Assignment No-04

Problem 1. Consider the language $L=\{a^n b^{n^2} \mid n \geq 0\}$. Use Parikh's Theorem or the Pumping Lemma for CFLs to show that L is not a CFL.

Solution:

Parikh's Theorem

We will show that the language $L=\{a^n b^{n^2} \mid n \geq 0\}$ is not CFL.

- Parikh's Theorem state that if a language L is CFL, then the set of its Parikh Vector (the counts of each symbol in its strings) is a semilinear Set. If A set $S \subseteq \mathbb{N}^k$ is semilinear if it can be expressed as a finite union of linear sets. In one dimension, semilinear sets are exactly the ultimately periodic sets.

→ For any string in L, of the form $a^n b^{n^2}$, the Parikh Vector is:

$$\Psi(a^n b^{n^2}) = (n, n^2).$$

Thus, the Parikh of L is

$$\Psi(L) = \{(n, n^2) : n \geq 0\}.$$

Now, Claim that $\Psi(L)$ is NOT Semilinear.

By contradiction, that $\Psi(L)$ were semilinear. Then, by definien, it would be finite union of linear set. (set of b-counts)

$$\{n^2 \mid n \geq 0\}$$

would have to be semilinear (i.e. ultimately periodic in case of subsets of $\mathbb{N}$). However, consider the set of perfect sq.

- The differences between consecutive perfect sq. are:

$$(n^2+1)^2 - n^2 = 2n+1,$$

which increases without bound.

- If $\{n^2 : n \geq 0\}$ were ultimately periodic, then beyond some point the differences between successive no. would be constant. Clearly, this is not the case.

Thus, $\{n^2 \mid n \geq 0\}$ is not semilinear.

Consequently, $\Psi(L)$ is not semilinear. By Parikh's Theorem, every context-free language has a semilinear Parikh image. Since we have shown that the Parikh image

$$\Psi(L) = \{(n, n^2) : n \geq 0\} \text{ is not semilinear,}$$

it follows that L cannot be a CFL.

BY PUMPING LEMMA

→ Suppose L is a CFL. Let K be the pumping constant. Consider the string $z = a^K b^{K^2}$. Then $z = uvwxy$ such that.

(i) $|vwx| \leq K$,

(ii) $vx \neq \epsilon$, and

(iii) $uv^i wx^i y \in L$ for all $i \geq 0$.

Suppose $v = a^\alpha b^\beta$ with $\alpha\beta \neq 0$. Then $uv^2wx^2y = ua^\alpha b^\beta a^\alpha b^\beta wx^2y$ contains at least two runs of a's and hence cannot be in L.

Thus, 'v' is not of this form. Similarly, neither is x.
Thus 'v' and 'x' can only contain either all a's or all b's. If they were both all a's, then uv^2wx^2y would contain more than k a's but only k^2 b's. Similarly, if they were both all b's, then uv^2wx^2y would contain more than k^2 b's but only ka's.
Thus, $v = a^\alpha$ and $x = b^\beta$. Then $uv^2wx^2y = a^{k+\alpha} b^{k^2+\beta} \in L$. This gives $\beta = 2k\alpha + \alpha^2$. Also, $uwy = a^{k-\alpha} b^{k^2-\beta} \in L$. This gives $\beta = 2k\alpha - \alpha^2$.
Thus, $\alpha^2 = 0 \Rightarrow \alpha = 0$ and $\beta = 0$. But then $vx = \epsilon$, a Contradiction.
Thus, L is not a CFL.

10

Problem 2. Consider the CFG G below:

$$S \to XC \mid AY$$
$$X \to aXb \mid ab$$
$$Y \to bYc \mid bc$$
$$A \to aA \mid a$$
$$C \to cC \mid c$$

(a) Describe the language accepted by G.

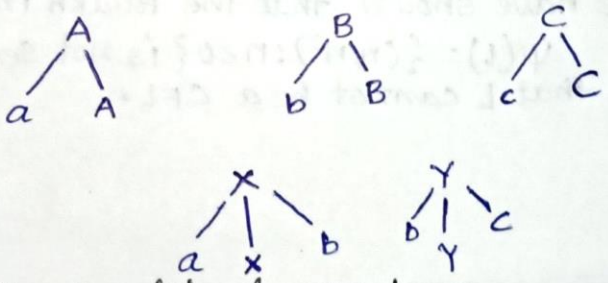
Solution: X produces strings of the form $a^n b^n$ where $n \geq 1$. C produces strings of the form c^m where $m \geq 1$. A produces strings of the form a^l where $l \geq 1$. Y produces strings of the form $b^s c^s$ where $s \geq 1$. From Z, we get either XC or AY. Thus we get either $a^n b^n c^m$ or $a^l b^m c^m$. The language generated by this grammar can be written as:

$$L(G) = \{a^l b^m c^n \mid l, m, n \geq 0 \text{ and } m = l \text{ or } m = n\}.$$

