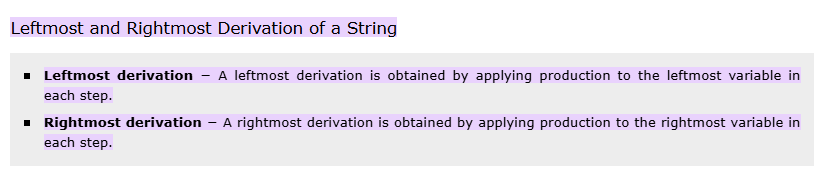
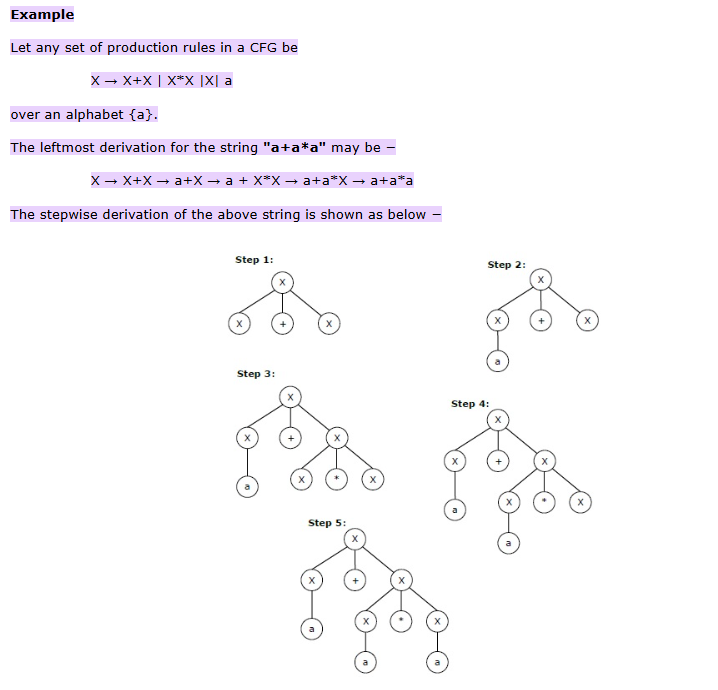
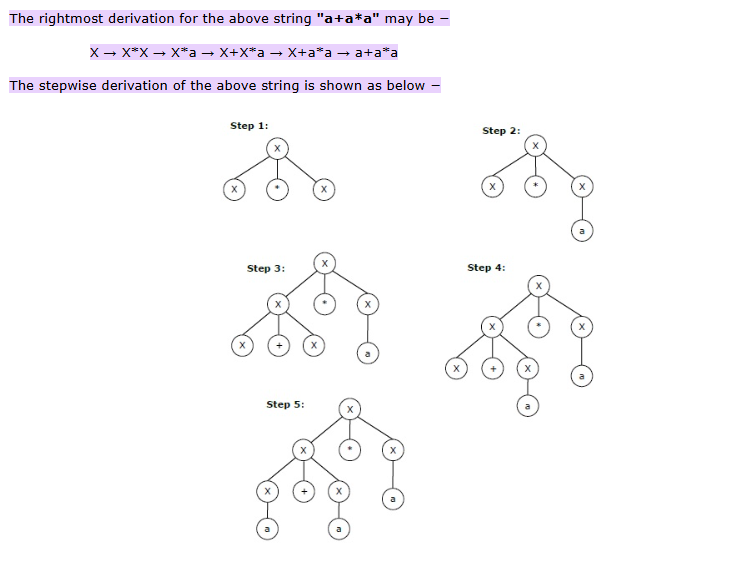
Unit-3

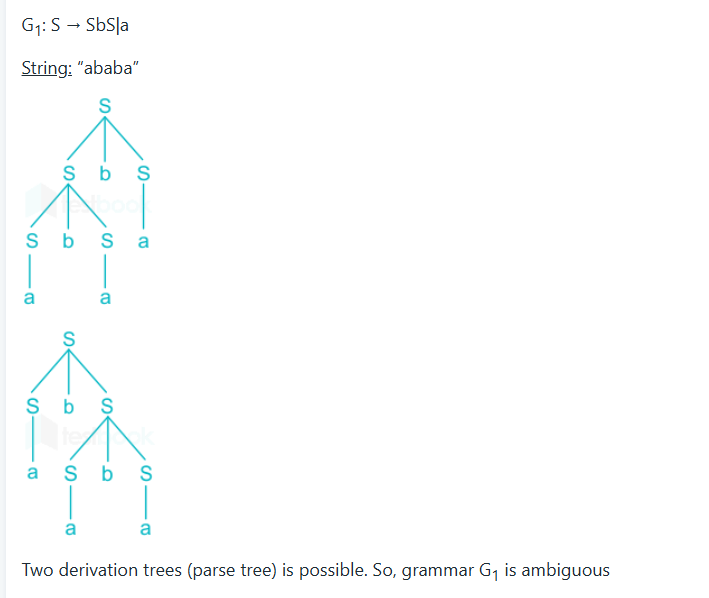
22. Explain rightmost derivation and left most derivation with an suitable examples.





