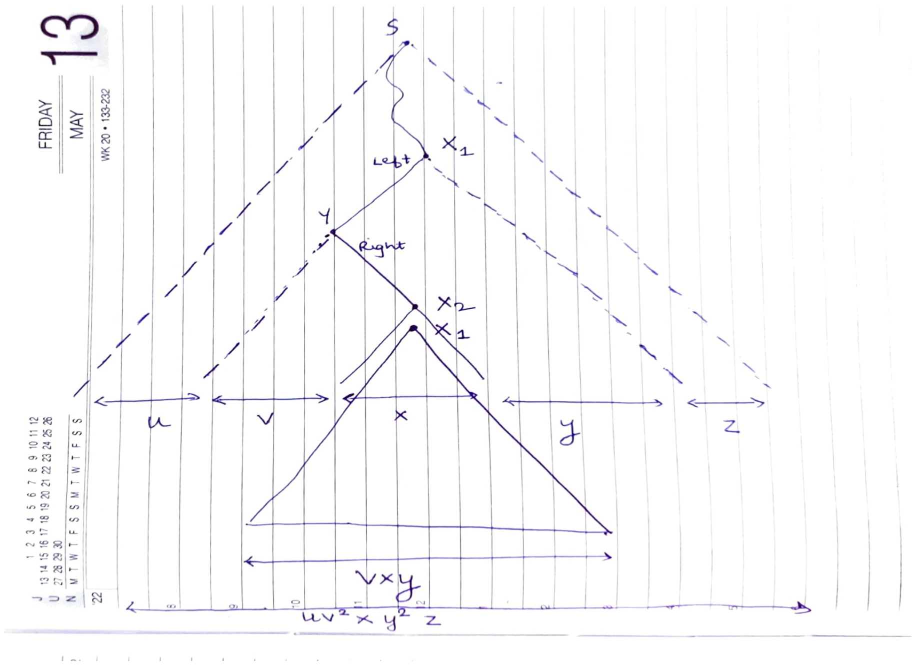
   Using the fact above, set the number k = 2^{2n}. So, any string s having length at least k and belonging to this language will require a parse tree of depth at least 2*n. Let this parse tree be P. Using this parse tree, we have to prove the three given conditions.

   Since P has a depth at least 2*n, there is a path of length at least 2*n (containing 2*n + 1 nodes). Then, by Pigeonhole principle, there is at least one non-terminal, say X, which repeats thrice on this path(proof by contradiction). And X cannot be S, since the start symbol never appears on the RHS in Chomsky Normal Form. The above structure is as shown below:



   Then on the basis of this path, and the configurations of the three Xs in the parse tree P, there are several cases, and we prove the above three conditions for each of them:

Case 1: When any two of the three Xs are in a zigzag fashion.
The above statement can be explained through subcases:
Subcase 1: If X2 is in the left subtree of X1, then in the subpath from X1 to X2, $\exists$ at least one non-terminal, say Y, such that X2 is present in the right subtree of Y.



The above figure summarises the structure of the parse tree in this case. We now prove the three points one by one:

(a) As shown in the figure, we can replace the tree rooted at X2 with the tree rooted at X1. Due to this, the string labelled as x will be replaced effectively by the string produced by X1, which is vxy. So, the final string is uv^2xy^2z. Thus, by successive applications of the above rule, we can obtain any string of the form uv^ixy^iz $\forall$ i $>$ 0. For i=0, i.e. for obtaining, the string uxz, we can just replace the tree rooted at X1 with the tree rooted at X2. So, the string vxy will be replaced with x (thus pumping down). So, we have proved that all strings of the form uv^ixy^iz $\forall$ i $\geq$ 0 can be generated from this parse tree by some manipulation.

(b) v is the string generated from the left subtree of Y as shown in the figure, while y is the string generated from the right subtree of X1. We know that both these subtrees are non-empty since the grammar is in Chomsky Normal Form, which assures that each internal node either splits into exactly two non-terminals(which is applicable in this case) or exactly one terminal. So, both v and y are non empty strings.

