Pumping lemma theorem for regular language

=> Pumping lemma theorem is used to find out whether the language is regular or not.

=> It is use to prove that a language is not Content free.

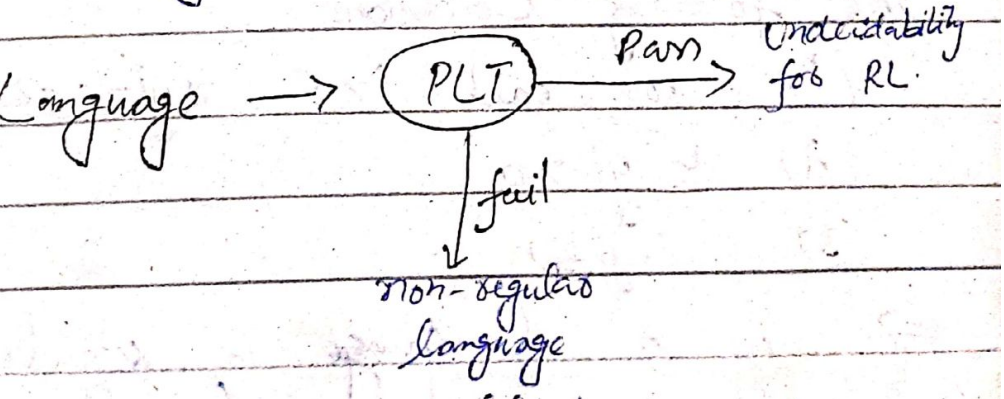
=> There are two types of languages.

- Finite and infinite.

=> Finite language is always regular language.

=> Infinite language may be a regular or may be non-regular.

=> Pumping lemma is a negative test.



=> That's why it is called negative test.

⇒ It is called "pumping" because pump more stuff into the middle of the word, swelling it up without changing the front and back part of the string

Statements

⇒ if L is an infinite language then there exists some positive integer 'n' (pumping length) such that any string $w \in L$ has length greater than equal to 'n'

⇒ $|w| \geq n$ then w can be divided into three parts, $w = xyz$ satisfy the following conditions:

(1) for each $i \geq 0$, $xy^iz \in L$

(2) $|y| > 0$

(3) $|xy| \leq n$

⇒ if a language is regular then every sufficient long string in the language has a non-empty substring that can be "pumped", that is repeated any number of times while the resulting strings are also in the language.

It is based on pigeonhole principle.

Q. Prove that language is non-regular for L is $a^n b^n$ over $\Sigma = \{a, b\}$

⇒ Suppose that $L = \{a^n b^n\}$ is regular language. $L = \{ab, aabb, aaabbb, \ldots\}$

⇒ ∃ exist a length P for L.

⇒ Let $w = a^P b^P$

$P = 6$

$w = aaabbb$

⇒ Look at all decompositions

$w = \underset{x}{aa}\ \underset{y}{ab}\ \underset{z}{bb}$

- $|xy| \leq n$
  $4 \leq 6$
- $|y| > 0$
  $2 > 0$
- increase the power y
  $i = 2$
  $w = xy^2z = aaaabbbb \notin L$
- So, the given language is non-regular

Q. Prove that language is reg... ulas for all the strings end... with "bab" defined over Σ = {a,b}

⇒ Suppose that language is regular

⇒ ∃ exist a pumping length p.

|P| = 8

w = abababab , |w| ≥ P

⇒ Look at all decompositions.

w = ab ab abab
    x y z

- |xy| ≤ 8
  |5| ≤ 8
- |y| > 0
  3 > 0
- Now, pump y
  i = 2
  xy^2z
  abababababab ∈ L

So, the given language is regular language.

Q. Prove that $\{0^n 1^n | n \geq 1\}$ is non-regular language.

⇒ Suppose that language is regular.

⇒ ∃ exist a pumping length P for L

⇒ Choose w ∈ L and |w| > P

$w = 0^P 1^P$

Look at all decomposition of w.

$x = 0^\alpha$

$y = 0^\beta$

$z = 0^{P-\alpha-\beta} 1^P$

Suppose i = 2:

$xy^2z = x y^2 z$

$= 0^\alpha 0^\beta 0^\beta 0^{P-\alpha-\beta} 1^P$

$= 0^\alpha 0^P 0^\beta 0^P 0^{-\alpha} 0^{-\beta} 1^P$

$= 0^{P+\beta} 1^P \notin L$

⇒ L is not regular language.

Q. Prove that language L = $\{0^i 1^j : i > j\}$ is non-regular.

⇒ Suppose that L is regular.

⇒ ∃ a pumping length P for language.

Choose $w = 0^{P+1} 1^P$ (∈ L and |w| ≥ P)

Look at all decompositions.

$x = 0^{\alpha}$ $\alpha \geq 0$

$y = 0^{\beta}$ $\beta \geq 1$

$z = 0^{p+1-\alpha-\beta} 1^{p}$

Let

$i = 0$;

$xz = 0^{\alpha} 0^{p+1-\alpha-\beta} 1^{p}$

$= 0^{p+1-\beta} 1^{p}$

$\in L$ iff $p+1-\beta > p$

$\Leftrightarrow 1-\beta > 0$

$\Leftrightarrow 1 > \beta$ Contradiction

$\Rightarrow$ So L is not regular language

Q. Show that perfect square language is not a regular language.

$L = \{ 1^{n^2} ; n \geq 0 \}$

Step 1: Assume that L is a regular language

$L = \{ \varepsilon, 1, 11, 1^4, 1^9, 1^{16}, \ldots \}$

Step 2:

Assume that there exist a pumping length n.