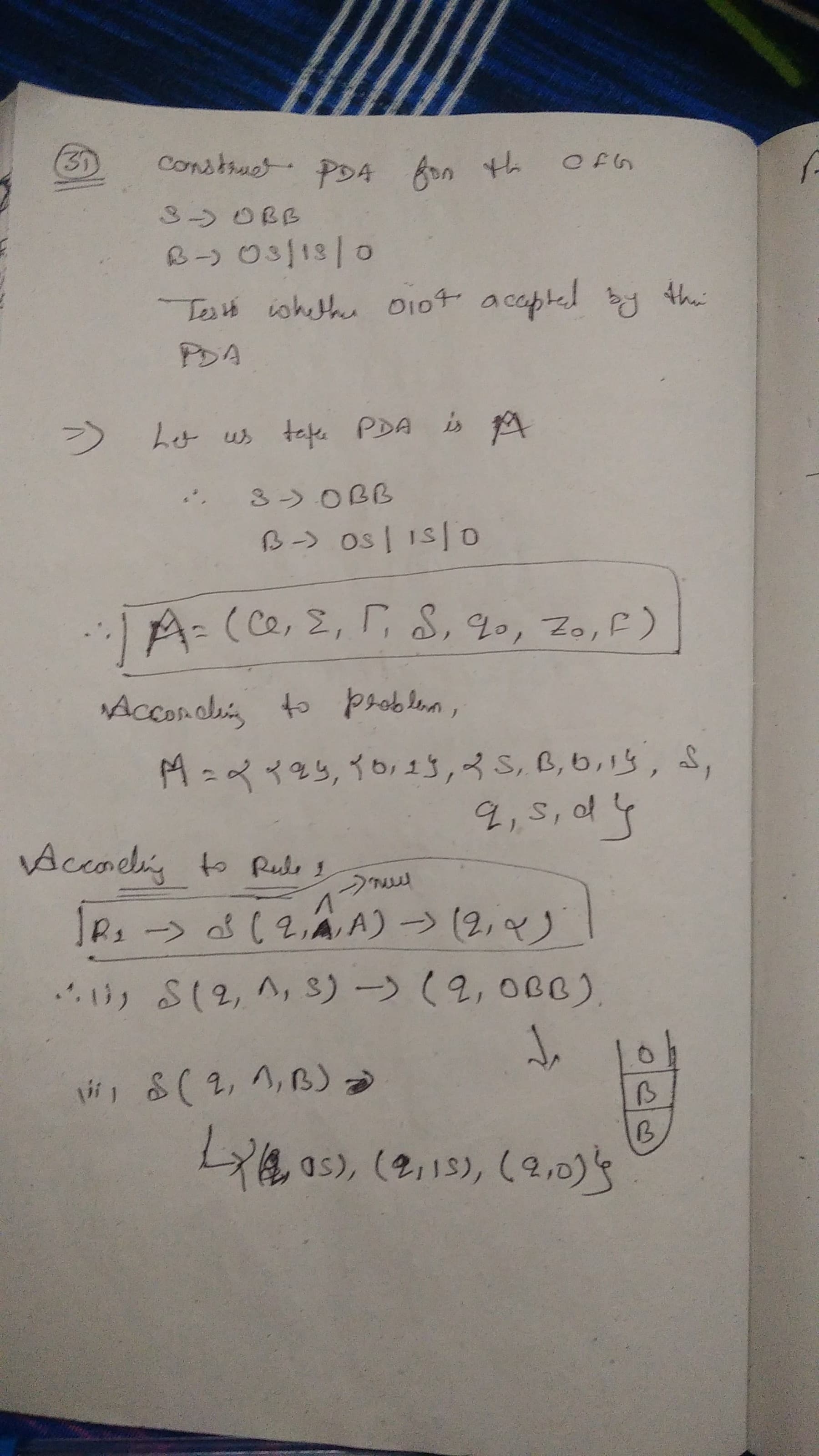


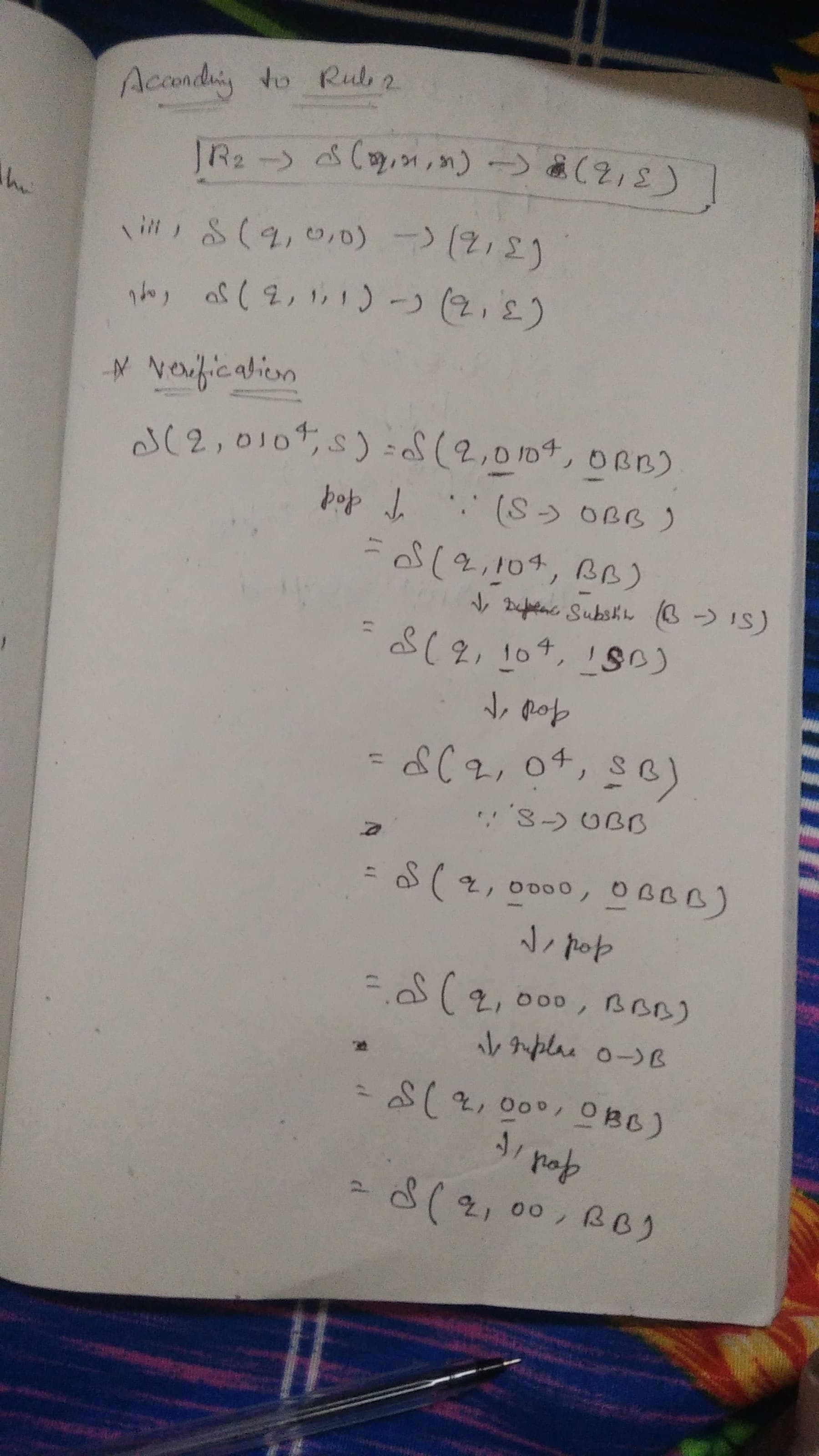
23. Show that the following grammar is ambiguous: S -> SbS/a