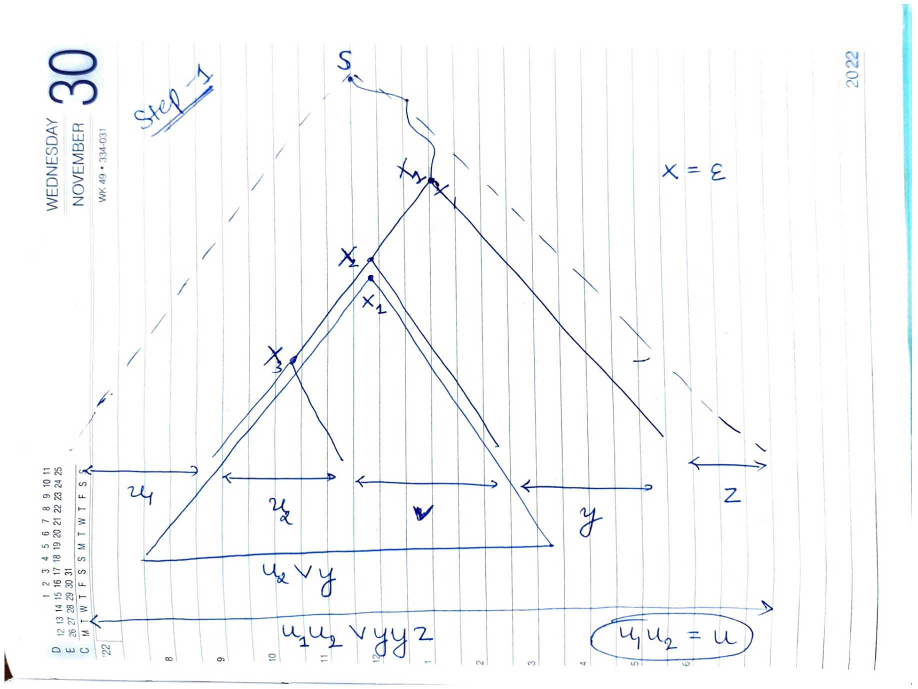
(c) Note that the string vxy has been generated from the tree rooted at X1. And this tree has a depth strictly less than that of the tree rooted at S. So, the maximum length of string which can be generated from X1 is less than 2^{2n} = k. So, $|vxy| \leq k$.

Subcase 2: If X2 is in the right subtree of X1, then in the subpath from X1 to X2, ∃ at least one non-terminal, say Y, such that X2 is present in the left subtree of Y.
This case is symmetric to subcase 1, and therefore the proof is also similar. The only difference here is that v will be produced from the left subtree of X1 while y will be produced from the right subtree of Y.

Case 2: When all three Xs are in a straight line.
Now, since we need to make both cases exhaustive, we needed a case such that none of the two out of three can form a zigzag structure as explained in the previous case. Then, we need all three Xs to be in a straight line in the sense that all the non-terminals in the path from X1 to X3 are either all left children or all right children of their parents. This can be shown using the following subcases:

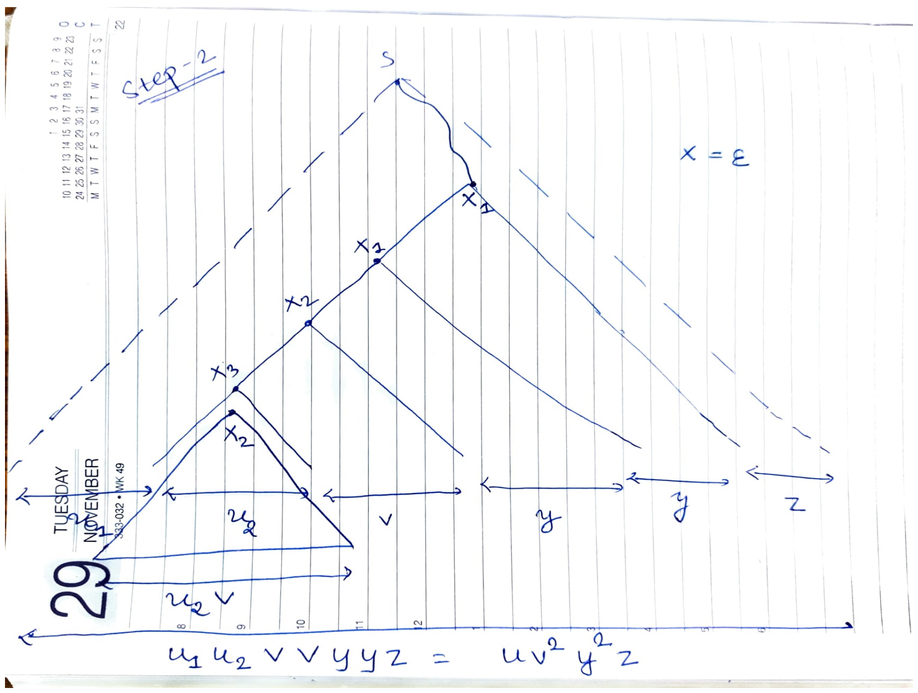
Subcase 1: All non-terminals on the path from X1 to X3 are all left children of their parents. Now, we prove the three points as in case 1(Note that we take x = ϵ in this case:

(a) We prove this part in two steps:



The above figure illustrates the straight line configuration of the Xs. Let the initial string be divided into parts u_1u_2vyz which is the same as $uvxyz$ since u = u_1u_2 and x = ϵ. *Note that there is nothing special about splitting u's instead of z's. None of the parts u1 or u2 will be altered. Here u is splitted since all Xs are tilted towards the left. If they were tilted towards right, as in subcase 2, then z will be splitted instead of u.*

Now, in the first step, we replace the tree rooted at X2 with the tree rooted at X1. Thus, the string labelled u_2v will be replaced by the string u_2vy. The final string at this stage is $u_1u_2vyyz = uvxyyz$.