Step 3:

Take a string $w \in L$,

$|w| \geq n$

Let $|w| = n^2 \geq n$

Step 4:

Look at all decomposition.

$|xy^i z|$ with the following condition.

$|y| > 0$

$|xy| \leq n$

min length of $|y| = 1$

max length of $|y| = n$ if $x = \varepsilon$

Step 5:

$|xy^2 z| = |xyz| + |y| = n^2 + 1 > n^2$

$|xy^2 z| = |xyz| + |y| = n^2 + n < (n+1)^2$

$\Rightarrow n^2 < |xy^2z| < (n+1)^2$

So, there is not perfect square $|xy^2z| \notin L$, strictly lie b/w n^2 and $(n+1)^2$ but is not square

So, the given language is non-regular language.

Q. Show that perfect cube language is not a regular language.

$L = \{ 0^{n^3}, n \geq 0 \}$

~~$L = \{ \varepsilon, 0, 0000000, 0^{64} \}$~~

$L = \{ \varepsilon, 0, 0^8, 0^{27}, \ldots \}$

$\Rightarrow$ Assume that language is a regular language.

$\exists$ a pumping length n.

$\Rightarrow$ Take $w \in L$,

$|w| \geq n$

Let $|w| = n^3 \geq n$

$|xy^iz|$

min length $|y| = 1$

max length $|y| = n$

$\Rightarrow |xy^2z| = |xyz| + |y| = n^3 + 1 > n^3$

$\Rightarrow |xy^2z| = |xyz| + |y| = n^3 + n < (n+1)^3$

$\Rightarrow$ So, $|xy^iz| \notin L$, is a non-regular language.

Q. Show that a^{2^i} is a non-regular language.

$L = \{ 0^{2^n}, n \geq 1 \}$

Step 1: Assume language is a regular $\exists$ a pumping length m.

Step 2:

$\Rightarrow$ Take $w \in L$, $|w| \geq m$

$|w| = 2^n \geq m$ $\quad m \leq |y| \leq 1$

Step 3:

$|xy^2z| = |xyz| + |y| = 2^m + 1 > 2^m$

$|xy^2z| = |xyz| + |y| = 2^m + m < 2^{m+1}$

$\Rightarrow 2^m < |xy^2z| < 2^{m+1}$

$\Rightarrow |xy^2z| \notin L$

So it is not a regular language

Q. Show that $L\{a^n b^{2n} \mid n \geq 1\}$ is not a regular language.

$L = \{ab^2, a^2b^4, a^3b^6, \ldots\}$

Step 1: Suppose that given language is regular language.

$\exists$ exist a pumping length P.

Step 2:

Take $w \in L$,

$|w| \geq P$

can be represented as

$|xyiz|$ with $|y| > 0$ or $|y| \neq 0$ and $|xy| \leq P$.

$|a^P b^{2P}| = 3P$

$|w| = 3P > P$

Step 3:

$|xy^2z| = |xyz| + |y| = 3P + 1 > P$

$|xyz| = |xyz| + |y| = 3P + P = 4P < 4P+1$

$\Rightarrow 3P < |x^P y^2 z| < 4P+1$

So, $|xy^2z| \notin L$, $a^n b^{2n}$ is not a regular language.

Q. Show that $\{0^n 1^m / n \leq m\}$ is not a regular language.

→ Method 2:

⇒ Suppose that $L = \{a^n b^{2n}\}$ is a regular language and for regular language there is pumping length P.

⇒ Consider the string $w = a^P b^{2P} \in L$, where $|w| = 3P > P$.

⇒ According to the pumping lemma, we can divide into three parts:

$S = xyz$,

$|xy| \leq P$

∴ $|S| = 3P$ and $|xy| \leq P$, it follows that y must consist of only 'a' characters.

Let $y = a^k$, where $0 < k \leq P$

$|y| > 0$:

⇒ for all $k \geq 0$; $xy^k z$

$k = 2$

$xy^2z = xyyz$ ⇒ Since y consists of only of 'a' characters, adding y to xz will increase the number of a's in S but will have more more a's than b's. So, violates.

Q. Show that $\{a^n \mid n$ is a prime$\}$ is not a regular language.

=> Suppose that the given language is regular and L is accepted by DFA. Then it have n states.

=> ∃ exist a pumping length n

=> Take a string $w = a^p$ where p is a prime number and $p \geq n$.

$|w| \geq n$.

can be represented as:

$|xy^iz|$ with $|y| \geq 0$

And $|xy| \leq n$

=> $w = a^q = xyz$

$|xyz| = q$

$y = a^m$, $|y| = m$

$xy^iz \in L$

$|xy^iz| = |xyz| + |y^{i-1}|$

$= q + (i-1)m$

$= q + (q+1-1)m \quad \because i = q+1$

$i: q+1 \quad = q + qm$

$= q(1+m) \quad (m \geq 1)$

$|xy^iz|$ Not a prime number because it has two factors

Therefore, L is not regular.