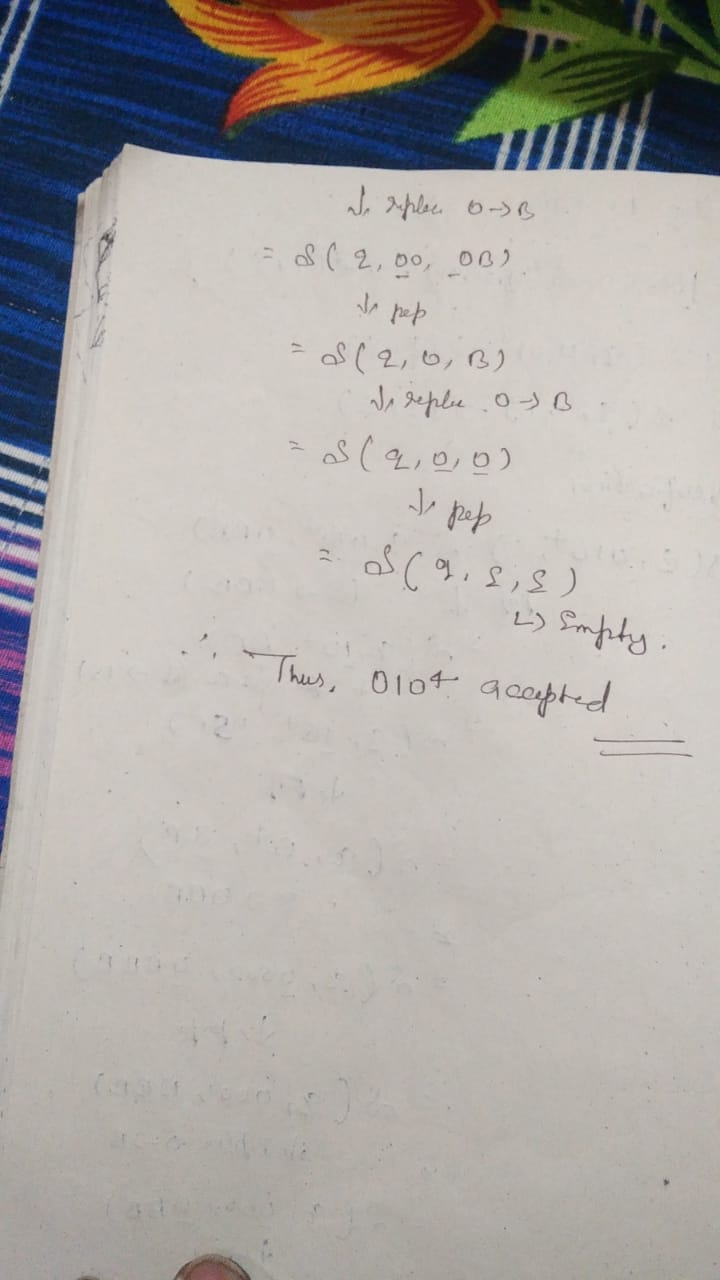


Unit-4

**31.**

****

****

****

**32.** Convert the following CFG into Chomsky’s

Normal Form (CNF)