In the next step, we replace the tree rooted at X3 with the tree rooted at X2. Thus the string labelled u_2 will be replaced by the string u_2v. So, the final string now will be $u_1u_2vvyyz = uvvxyyz = uv^2xy^2z$. The same process can be repeated to raise the strings v and y to higher powers. Thus, the same parse tree can be used to derive all strings of the form uv^ixy^iz ∀ i ≥ 1.

For i=0, at step 1, we can directly replace the tree rooted at X1 with the tree rooted at X3, and so the string u_2vy will be replaced by u_2 giving the final string $u_1u_2z = uxz$. So, we proved that all strings of the form uv^ixy^iz ∀ i ≥ 0 can be produced from the given CFG.

(b) Note that the string v is derived from the right subtree of X2 and y is derived from the right subtree of X1. Both these subtrees are non-empty since G is in Chomsky Normal Form(which says that no symbol other than S can go to an ϵ). Thus, both v and y are non-empty.

(c) The tree rooted at X1 has strictly lesser depth than the tree rooted at S. And the string $vy = vxy$ is a substring of the string produced by X1. So, the length of vxy is strictly less than $2^{2n} = k$. So, $|vxy| < k$.

Subcase 2: All non-terminals on the path from X1 to X3 are all right children of their parents.
Again, this subcase is symmetric to subcase 1. The only difference here is that instead of u, z will be now split into z_1z_2. So, the sequence of strings formed in this case will be $uvyz_1z_2$ to $uvvyz_1z_2$ to $uvvyyz_1z_2$. Similarly, now, v will be produced from the left subtree of X1 and y will be produced from the left subtree of X2. The rest of the proof is the same.

7. Give an example of a language that is not a CFL but nevertheless acts like a CFL in the pumping lemma for CFL (Recall we saw such an example in class while studying pumping lemma for regular languages).

Solution:
We need to find a non-context free language which satisfies the Pumping Lemma.

Observation 7.1:
Using the result proved in Q2, if L_1 is a CFL and L_2 is a regular language, then $L1 \cap L2$ is a CFL. Taking contrapositive, if $L_1 \cap L_2$ is not a CFL, then at least one of the following two things are true:

(a) L_1 is not a CFL.

(b) L_2 is not regular.

Now, choose $L_2 = \{01^*2^*3^*\}$. Clearly, L_2 is regular since it is a concatenation of Kleene stars of regular languages and we have proved that regular languages are closed under both operations.
Also, take $L_1 = \{0^a1^b2^c3^d | a \neq 1\} \cup \{01^b2^c3^d | 0 \leq b \leq c \leq d\}$.
Then, $L_3 = L_1 \cap L_2 = \{01^b2^c3^d | 0 \leq b \leq c \leq d\}$.

Claim 7.1: L_3 is not a CFL.
Proof of Claim 7.1:
We prove this by using the contrapositive of Pumping Lemma. Suppose the pumping length is p. Then choose the word s = $01^p2^p3^p$ (clearly, $|s|$ > p). Then, s $\in L_3$. Now, we need to prove that for all partitions $uvwxy$ ($|vx| > \epsilon$ and $|vwx| \leq$ p) $\exists$ k $\geq$ 0 such that s' = uv^kwx^ky $\notin L_3$. This proof can be divided into several cases:

(a) The repeating substrings i.e. v or x contains the letter 0. In this case, any value of k except 1 will change the number of 0's in the string. But, any word in L_3 must contain only one 0. Thus, s' $\notin L_3$.

(b) None of the repeating substrings i.e. v or x contain any letter 0. Then the entire substring vwx is contained in $1^p2^p3^p$. Note that $|vwx| \leq p$. This means that the string vwx either entirely contains only one letter, or it is present in the boundary between exactly two letters, i.e. either 1 and 2 or 2 and 3. In other words, the string vwx can never contain all the three letters 1,2 and 3 at once. So, we further split our proof into two subcases:

   i. When the string vwx contains a single letter out of 1,2 or 3. If it is contained within 1 then pumping up v or x will increase the number of 1's in the string. Thus, s' = $01^{p'}2^p3^p$ such that p' > p. Thus the number of 1's are greater than 3's in s'. So, s' $\notin L_3$. Instead, if vwx contains only 2. Then pumping v and x down would cause the number of 2's to decrease. So, the new string has greater number of 1's than 2's. Again, s' $\notin L_3$. This same argument can also be used when vwx contains only 3's. Thus, for all cases, s' = $uv^kwx^ky \notin L_3$