(b) Use the construction in Parikh's theorem to construct a semi-linear expression for $\psi(L(G))$. That is

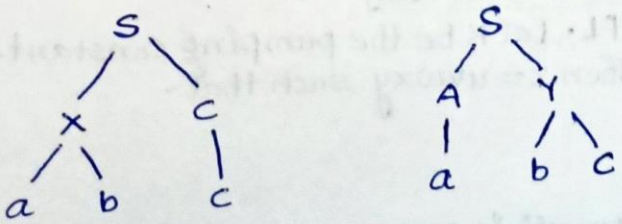
(i) Identify the basic pumps for G.

→ The basic pumps are $A \to aA$, $B \to bB$, $C \to cC$, $X \to aXb$ and $Y \to bYc$.



(ii) Identify the $\leq$-minimal parse trees.

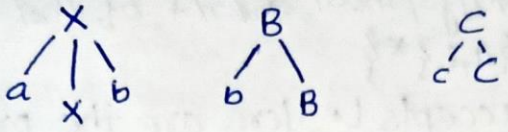
→ $S \to XC \Rightarrow abc$ & $S \to AY \Rightarrow abc$.



(iii) Use these to obtain an expression for $\psi(L(G))$.

→ The basic pumps, in the order they are listed, are of lengths 1, 1, 1, 2 and 2.

- For the first tree: $X \to aXb$, $B \to bB$ or $C \to cC$



- For the Second tree: $A \to aA$, $B \to bB$ or $Y \to bYc$ (similarly)

Thus, that bucket gives the same Parikh map.

Thus that Parikh map of the language is

$$\psi(L(G)) = 3 + \langle\langle 2, 1 \rangle\rangle = 3 + \langle\langle 1 \rangle\rangle$$

or Simply $\{3, 4, \ldots\}$

(c) Use the semi-linear expression obtained above to give a regular expression that is letter-equivalent to $L(G)$.

→ To obtain a letter equivalent regular expression, replace every 1 in the semi linear set by the corresponding letter. Thus, the regular expression that is letter equivalent to $L(G)$ is:

$$abc \cdot (ab+c)^* + abc \cdot (bc+a)^*$$

$$= abc \cdot ((ab+c)^* + (bc+a)^*)$$

(or)

$aaaa^*$ also has Parikh map $\{3, 4, \ldots\}$.

Problem 3. Give a PDA that accepts the language $\{a,b\}^* - \{ww \mid w \in \{a,b\}^*\}$.

Solution : We need to design a PDA for the language $\{a,b\}^* - \{ww \mid w \in \{a,b\}^*\}$ which represents all strings that aren't of the form ww.

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

Let 'G' be a CFG that accepts L. Here, are the productions of G:

$S \to AB \mid BA \mid A \mid B$

$A \to CAC \mid a$.

$B \to CBC \mid b$.

$C \to a \mid b$.

G generates:

1.) All strings of odd length (starting with $S \to A$ or $S \to B$)

2.) String of form $xayubv$ or $ubvxay$ $\quad x,y,u,v \in \{a,b\}^*$ $|x| = |y|$

$|u| = |v|$ (if we start from $S \to AB$ or $S \to BA$)

A generates strings of form xay where $|x| = |y|$

B generates of form ubv where $|u| = |v|$

- No string in 1.) can be of form ww as ww is an even length string.
- No string in 2.) can be of form ww as occurences of a & b are sep. by n/2 distance.

→ We can construct a NPDA M for CFG G by the following method: For each production $A \to CB, B_L \ldots B_K$ {Transition relation of NPDA} is $((q, C, A), (q, B_1, B_2, \ldots B_K))$.