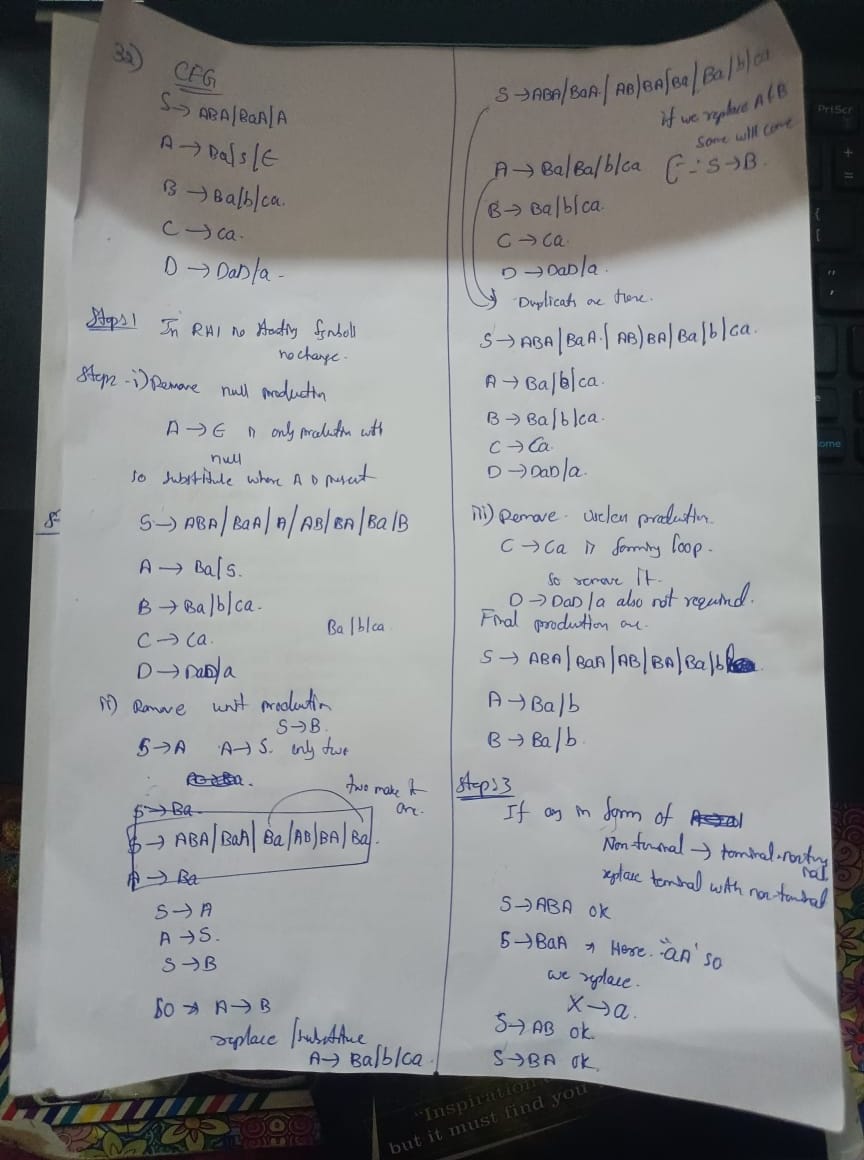
S→ A B A | B a A | A

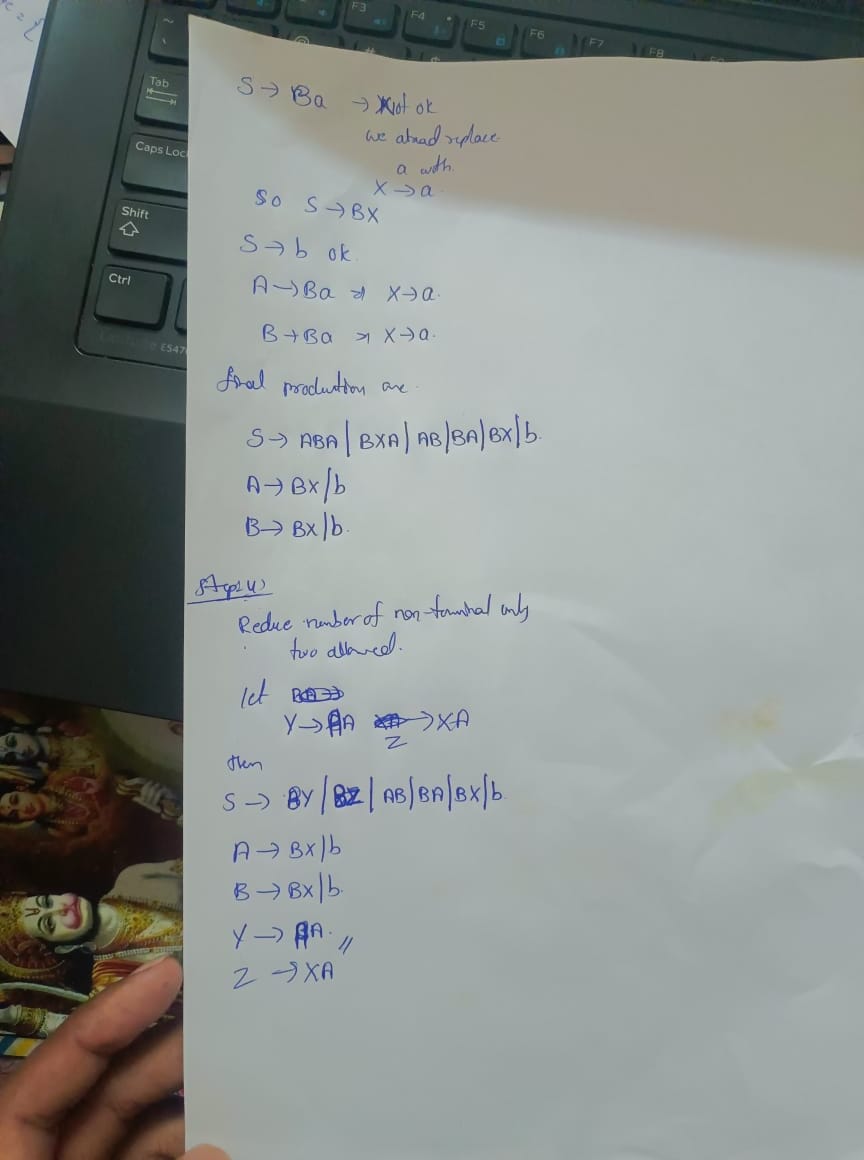
A→ B a | S | Ɛ

B → B a | b | Ca

C → C a

D → D a D | a





**33. Difference Between PDA and DPDA**

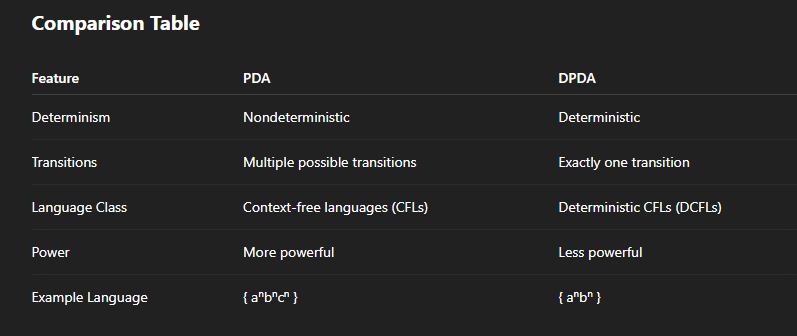
A **PDA** and a **DPDA** are both computational models used to recognize **context-free languages (CFLs)**, but they differ mainly in how they handle transitions — **nondeterminism vs determinism**.

**1. Pushdown Automaton (PDA)**

* **Nature**: **Nondeterministic**  
  A PDA can have **multiple possible transitions** for the same input, stack symbol, and state. It may explore multiple paths and accept if *any* path leads to an accepting state.
* **Memory**: Uses a **stack** to store symbols for processing.
* **Formal Definition**:  
  A PDA is defined by a 7-tuple:  
  **(Q, Σ, Γ, δ, q₀, Z₀, F)**  
  Where:
  + **Q**: Set of states
  + **Σ**: Input alphabet
  + **Γ**: Stack alphabet
  + **δ**: Transition function
  + **q₀**: Initial state
  + **Z₀**: Initial stack symbol
  + **F**: Set of accepting states
* **Transition Function**:  
  δ(q, a, X) → *set* of possible (next state, stack action)  
  Allows multiple options (nondeterministic).
* **Language Power**:  
  Recognizes **context-free languages (CFLs)**.
* **Example Language**:  
  **L = { aⁿbⁿcⁿ | n ≥ 0 }**  
  This language is CFL but **not** deterministic — requires guessing when to switch from reading b's to c's.

**2. Deterministic Pushdown Automaton (DPDA)**

* **Nature**: **Deterministic**  
  A DPDA must have **exactly one possible transition** for each combination of input symbol, stack symbol, and current state. No ambiguity is allowed.
* **Memory**: Also uses a **stack** like PDA.
* **Formal Definition**:  
  Also defined as a 7-tuple:  
  **(Q, Σ, Γ, δ, q₀, Z₀, F)**  
  But with the restriction that δ is **deterministic**.
* **Transition Function**:  
  δ(q, a, X) → *exactly one* (next state, stack action)  
  No multiple choices allowed.
* **Language Power**:  
  Recognizes **deterministic context-free languages (DCFLs)**, which are a **subset** of CFLs.
* **Example Language**:  
  **L = { aⁿbⁿ | n ≥ 0 }**  
  This language can be accepted **deterministically** by matching each "a" with a "b".



**25. Explain about pumping lemma algorithm**

<https://www.youtube.com/watch?v=KyQc054-BEU>

**26. Write about closure properties of context free language**

**1.Union Property**

If you have two context-free languages, L1 and L2, the union of these two, represented as L1∪L2, will also be a context-free language.

**Example**

Let's say L1 = { axby , x > 0}

The corresponding grammar G1 would be P: S1 → aAb | ab

And if L2 = { czdz , z ≥ 0}

The corresponding grammar G2 would be P: S2 → cBb| ε

The union of L1 and L2 would be L = L1 ∪ L2 = { axby } ∪ { czdz }

Here, the corresponding grammar G would have the additional production, that is, S → S1 | S2

## 2.Concatenation Property

If L1 and L2 are CFLs, then the concatenation of these two, represented as L1L2, will also be a context-free language.

### Example

The concatenation of the languages L1 and L2 would be L = L1L2 = { axbyczdz }

The corresponding grammar G would have the additional production, that is, S → S1 S2

## 3.Kleene Star Property

If L is a CFL, then the Kleene Star of L, represented as L\*, will also be a context-free language.

### Example

If L = { axby , x ≥ 0}

Then, the corresponding grammar G would have P: S → aAb| ε

Thus, the Kleene Star L1 = { axby }\*

Here, the corresponding grammar G1 would have additional productions, and they are S1 → SS1 | ε

However, CFLs are not closed under the following operations:

* **Intersection**− If L1 and L2 are CFLs, then the intersection of these two, represented as L1 ∩ L2, may not be a CFL.
* **Intersection with a regular language**− If L1 is a regular language and L2 is a CFL, then the intersection of these two, represented as L1 ∩ L2, will be a CFL.
* **Complement**− If L1 is a CFL, the complement of L1, represented as L1’, may not be a CFL