ii. When the string vwx contains exactly two letters either 1 and 2 or 2 and 3. Suppose if vwx contains both 1 and 2. Then by pumping up v and x, we obtain a string s' of the form $01^a2^b3^p$, where at least one of a or b is greater than p. This means that the number of at least one of the letters 1 or 2 is strictly greater than the number of 3's in s'. Thus, s' $\notin L_3$. Now, if vwx contains both 2 and 3. Then by pumping down v and x, we obtain a string s' of the form $01^p2^a3^b$, where at least one of a or b is strictly lesser than p. This means that the number of at least one of the letters 2 or 3 is lesser than the number of 1's in s'. Thus, s' $\notin L_3$.

So, from all the above cases, we proved that $\forall$ p $\geq$ 0, $\exists$ a word s $\in L_3$, such that for all partitions of s of the form $uwxyz$ where $|vx| > \epsilon$ and $vwx \leq p$, $\exists$ k $\geq$ 0 such that the word s' = $uv^kwx^ky \notin L_3$, which is precisely the contrapositive of the Pumping Lemma. So, L_3 is not a CFL.

We have proved that $L_1 \cap L_2$ is not a CFL. Then, from observation 7.1, either L_1 is not a CFL, or L_2 is not regular(or both, but at least one). But, we know that L_2 is regular, which means that L_1 is not a CFL. Now, if we prove that L_1 follows the pumping lemma, then we are done. L_1 is in that case, the example we are looking for.

Claim 7.2: L_1 follows the Pumping Lemma.
Proof of Claim 7.2:
Take p = 1. Then we need to prove that for all words s with length at least one, there exists a partition $uvwxy$ such that $|vx| > \epsilon$ and $|vwx| \leq 1$(this simplifies to either v or x containing a single letter and rest empty), $\forall$ k $\geq$ 0, the string s' = $uv^kwx^ky \in L_1$. There are several possible cases for the word s:

(a) When a < 2, i.e. the number of 0's in s is less than 2. In this case, take $u = \epsilon$, v = first letter of s, $w = x = \epsilon$ and y = rest of the string s. Then there are two subcases:

   i. a = 0, s would be of the form $1^b2^c3^d$. Then v = 1, 2 or 3 depending on which is the first letter. On pumping v, this letter will be multiplied. Consequently, s' = uv^kwx^ky is also of the form $1^b2^c3^d$. So, s' $\in L_1$ $\forall$ k $\geq$ 0.

   ii. a = 1, s would be of the form $01^b2^b3^b$. Then the first letter and therefore the string v is 0. On pumping the string v, the letter 0 will be multiplied. Consequently, s' = uv^kwx^ky is of the form $0^k1^b2^b3^b$. Note that since the number of 1, 2 and 3 are same, it doesn't matter even if k=1. Thus, s' $\in L_1$ $\forall$ k $\geq$ 0.

(b) When a $\geq$ 2. In this case, take $w = x = y = \epsilon$, v = last letter of the string s, u = rest of the string s. Again there are two subcases:

   i. When the string is of the form s = 0^a, which means there are only 0's in the string. Then u = 0^{a-1} and v = 0. After pumping v, s' = 0^{k+a-1}. So, s' $\in L_1$ $\forall$ k $\geq$ 0. Note that even when k + a = 2, i.e. s' = 0, the string is in L_1 since the number of 1, 2 and 3 are equal(i.e. all occurring zero times).

   ii. When the last letter is not a i.e. s is of the form $0^a1^b2^c3^d$ (b + c + d $\geq$ 1), then v = 1,2 or 3. After pumping v, the new string s' is also of the form $0^a1^b2^c3^d$ which is in L_1 since a > 2 as assumed earlier. So, s' $\in L_1$ $\forall$ k $\geq$ 0.

Note that in both the cases above, we set the v to be equal to a single letter and set w and x to be empty, which means that both the conditions $|vx| > \epsilon$ and $|vwx| \leq 1$ hold.

From the above cases, we proved that $\exists$ p(which we chose to be 1), such that all words of L_1 of length at least p are such that they can be partitioned into the form s = $uvwxy$ such that $|vx| > \epsilon$ and $|vwx| \leq p$, $\forall$ k $\geq$ 0, s' = uv^kwx^ky also is in L_1. So, L_1 follows pumping lemma.

Thus, L_1 is a non-context free language which follows the pumping lemma. Thus the pumping lemma is not a sufficient condition for checking if a language is context free.