PDA

- PDA stands for push down automata.
- PDA is a way to implement a CFG into diagramatic form.
- It implementation is same like as DFA, but DFA has finite memory and PDA has infinite memory.
- It uses Stack datastructure.
- PDA = FSM + Stack

formal def:

- A push down Automata is formally defined by 7 Tuples as shown below.

$$P = (Q, \Sigma, \Gamma, \delta, q_0, z_0, F)$$

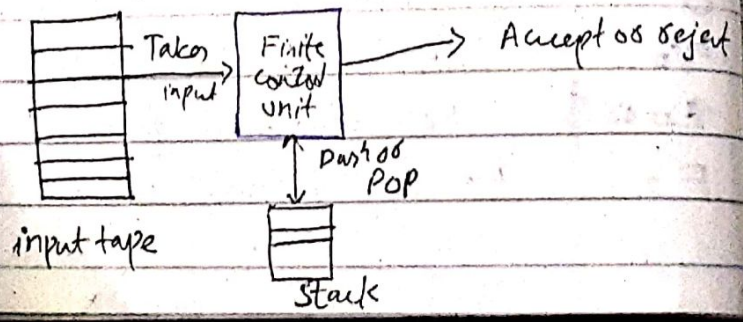
where,

- Q = A finite set of states
- Σ = A finite set of input symbols
- Γ = A finite stack alphabet.
- δ = The transition function
- q_0 = The start state
- z_0 or \$ = The start stack symbol
- F = The set of final / Accepting states

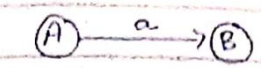
- δ takes as argument a triple (q, a, X) where
  - **q:** is state symbol
  - **a:** a is either an input symbol in Σ or a = ∈
  - **X:** X is a stack symbol, that is a member of Γ.
- The output of δ is finite set of pair (P, γ) where
  - P is a new state
  - γ is a string of stack symbols that replaces X at the top of the stack

PDA block Diagram:

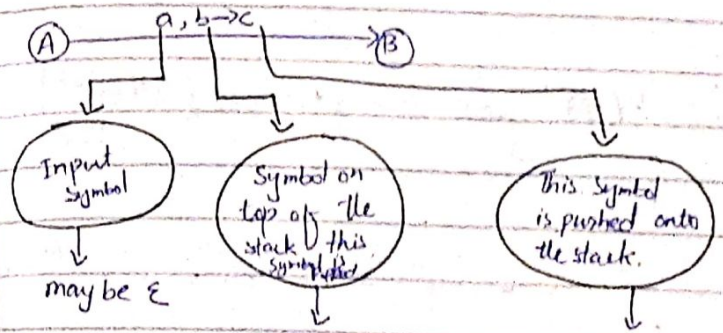
⇒ A pushdown Automata has 3-components
- An input tape.
- A finite Control unit
- A stack with infinite size



FSM



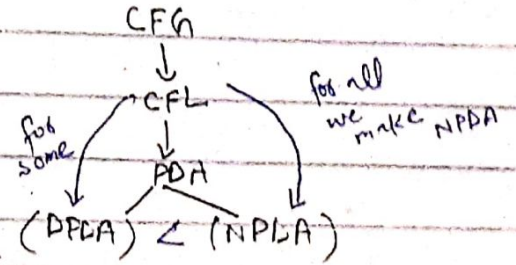
Pushdown Automata:



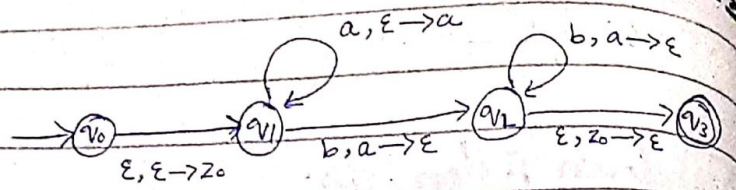
Note:

⇒ Z_0 is by default into the stack that is popped at last in FILO manner

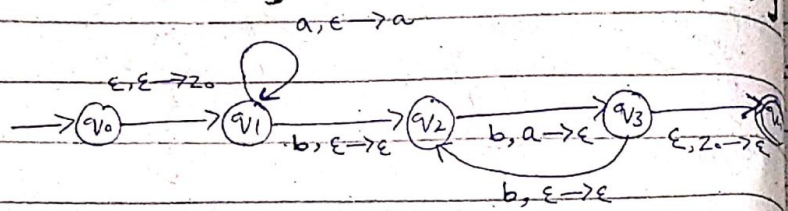
Note:



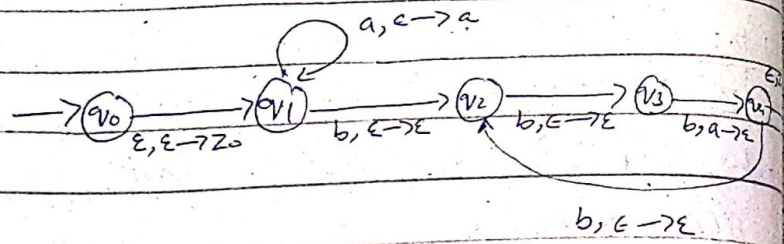
Q# PDA for language $\{a^n b^n : n \geq 1\}$



Q# PDA for language $\{a^n b^{2n} | n \geq 1\}$

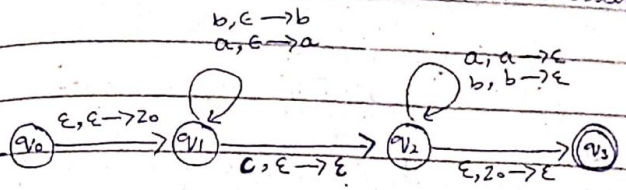


Q # PDA for language $\{a^n b^{2n+1} | n \geq 1\}$



Q# PDA for odd palindrom language. $L = WcW^R$

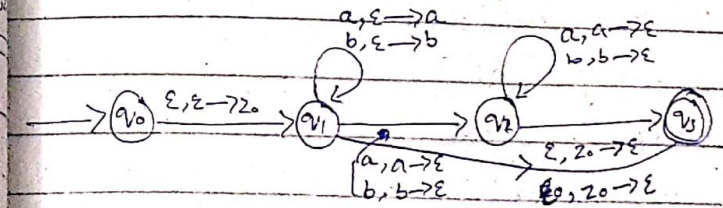
⇒ c is the middle element that can be either a or b.