The PDA for G is given by:

(i) $(S, \varepsilon, \vdash) \longrightarrow (S, AB)$

(ii) $(S, \varepsilon, \vdash) \longrightarrow (S, BA)$

(iii) $(S, \varepsilon, \vdash) \longrightarrow (S, A)$

(iv) $(S, \varepsilon, \vdash) \longrightarrow (S, B)$

(v) $(S, \varepsilon, A) \longrightarrow (S, CAC)$

(vi) $(S, a, A) \longrightarrow (S, \varepsilon)$

(vii) $(S, \varepsilon, B) \longrightarrow (S, CBC)$

(viii) $(S, b, B) \longrightarrow (S, \varepsilon)$

(ix) $(S, ac) \longrightarrow (S, \varepsilon)$

(x) $(S, b, c) \longrightarrow (S, \varepsilon)$.

Problem 4: Is the class of context-free languages closed under the Prefix operation?

Solution: Yes,

Let $pref(L)$ denote the set of prefixes of strings in L.

$$Pref(L) = \{u \mid \exists v \in A^*, uv \in L\}.$$

We first do away with the case of the empty language. $pref(\phi) = \phi$ is a CFL iff ϕ is a CFL. whether ϕ is a CFL or not (it is) is not even relevant.

Next, notice that $pref(L)$ contains the empty string for any non-empty L. Thus $pref(L) = pref(L \setminus \{\epsilon\})$. So we can assume that L does not contain the empty string.

Now Let $G = (N, A, S, P)$ be a CFG for L in CNF.

Assume that there are no non-terminals from which no terminal string can be derived.

For each non-terminal $X \in N$, introduce a new non-terminal $X_\perp \notin N$. Denote the set of these new non-terminals $N_\perp = \{X_\perp \mid X \in N\}$. we will call these "vanishing" non-terminals, or simply "vanishers". For each $X_\perp$, we add the following productions to P:

- $X_\perp \to \epsilon$
- $X_\perp \to a$, for each production $X \to a$.
- $X_\perp \to YZ_\perp$ and $X_\perp \to Y_\perp$, for each production $X \to YZ$.

Call this new set of productions P'

Intuitively, vanishing non-terminals are allowed to go the empty string directly. They represent truncation, and as such can only be allowed to appear at the end of any sentential form.

we claim that $G' = (N \cup N_\perp, A, S_\perp, P')$ is a CFG for the prefix of L, i.e.,

$$L(G') = pref(L)$$

We will use "non-terminal sentential form" to mean a sentential form which contains at least one non-terminal. we use $\Rightarrow$ to denote derivation steps in both G and G'. The context will make it clear which grammar we are referring to (if any vanisher is Present, we are refferring to G'.)

Lemma 4.1: Let $S_\perp \overset{n}{\Rightarrow} w \overset{\perp}{\Rightarrow} w$ be a leftmost derivation of $w \in L(G')$. Then $w = zX_\perp$ for some $z \in A^*$ and $X \in N$.

Moreover, each intermediate sentential form is of the form $\gamma X_\perp$ for some $\gamma \in (A \cup N)^*$ and $X \in N$.

Proof: The last symbol in $S_\perp$ is vanishing. From the definition of P' we see that for any production $X_1 \to \gamma$, γ is either a terminal string or ends with a vanisher. If a production from the rightmost non-terminal contains no non-terminal then, it must be the last step in the derivation (since it is leftmost). Combining these two, we can induct on the length 'n' of the derivation. For $n = 0$ the only sentential form in $S_\perp$, and the claim holds. In each step before the last, it is not possible for the last non-terminal to go to a string But then each step must maintain that the last symbol is a vanisher.

By Induction each form (including w) is form $\gamma X_\perp$ for some $\gamma \in (A \cup N)^*$ and $X \in N$.

But w contains exactly one non-terminal, since it derives a terminal string in a single step. Thus $w = zX_\perp$ for some $z \in A^*$ and $X \in N$.

Lemma 4.2: Suppose zX_1 is a sentential form in G'. Then there is a sentential form $zX\gamma$ in G.

Proof: Consider a leftmost derivation $S_1 \overset{*}{\Rightarrow} zX_1$. By the previous lemma, each intermediate sentential form is of the form γY_1. For any Production $Z \to \zeta$, keep it as is. For any production $Z_1 \to \zeta$, replace it with the corresponding production in P. That is,

$$Z_1 \to a \quad \Rightarrow Z \to a$$

$$Z_1 \to Z_1(Z_2)_1 \quad \Rightarrow Z \to Z_1 Z_2$$

$$Z_1 \to (Z_1)_1 \quad \Rightarrow Z \to Z_1 Z_2$$

where in the last case, $Z \to Z_1 Z_2$ must exist in P for some Z_2, in order for $Z_1 \to (Z_1)_1$ to be in P'.