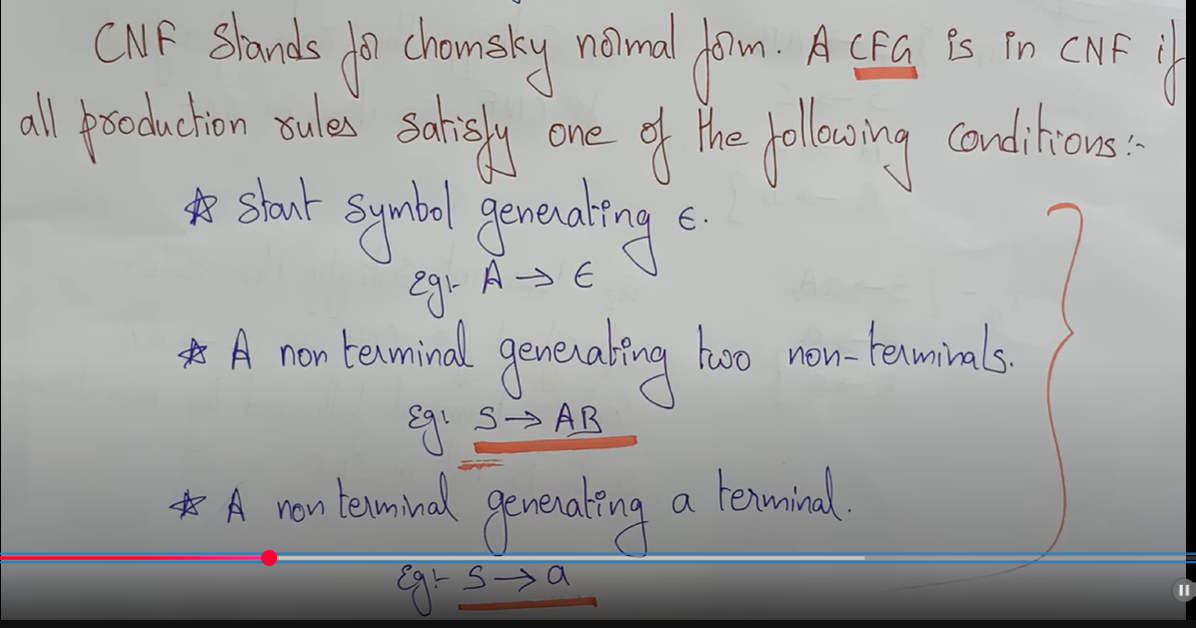
**27. Enumerate normal forms for context free language**

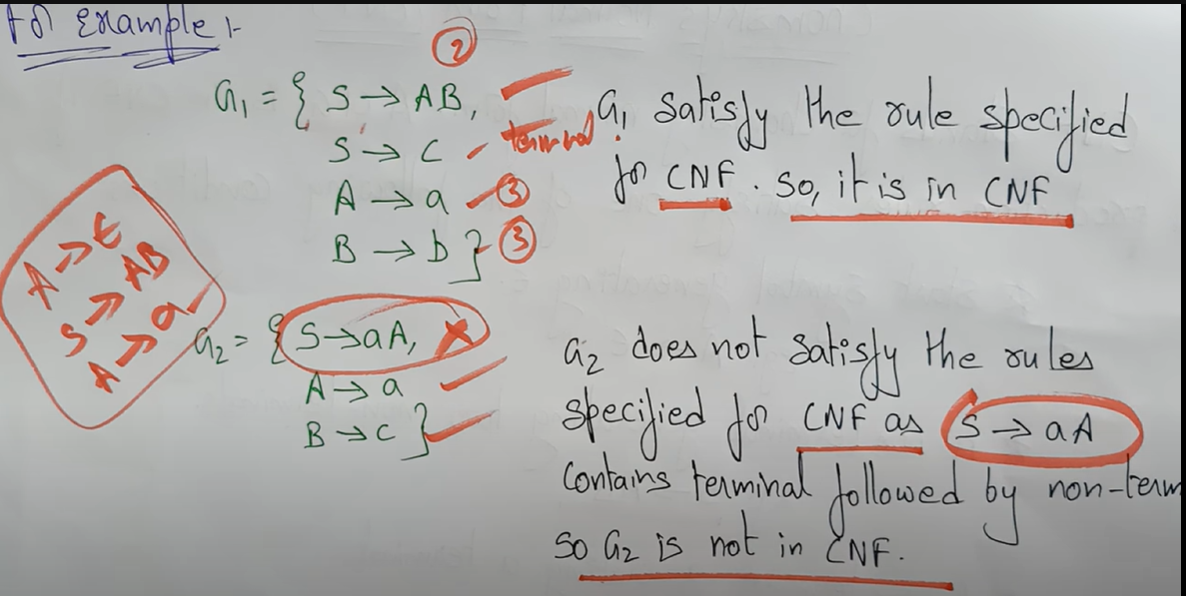
Ans:

There are two primary normal forms for Context-Free Grammars (CFGs) that are:

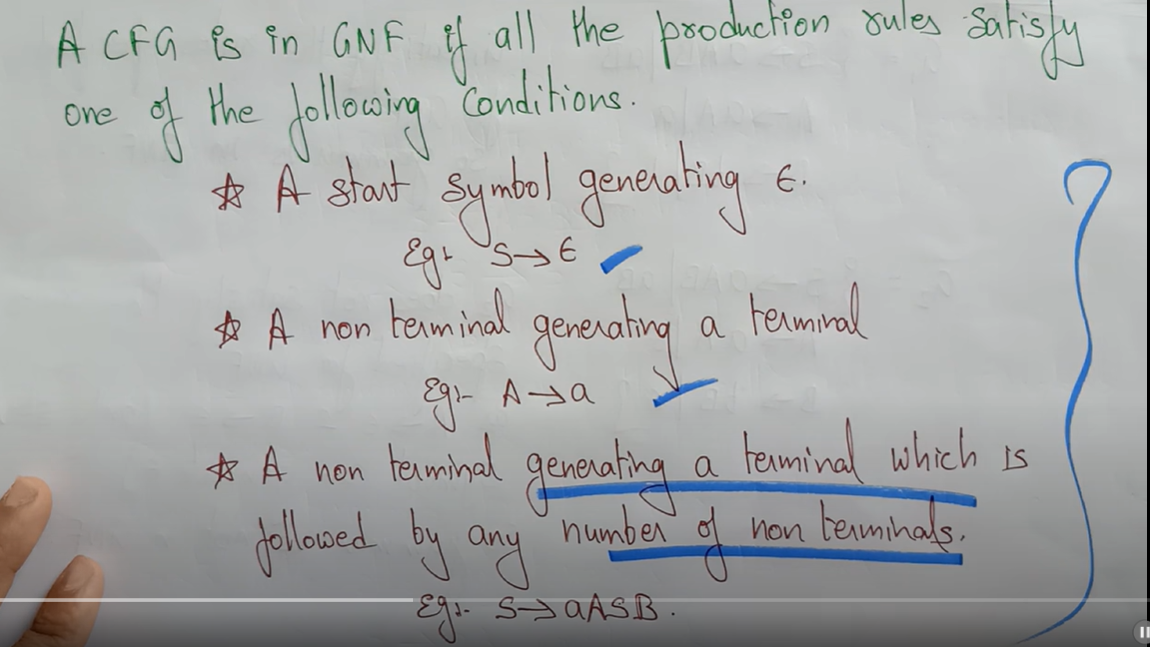
1. **Chomsky Normal Form (CNF)**
2. **Greibach Normal Form (GNF)**