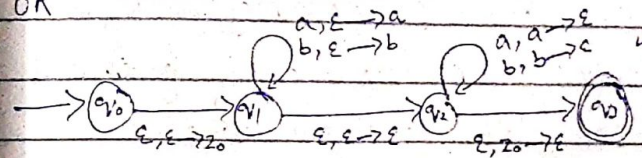
c = a or b

⇒ Note: There is no method to find the half of the string we find it non-deterministicly.

Q#: PDA for even palindrom language. WW^R = L = { ε, aa, bb, ...

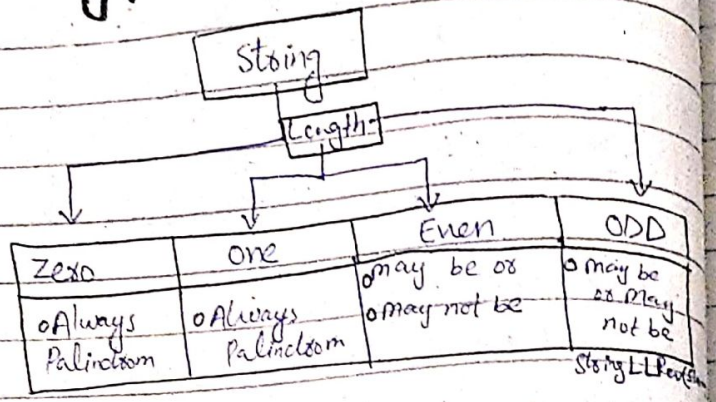


OR



CamScanner

Q. How to make Palindrom String?



Note:

=> We can generate a new palindrom even string from even length palindrom string.

=> We can also generate a new even length palindrom even if it is not palindrom:

W = String = ab

W^R = Rev (String) = ba

WW^R = abba

=> An odd string may or may not palindrom but we can generate a new odd length palindrom string.

Example:-

String = 110

Rev(String) = 011

String LL Rev(String)

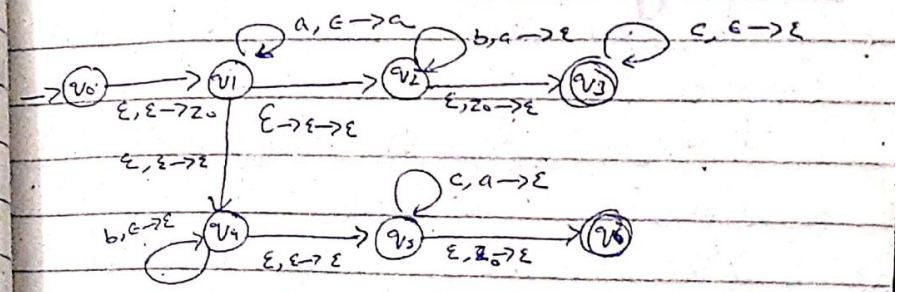
= 110 11

PDA for language:

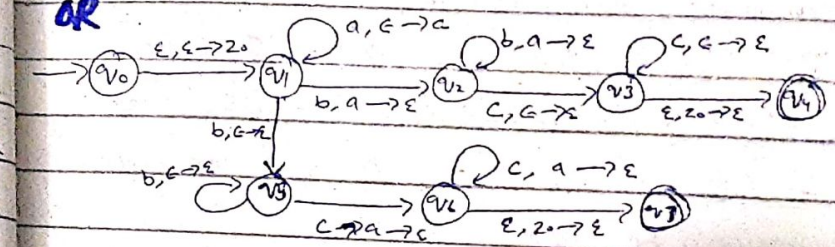
$\{a^i b^j c^k \mid i, j, k \geq 0$ and $i = j$ or $i = k\}$

Possibilities:

$a^i b^i c^k$ or $a^i b^j c^i$



OR



Q# What is the concept of pumping lemma I and II and What is the difference between them?

⇒ The pumping lemma is a fundamental concept in TOAT. to prove that a certain language is regular or not.

⇒ There are two versions:

- Pumping lemma I
- Pumping lemma II

Pumping lemma I:-

⇒ Pumping lemma I is used for regular language.

⇒ It shows a given language is not a regular language.

⇒ It states that, if A is a regular language, then A has a pumping length 'P' such that any string 'S' where $|S| \geq P$ may be divided into three parts $S = xyz$ such that the following conditions must be true.

(1) $xy^iz \in A$ for every $i \geq 0$

(2) $|y| > 0$ or $|y| \neq \varepsilon$

(3) $|xy| \leq P$

- it is proved by using Contradiction.
- S cannot be pumped == Contradiction.

⇒ In Simple, pumping lemma I states that for any regular language, there exists a certain length at which longer strings in the language can be "pumped" by repeating a portion of the string (y) any number of times and still remain the language.

Pumping lemma II:

- pumping lemma II is used to prove that a given language is not a content free language.

⇒ Statement:

If A is a content free language then A has a pumping length 'P' such that any string 'S' where $|S| \geq P$

may be divided into three parts $S = uvxyz$ such that the following conditions must be true:

- uv^ixy^iz is in A for every $i \geq 0$
- $|vy| > 0$ or $|vy| \neq \varepsilon$
- $|vxy| \leq P$

Difference b/w:

=> The only difference is that conditions in pumping lemma II are more strict than pumping lemma I that are difficult to prove non-regular by pumping lemma I are proved non-regular by pumping lemma II.