Then for any $\gamma Y_1 \overset{1}{\Rightarrow} \gamma'$, this process preserves the first $|\gamma'|$ symbols, but with each vanishing non-terminal replaced by its non-vanishing counterpart. Thus, this yields a derivation of some sentential form γ'', whose $|zX_1|$-length prefix is zX.

Proposition 4.3. $L(G') \subseteq \text{pref}(L(G))$.

Proof: Let $S_1 \overset{*}{\Rightarrow} w \overset{1}{\Rightarrow} w$ be a leftmost derivation of $w \in L(G')$.
(By Lemma 4.1) $w = zX_1$ for some $z \in A^*$ and $X \in N$. By Lemma 4.2, there is a sentential form $zX\gamma$ of G.
We assumed that each non-terminal derives some terminal string. By extension, G derives a terminal string. Let γ derives a terminal string y, and X derive a terminal string x.
The last step in the derivation of w must involve X_1.
If it is $X_1 \to a$, then $X \to a$ is a production in P and so $zay \in L(G)$. Then $w = za \in \text{pref}(L(G))$. If it is $X_1 \to \epsilon$, then $w = x$. But $zxy \in L(G)$. So $w \in \text{pref}(L(G))$. Thus, $L(G') \subseteq \text{pref}(L(G))$.

Proposition 4.4. $\text{pref}(L(G)) \subseteq L(G')$.

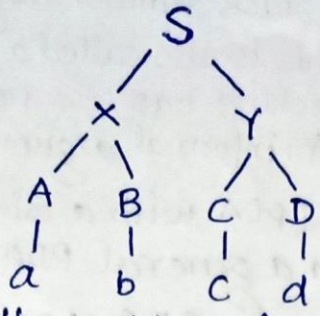
Proof: Let $w \in \text{pref}(L(G))$
Look at the parsh tree of a derivation of w in G. Since $w \in \text{pref}(L(G))$, the leftmost $|w|$ leaves of the parsh tree given w.
We will give a scheme to derive w in G'.

- If $w = \epsilon$ then apply production $S_1 \to \epsilon$. otherwise subtree rooted at root which contain only leaves of w.
- If the root production $S \to a$, then $w = \epsilon$ or $w = a$, which can both be derived from S_1 in a single step.
- Otherwise root production $S \to XY$, and the w-subtree must contain at least one child of the root.
  - → If the subtree for w only contains left child, then apply the production $S_1 \to X_1$. This reduces the problem to deriving w from X_1, which can be done by repeating this process on the subtree rooted at X.
  - → If it contains both, then $w = xy$, where x is derived from X, and y is the prefix of a terminal string derived from Y. Keep the productions from X as is. Apply this same process to the subtree rooted at Y to derive y.

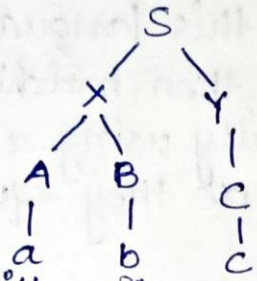
For Example, Consider the grammar G in CNF with productions

$$S \to YY$$
$$X \to AB$$
$$Y \to CD$$
$$A \to a \quad B \to b \quad C \to c \quad D \to d$$

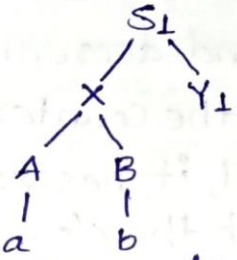
The parse tree for abcd is



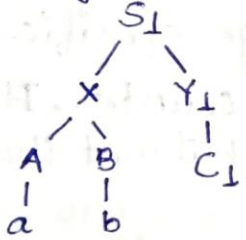
- For the prefix abc, the subtree is;



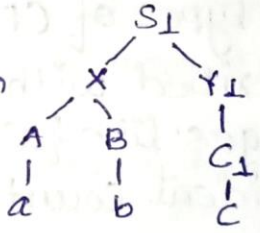
- Then the above algorithm gives the first step as $S_1 \rightarrow XY_1$ and asks to keep the tree rooted at x fixed.



Then applying this process to derived c from Y_1 gives.



and then



Which is a derivation of abc in G'.

Combining the two propositions, we have

$$L(G') = \text{pref}(L(G)).$$

Problem 5 Consider a PDA with a single stack symbol say "1" (apart from the bottom-of-stack symbol "1"). This is also called a one-counter machine. Argue that a one-counter machine has less power (in terms of language recognition) than a full PDA. An informal argument is sufficient.