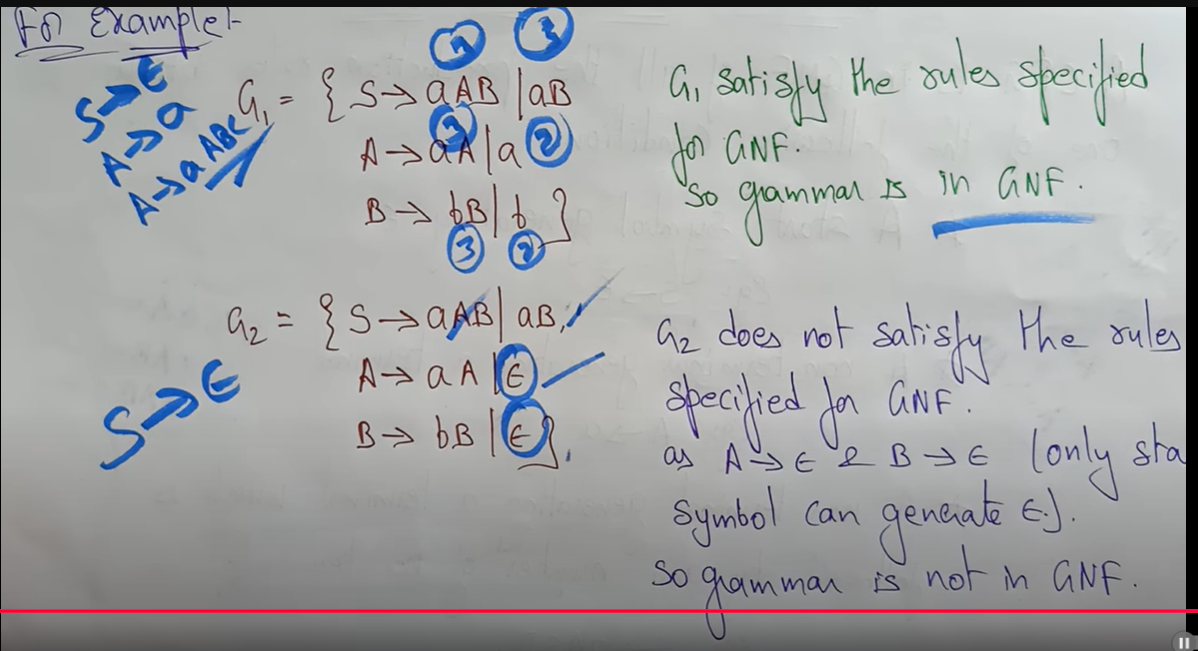
**1.Chomsky Normal Form(CNF)**





**2. Greibach Normal Form (GNF)**





Start

**28. Convert the following context free language to**

**CNF**

**S -> ABC**

**A -> Aa/epsilon**

**B -> bB/epsilon**

**C -> cC/epsilon**

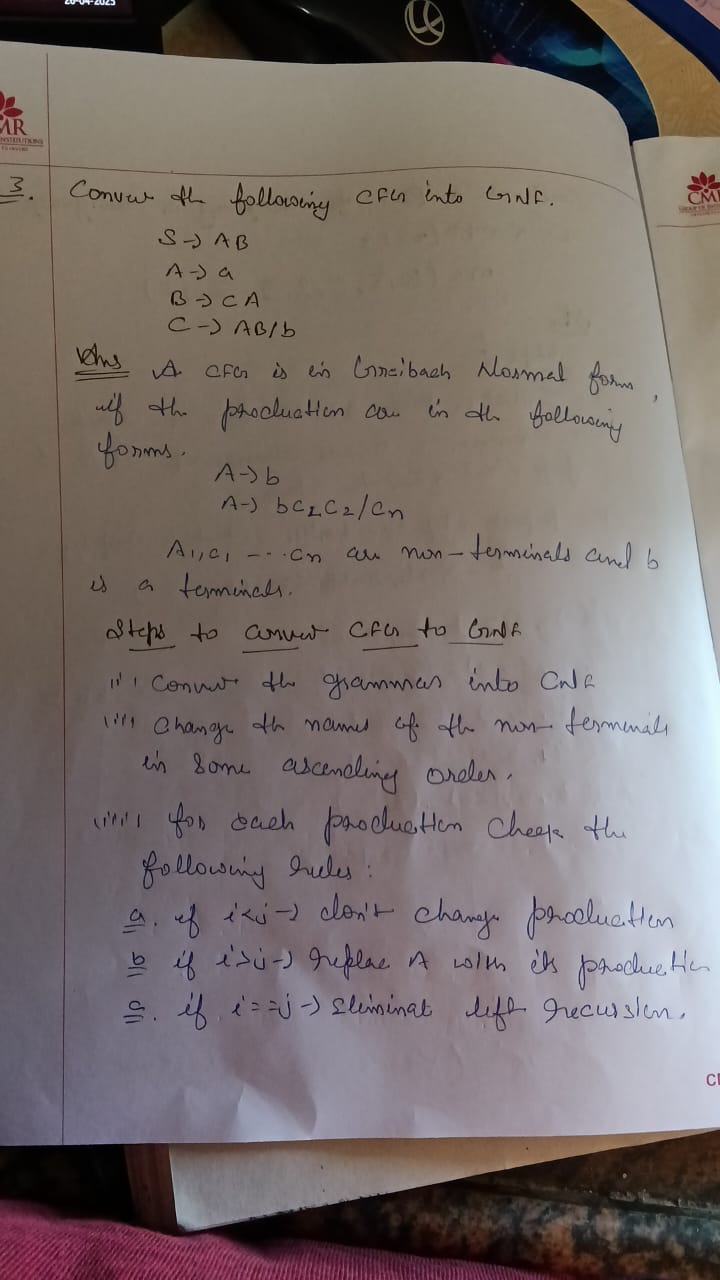
**29.** **Convert the following CFG into GNF.**