=> In pumping lemma I we have to generate all words to of a language but in pumping lemma II we have to generate a single word to prove a language non-regular.

Explanation:

Some languages like PALINDROM that are proved to be regular by first version due to some of their symmetrical words when we pump these words they remain to be the parts of the language like:-

bbabb

=> By pumping lemma I:-

$y = a$

=> Now, repeating y three times results in

bbaaabb

=> That is also a valid word of PALINDROM so by pumping lemma I PALINDROM can not be proved non regular, so these was the need of pumping lemma version 2.

=> Now Consider for the word

bbabb

if we take $N = 2$

=> Then by pumping y two times results in

bbbbabb

=> But if we take $N = 3$ and $y = a$

Then by pumping y two times results in

bbaaabb

That word is in palindrom. So be

Careful in taking total no of states of the FA and also the repeating factor (y).

⇒ Both pumping Lemma I and pumping Lemma II provide necessary conditions for regular and Content-free languages, respectively, but they do not provide sufficient conditions.

⇒ That is, if a language does not satisfy the conditions of the pumping Lemma, it is not necessarily non-regular or non-content free.

Closure Properties for CFL:

⇒ The closure properties of CFL are:

- Closed under '+'
- Closed under '$L_1 \cdot L_2$' (Concatenation)
- Closed under Kleene closure
- Not closed under intersection
- Not closed under complementation

Decision properties of CFL:-

- Decision properties means whether a problem is decidable or not decidable under CFL.
- Decidable means we can develop algorithm that answers the following questions.
  (1) Is the given string in the language (membership).
  (2) Is the language empty? (Emptiness)
  (3) Is the language finite? (Finiteness)
  (4) Equality of DCFL and CFGs

- CFGs is undecidable for ambiguity, equality, regularity of CFGs.

Applications of CFGs

- CFGs were originally conceived by Noam Chomsky as a way to describe natural languages.
- CFGs are used in Compilers Science because of its capability of defining recursive rules.
- This recursive way of defining rules and definitions has two major applications:

Parsers:

- A component of compilers that discovers the structure of the source program and represents that structure by a parse tree.

DTD's in XML:

- Describes the allowable tags and the way in which tags may be nested.

Q. Prove that CFL may or may not closed under the operation of complement.

⇒ Suppose CFL is closed under complement

To prove complement:

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

L.H.S

The languages L_1 and L_2 are CF

$L_1 = \{a^n b^n c^m \mid n \geq 0 \wedge m \geq 0\}$ and $L_2 = \{a^n b^m c^n \mid n \geq 0 \wedge m \geq 0\}$

$L_1 \cap L_2 = \{a^n b^n c^n \mid n \geq 0\}$ not a regular language.

R.H.S

$= \overline{\overline{L_1} \cup \overline{L_2}}$ ∴ we assume CFL is closed under complement

$= \overline{\overline{CFL} \cup \overline{CFL}}$

$= \overline{\overline{CFL}} = \overline{\overline{CFL}}$

L.H.S ≠ R.H.S

So, our assumption is wrong; hence prove that CFL is not closed under complement.

Because if they are closed under complement, then they are closed under intersection. which is false.

Q. Prove that intersection of CFL is not always CFL.

Let L_1 and L_2 are Content free.

$L_1 = \{ a^n b^n c^m \mid n \geq 0 \wedge m \geq 0 \}$.

$L_2 = \{ a^n b^m c^m \mid n \geq 0 \wedge m \geq 0 \}$

$L_1 \cap L_2 = \{ a^n b^n c^n \mid n \geq 0 \}$ is not content free

So, it is not closed under intersection.

Q. prove that CFL closed under union.

⇒ Let L_1 and L_2 are Content free.

$L_1 = \{ a^n b^n \mid n \geq 0 \} \Rightarrow S_1 \rightarrow aS_1b \mid \varepsilon$

$L_2 = \{ c^n d^n \mid n \geq 0 \} \Rightarrow S_2 \rightarrow cS_2d \mid \varepsilon$