Solution: A one Counter machine (PDA with a single stack symbol) is strictly less powerful than a general PDA.

To see, we consider the classic CFL $L = \{a^n b^n c^n \mid n \geq 0\}$.

A General PDA can recognize this language by using its stack to count the no. of a's, then matching them with b's by popping the stack, and finally using a second stack symbol to count c's and ensure they equal the no. of a's and b's.

However, a one-counter machine cannot recognize this language. While it can use its counter to match a's with b's (by incrementing for each a and decrementing for each b), once it finishes processing b's, the counter would be zero. With only a single stack symbol, it has no way to remember the original count to match with the c's.

This limitation means a one-counter machine can only recognize certain types of CF. language, specifically those that can be described with a single counter. It can recognize languages like $\{a^n b^n \mid n \geq 0\}$ but not those requiring multiple independent counts.

Note that, language accepted by one-counter machines from a proper subset of context-free language, sometimes called one-counter languages. They sit between regular languages and general context-free language in the chomsky hierarchy.

Problem 6- show that the following inter division function div is computable by a Turing Machine in the sense discussed in class. Give a complete description of the moves of the TM.

The function div is defined as:

$$div : \mathbb{N} \times \mathbb{N} \to \mathbb{N}, \text{ where } div(m,n) = \begin{cases} \lfloor m/n \rfloor, & \text{if } n > 0 \\ 0, & \text{otherwise} \end{cases}$$

Solution: **First Tape:**

⊢	1	1	1	1	b	b
↑						

Second Tape:

⊢	1	1	b	b	b	b	b
	↑						

Third Tape:

⊢	b	b	b	b	b	b	b

→ we will use three tape turing machine to simulate the division fn. The first tape will contain the input m, the second tape will contain the input n and the third tape will contain the output of the division function. The Turing machine will work as follows:

- Intially the all the three pointer will be pointing the ⊢ symbol on the tape, we will use the principle for the division that is we will subtract the n from m until m is less than n and we will count the no. of times we subtracted the n from m and that will be the output of the division function. The tape will have string in the form of ⊢ 1* b* here n and m are represented in the unary form.

So, here we are considering m > n then mark the first symbol of first and second tape as 1 → X and now traverse the m,n consequently till we complete all the 1's in the second tape and then we will keep the pointer in the first tape where it was and we will move the pointer of the second tape of the left on the intiall symbol but while traversing to the left in the second tape we will change X → 1 and now the third Pointer the bottommost tape we will change b → $\hat{1}$ and shift the pointer to the right. Now we will transverse again in the same Manner we will move the pointer of the first tape where we left off and pointer of the second tape simultaneously to the right while changing 1 → 1 when we again complete the traversal we will change the X → 1 and $\hat{1}$ → b and will move the pointer of the third tape to the right and we will repeat this process until we reach at a point where we can't traverse the first tape and the second tape simultaneously. Since the no. of 1's in the first tape left are lesser than the 1's in the second tape and we will return the no. of $\hat{1}$'s in the third tape as the output of the division function.

The case where n ≤ 0 will return 0 as the output of the division function.

P.T.O

The case where $m < n$ we will return 0 as the output of the division function. Since we can't simultaneously traverse the first tape and the second tape.

Formal Description:

1. Initialize the three tapes with the input m,n and the output of the division function.
   - Tape 1: $\vdash 1^m b^*$
   - Tape 2: $\vdash 1^n b^*$
   - Tape 3: $\vdash b^*$
2. Step: Move the head of Tape 1 and Tape 2 rightward and do $1 \to X$ to the first 1 we encounter.
3. Do the step till one of the following happens.
   - Tape 2 loses all its 1s but tape 1 has 1s. In this we say the unary representation of $\lfloor m/n \rfloor$ is in tape3. (if there are no $\hat{1}$s in tape3, that implies that $\lfloor m/n \rfloor = 0$).
   - Tape 2 loses all its 1s, in this case we move the head of Tape 3 to the right and turn that value to $\hat{1}$. After this we move leftward, turning all the $X \to 1$. The head is now at the start symbol.
4. Termination:
   Stop the process when we have no 1s left in Tape 1 or no. of 1's left unmarked in first tape is lesser than the no. of 1's of n. Then Tape 3 contains the Unary representation of $\lfloor m/n \rfloor$.

This Turing machine correctly compute the div(m,n) function as described.

▽△