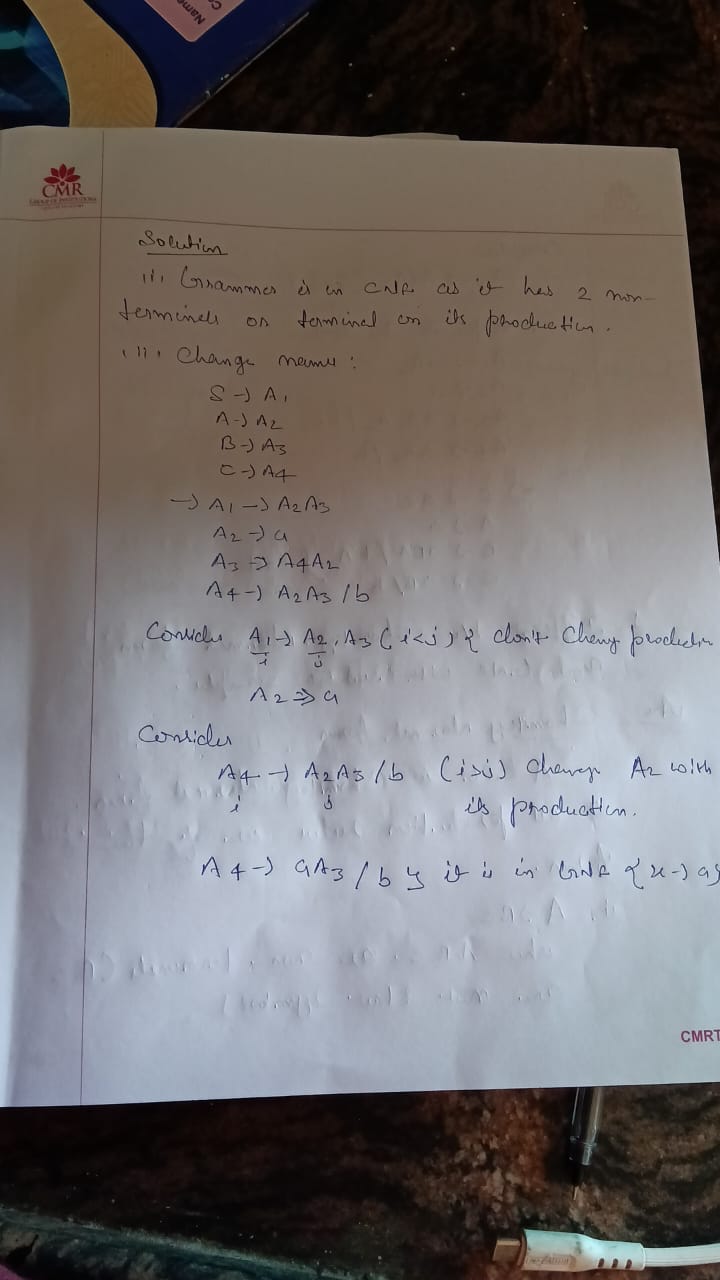
**S->AB**

**A->a**

**B-> CA**

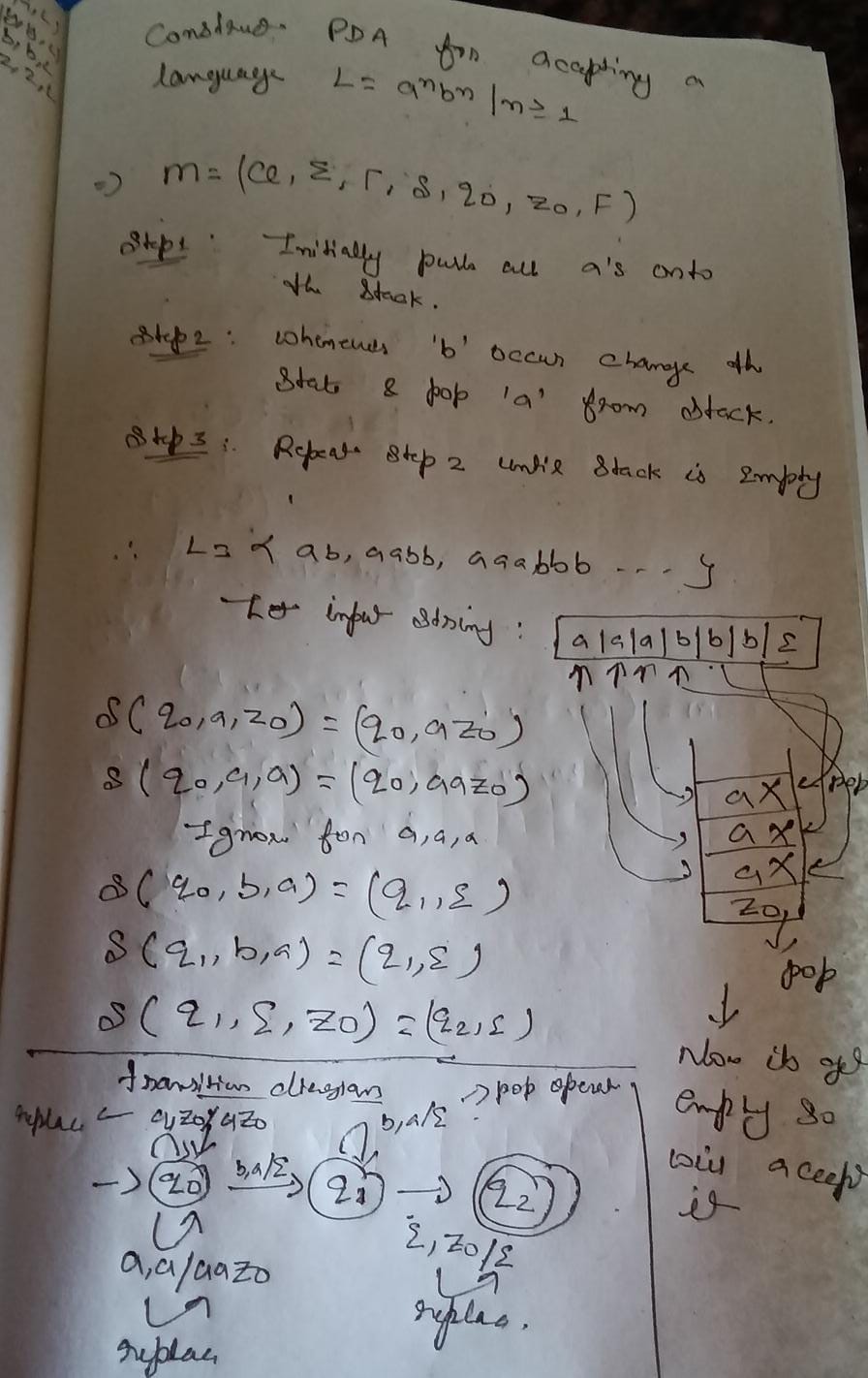
**C->AB/b**

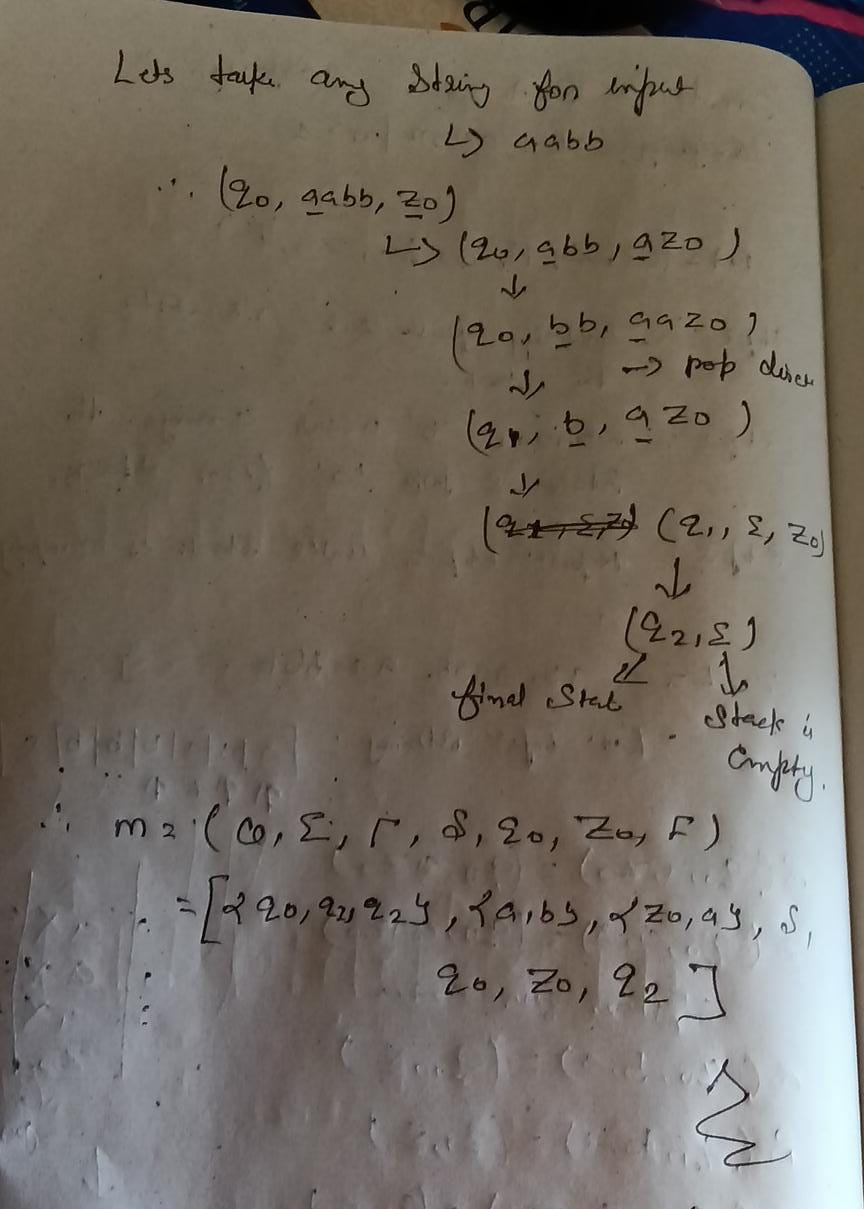
****

****

**30.** **Construct a PDA for accepting a language**

**{L=a^nb^n | n ≥ 1}**

****

****

**31. Construct PDA for the given CFG**

**S→0BB**

**B→0S|1S|0**

**Test whether 01044 is accepted by this PDA**

[**https://www.naukri.com/code360/library/cfg-to-pda-conversion**](https://www.naukri.com/code360/library/cfg-to-pda-conversion)

**Unit-5**

**40. short notes on:**

**i) P ii) NP iii) NP Hard iv) NP Complete with example**

**NP Class**

The NP in NP class stands for **Non-deterministic Polynomial Time**. It is the collection of decision problems that can be solved by a non-deterministic machine (note that our computers are deterministic) in polynomial time.

**Features:**

* The solutions of the NP class might be hard to find since they are being solved by a non-deterministic machine but the solutions are easy to verify.
* Problems of NP can be verified by a deterministic machine in polynomial time.

**Example:**

Let us consider an example to better understand the **NP class**. Suppose there is a company having a total of **1000** employees having unique employee **IDs**. Assume that there are **200** rooms available for them. A selection of **200** employees must be paired together, but the CEO of the company has the data of some employees who can’t work in the same room due to personal reasons.  
This is an example of an **NP**problem. Since it is easy to check if the given choice of **200** employees proposed by a coworker is satisfactory or not i.e. no pair taken from the coworker list appears on the list given by the CEO. But generating such a list from scratch seems to be so hard as to be completely impractical.

It indicates that if someone can provide us with the solution to the problem, we can find the correct and incorrect pair in polynomial time. Thus for the **NP** class problem, the answer is possible, which can be calculated in polynomial time.

This class contains many problem

# -Satisfiability (2-SAT) Problem

Last Updated : 29 Apr, 2024