$L_1 \cup L_2 \Rightarrow$ $S \rightarrow S_1 \mid S_2$

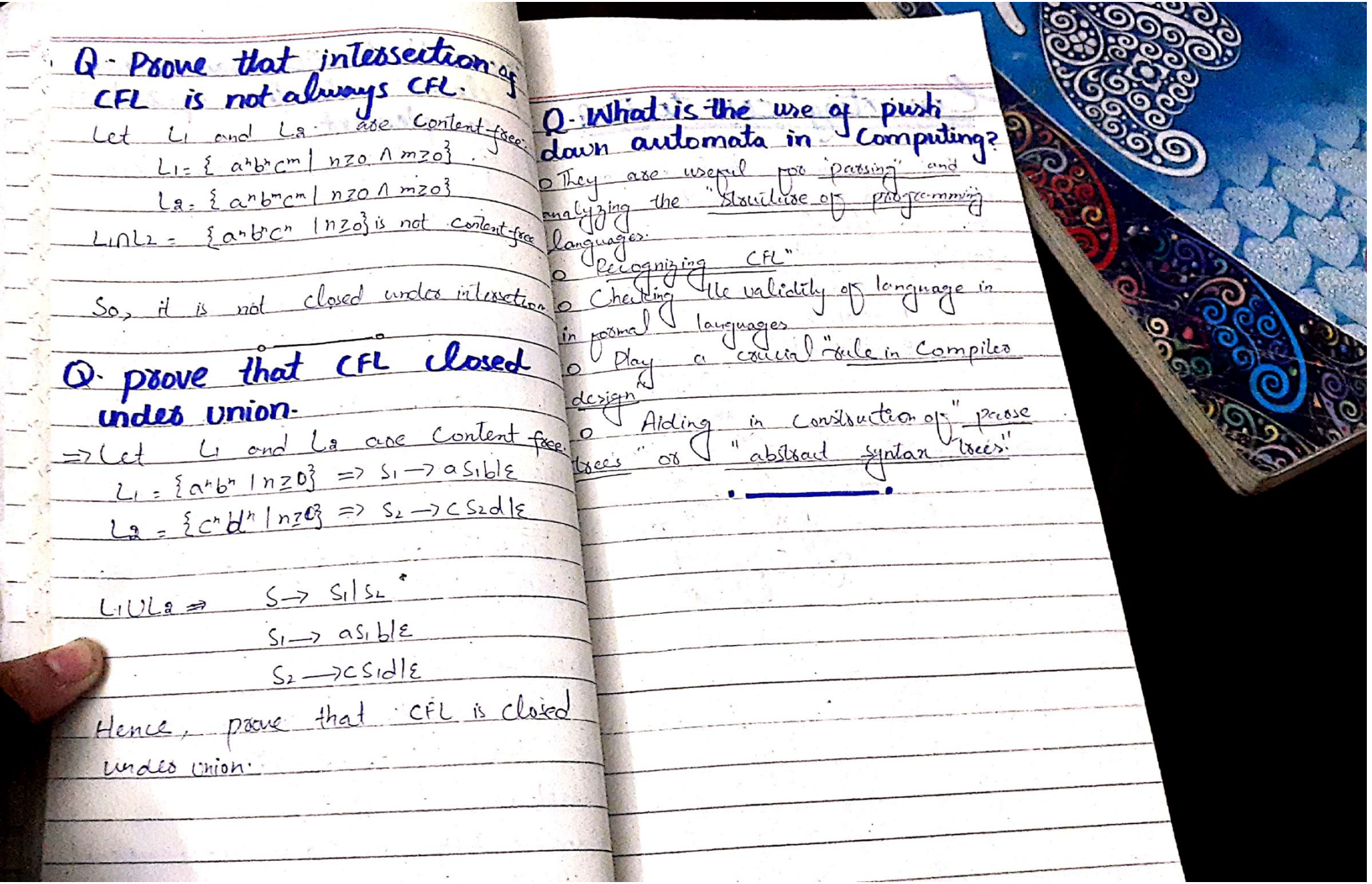
$S_1 \rightarrow aS_1b \mid \varepsilon$

$S_2 \rightarrow cS_1d \mid \varepsilon$

Hence, prove that CFL is closed under union.

Q. What is the use of push down automata in Computing?

- They are useful for "parsing" and analyzing the "structure of programming languages."
- "Recognizing CFL"
- Checking the validity of language in formal languages
- Play a crucial "role in Compiler design"
- Aiding in construction of "parse trees" or "abstract syntax trees."



Converting CNF to PDA

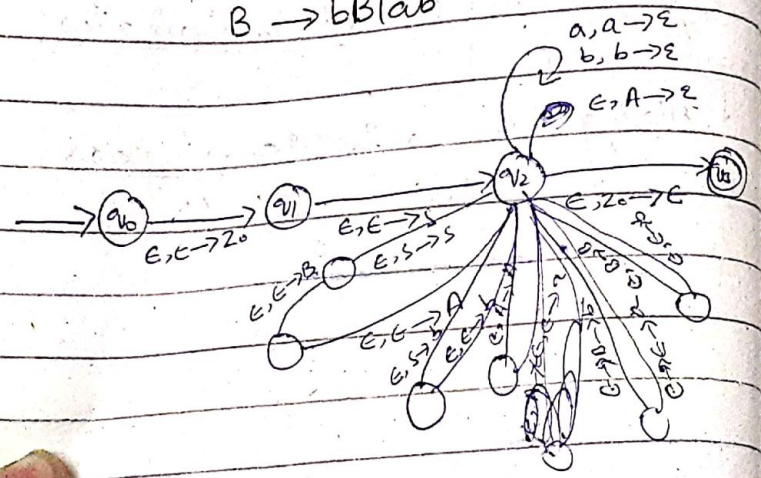
⇒ Each non-terminal should be popped before pushing the production of that non-terminal.

Example: Pop ← Push →

S → ABS | ab

A → aA | ε

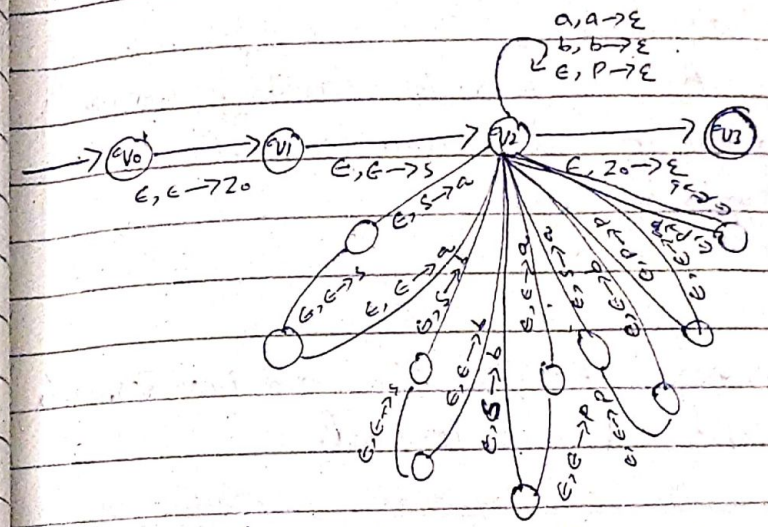
B → bBlab



Example 2:-

S → aSa | bSb | aPb | bPa

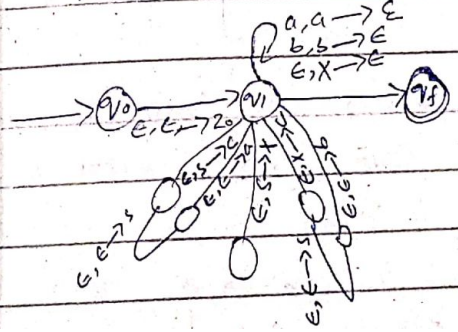
P → aP | bP | ε



Convert the CFG of following Lang into PDA. L = $a^i b^j c^k$ and $j+i=k$

S → aSc | X

X → bSc | ε



GNF :-

- GNF stands for Greibach Normal form.
- GNF is a method to normal the CFG.
- A CFG is in Greibach Normal form if the productions are in the following forms:

$A \rightarrow b$

$A \rightarrow bC_1C_2 \cdots C_n$

Where $A, C_1, \ldots, C_n$ are Non-Terminals and b is a terminals

Steps to Convert a given CFG to GNF:-

(1) Check if the given CFG has any unit productions or Null productions and remove if there are any.

(2) Check whether the CFG is already in CNF and Convert it to CNF if it is not.

(3) Change the names of the Non-terminal symbols into some A_i in ascending order of i.

Example:

$S \rightarrow CA \mid BB$

$B \rightarrow b \mid SB$

$C \rightarrow b$

$A \rightarrow a$

Replace:

S with A_1

C with A_2

A with A_3

B with A_4

we get:

$A_1 \rightarrow A_2A_3 \mid A_4A_4$

$A_4 \rightarrow b \mid A_1A_4$

$A_2 \rightarrow b$

$A_3 \rightarrow a$

Step 4: Alter the rules so that the non-terminals are in ascending order such that, if the production is of the form $A_i \rightarrow A_jx$, then, $i<j$ and should never be $i>j$.

Continue.....

CNF VS GNF

CNF

- Stands for Chomsky Normal form.
- $A \to BC$ {A, B, C ∈ Variables
  OR $A \to a$ {a ∈ Terminal
- No of steps required to generate a string of length 'n' is $2|n|-1$
- Used in membership algorithm.
- Derivative or parse tree obtained from CNF is always binary tree.
- Length of each production is restricted.

GNF

- stands for Greibach Normal form.
- $A \to ax$
  where { $x \in V^*$
        { $a \in T$
- No of steps required to generate a string of length 'n' is 'n'
- used to Convert CFG to PDA
- Not always
- Not

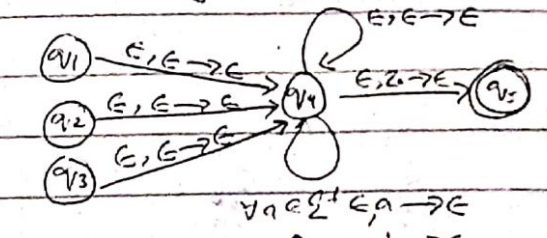
PDA to CFG

To Convert PDA to CFG the following steps should be followed:

- Simplify the PDA
- Build CFG from PDA

How to simplify PDA?

1) There should be Single accepting state.

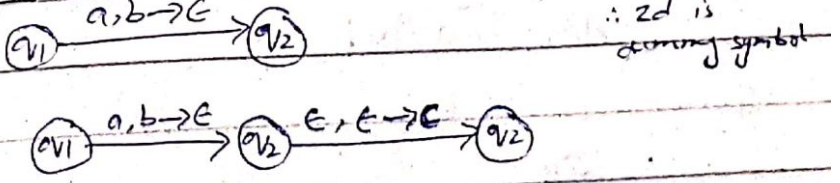


2) Empties the stack before accepting.

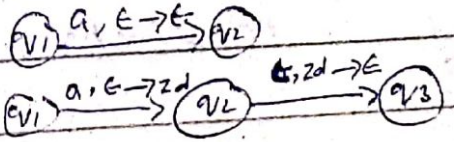
3) No symbol should be push and pop at the same time.

e.

Case I:



Case II



Build CFG

- $A_{xy} \rightarrow \epsilon$ Always include.

Type-I: ⇒ To get from any state

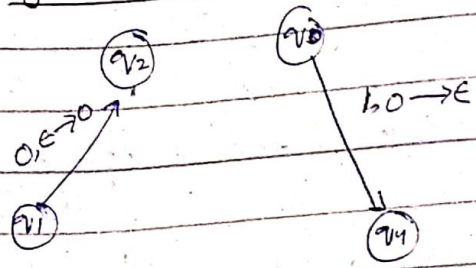
$A_0 q_0 \rightarrow \epsilon$

$A_1 q_1 \rightarrow \epsilon$

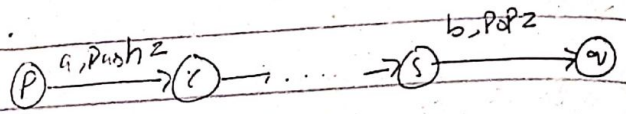
⋮

$A_n q_n \rightarrow \epsilon$

Type - II:

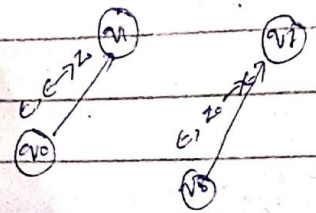


$A_{q_1 q_4} \rightarrow 0 A_{q_2 q_3} 1$



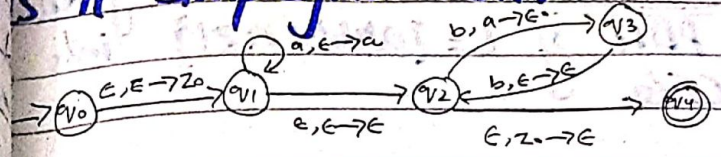
$A_{pq} \rightarrow a A_{rs} b$

Type 3



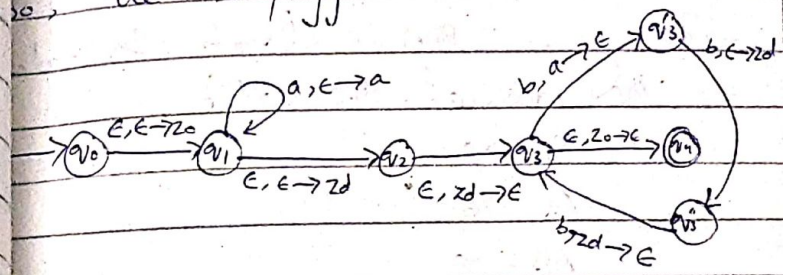
$A_{q_0 q_8} \rightarrow A_{q_1 q_7}$

Is it Simplified PDA?



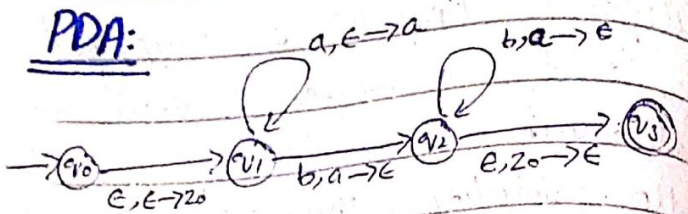
No, the given PDA is not simplified because at $q_2 \rightarrow q_3$, $\epsilon \rightarrow \epsilon$ are not allowed.

So, the Simplify PDA is:



Example: Convert the given PDA of $L=\{a^n b^n : n \geq 1\}$ into CFG.

PDA:



=> The given PDA is already Simplified.

=> The produced CFL will be:

$A_{11} \rightarrow \in$

$A_{22} \rightarrow \in$

$A_{33} \rightarrow \in$

$A_{44} \rightarrow \in$

$A_{11} \rightarrow A_{11}A_{11} \mid A_{12}A_{21} \mid A_{13}A_{31} \mid A_{14}A_{41}$

$A_{12} \rightarrow A_{11}A_{12} \mid A_{12}A_{22} \mid A_{13}A_{32} \mid A_{14}A_{42}$

$A_{13} \rightarrow A_{11}A_{13} \mid A_{12}A_{23} \mid A_{13}A_{33} \mid A_{14}A_{43}$

.....

$A_{42} \rightarrow A_{41}A_{12} \mid A_{42}A_{22} \mid A_{43}A_{32} \mid A_{44}A_{42}$

$A_{43} \rightarrow A_{41}A_{13} \mid A_{42}A_{23} \mid A_{43}A_{33} \mid A_{44}A_{43}$

$A_{44} \rightarrow A_{41}A_{14} \mid A_{42}A_{24} \mid A_{43}A_{34} \mid A_{44}A_{44}$

$A_{23} \rightarrow 0A_{22} \mid bA_{33}$

$A_{14} \rightarrow \in A_{22} \in$

$S \rightarrow A_{14}$

Topic remaining of CFGs to GNF

$A_4 \rightarrow bA_1A_4$

$A_4 \rightarrow b \mid bA_2A_3A_4 \mid A_4A_4A_4$

$A_4 \rightarrow b \mid bA_3A_4 \mid A_4A_4A_4$

↓ Left Recursion

Steps: Remove left Recursion

- Introduce a New variable to remove the left recursion.

$A \rightarrow b \mid bA_3A_4 \mid A_4A_4A_4$

$Z \rightarrow A_4A_4Z \mid A_4A_4$

Now,

$A \rightarrow b \mid bA_3A_4 \mid bZ \mid bZ \mid bA_3A_4Z$

Now the grammer is:-

$A_1 \rightarrow A_2A_3 \mid A_4A_4$

$A_4 \rightarrow bA_3A_4 \mid b \mid bZ \mid bA_3A_4Z$

$Z \rightarrow A_4A_4 \mid A_4A_4Z$

$A_2 \rightarrow b$

$A_3 \rightarrow a$

Now, $A_3 \rightarrow a$, $A_2 \rightarrow b$

$A_1 \rightarrow bA_3 \mid bA_4 \mid bA_3A_4A_4 \mid bZA_4 \mid bA_3A_4ZA$

$A_7 \rightarrow b \mid bA_3A_4 \mid bZ \mid bA_3A_4Z$

$Z \rightarrow bA_4 \mid bA_3A_4A_7 \mid bZA_4 \mid bA_3A_4ZA_4 \mid$

$bA_7 \mid bA_3A_4A_7 \mid bZA_7 \mid bA_3A_4ZA_7$

CamScanner

Design PDA for language $L = a^i b^j c^k$ where $i = j+k$