## Boolean Satisfiability Problem

Boolean Satisfiability or simply **SAT** is the problem of determining if a Boolean formula is satisfiable or unsatisfiable.

* **Satisfiable :**If the Boolean variables can be assigned values such that the formula turns out to be TRUE, then we say that the formula is satisfiable.
* **Unsatisfiable :** If it is not possible to assign such values, then we say that the formula is unsatisfiable.

**Examples:**

* F=A∧Bˉ      *F*=*A*∧*B*ˉ , is satisfiable, because A = TRUE and B = FALSE makes F = TRUE.
* G=A∧Aˉ      *G*=*A*∧*A*ˉ , is unsatisfiable, because:

| **A*A*** | **Aˉ*A*ˉ** | **G*G*** |
| --- | --- | --- |
| **TRUE** | **FALSE** | **FALSE** |
| **FALSE** | **TRUE** | **FALSE** |

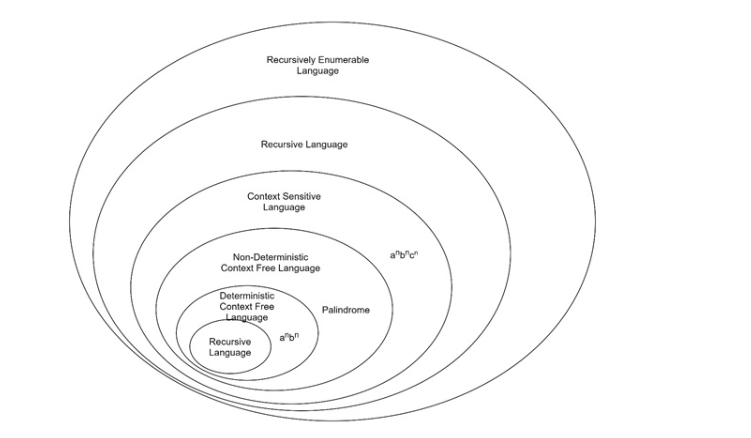
**41. Illustrate the process of Recursive languages and Recursively enumerable Languages with suitable examples?**

**Ans:**

## Recursively Enumerable Languages

In simple words, a "language" is a collection of strings, like words in a dictionary. A recursively enumerable language is a language where we can create a computer program (or a Turing machine) that can systematically list out all the strings that belong to the language.

Consider a machine that can generate all the possible sentences in the English language, one by one. This machine wouldn't necessarily know which sentences are not in the English language, but it could list out all the valid sentences. This is the idea of a recursively enumerable language. It can enumerate all the strings that are part of the language.

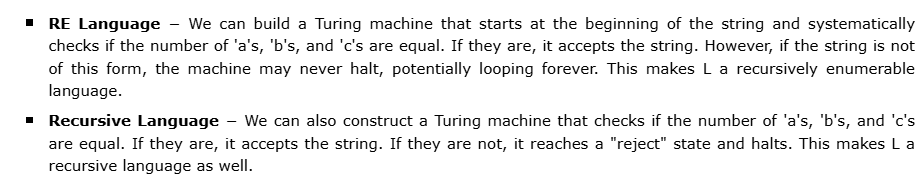


.

## Recursive Languages: A Subset of RE Languages

Another important subset of RE languages is recursive language. In a recursive language, the Turing machine not only accepts strings belonging to the language but also always halts for strings that are not in the language

As an example, Consider the language L={anbncn|n≥0}L={anbncn|n≥0}. This language consists of strings where the number of a's, b's, and c's are equal

****

## Closure Properties of Recursive Languages

## 

**42.** Construct a Turing Machine

L={ a^n b^n c^n | n ≥ 1 }

