Unit-3

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

Leftmost and Rightmost Derivation of a String

- **Leftmost derivation** – A leftmost derivation is obtained by applying production to the leftmost variable in each step.
- **Rightmost derivation** – A rightmost derivation is obtained by applying production to the rightmost variable in each step.

Example

Let any set of production rules in a CFG be

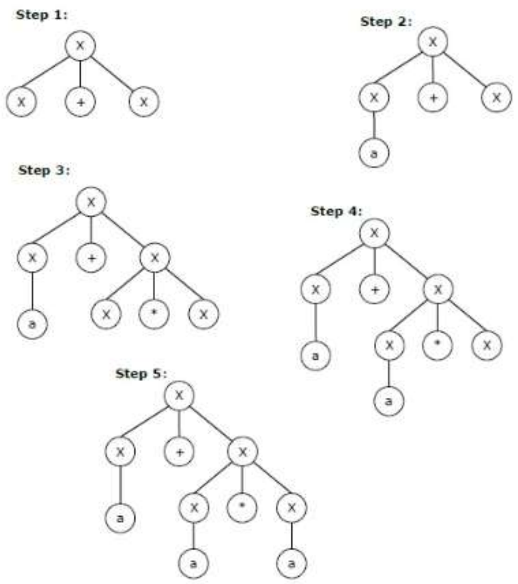
X → X+X | X*X |X| a

over an alphabet {a}.

The leftmost derivation for the string **"a+a*a"** may be –

X → X+X → a+X → a + X*X → a+a*X → a+a*a

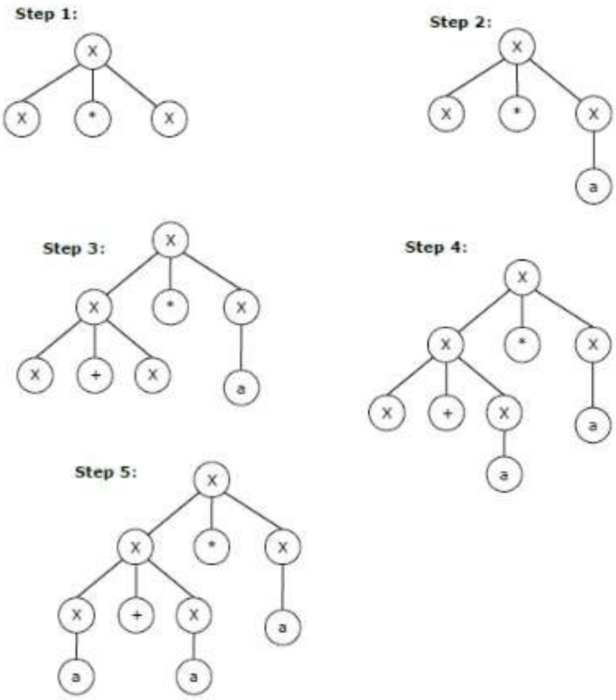
The stepwise derivation of the above string is shown as below –



The rightmost derivation for the above string **"a+a*a"** may be –

X → X*X → X*a → X+X*a → X+a*a → a+a*a

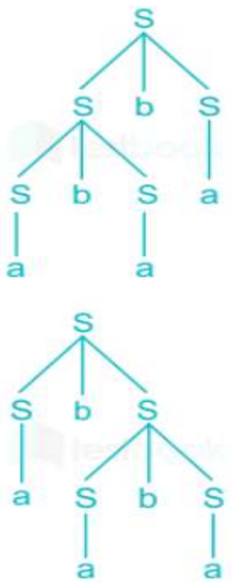
The stepwise derivation of the above string is shown as below –



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

G_1: S → SbS|a

String: "ababa"



Two derivation trees (parse tree) is possible. So, grammar G_1 is ambiguous

Unit-4

31.

(31) Construct PDA for the CFG

$S \rightarrow 0BB$

$B \rightarrow 0S \mid 1S \mid 0$

Test whether 010^4 accepted by the PDA

$\Rightarrow$ Let us take PDA is A

$\therefore S \rightarrow 0BB$

$B \rightarrow 0S \mid 1S \mid 0$

$$\therefore \boxed{A = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)}$$

According to problem,

$A = (\{q\}, \{0,1\}, \{S, B, 0, 1\}, \delta, q, S, \phi)$

According to Rule:

$$\boxed{R_1 \rightarrow \delta(q, \overset{\text{null}}{\Lambda}, A) \rightarrow (q, \alpha)}$$

$\therefore$ (i) $\delta(q, \Lambda, S) \rightarrow (q, 0BB)$

0
B
B

(ii) $\delta(q, \Lambda, B) \Rightarrow \{(q, 0S), (q, 1S), (q, 0)\}$

According to Rule 2

$$\boxed{R_2 \to \delta(q, x, x) \to (q, \varepsilon)}$$

iii) $\delta(q, 0, 0) \to (q, \varepsilon)$

iv) $\delta(q, 1, 1) \to (q, \varepsilon)$

Verification

$\delta(q, 010^4, S) = \delta(q, \underline{0}10^4, \underline{0}BB)$

pop ↓ ∵ $(S \to 0BB)$

$= \delta(q, \underline{1}0^4, \underline{B}B)$

↓ replace Substitute $(B \to 1S)$

$= \delta(q, \underline{1}0^4, \underline{1}SB)$

↓, pop

$= \delta(q, 0^4, \underline{S}B)$

∵ $S \to 0BB$

$= \delta(q, \underline{0}000, \underline{0}BBB)$

↓, pop

$= \delta(q, 000, BBB)$

↓ replace $0 \to B$

$= \delta(q, \underline{0}00, \underline{0}BB)$

↓, pop

$= \delta(q, 00, BB)$

$\downarrow$ replace $0 \to B$

$= \delta(q, \underline{0}0, \underline{0}B)$

$\downarrow$ pop

$= \delta(q, 0, B)$

$\downarrow$ replace $0 \to B$

$= \delta(q, \underline{0}, \underline{0})$

$\downarrow$ pop

$= \delta(q, \varepsilon, \varepsilon)$

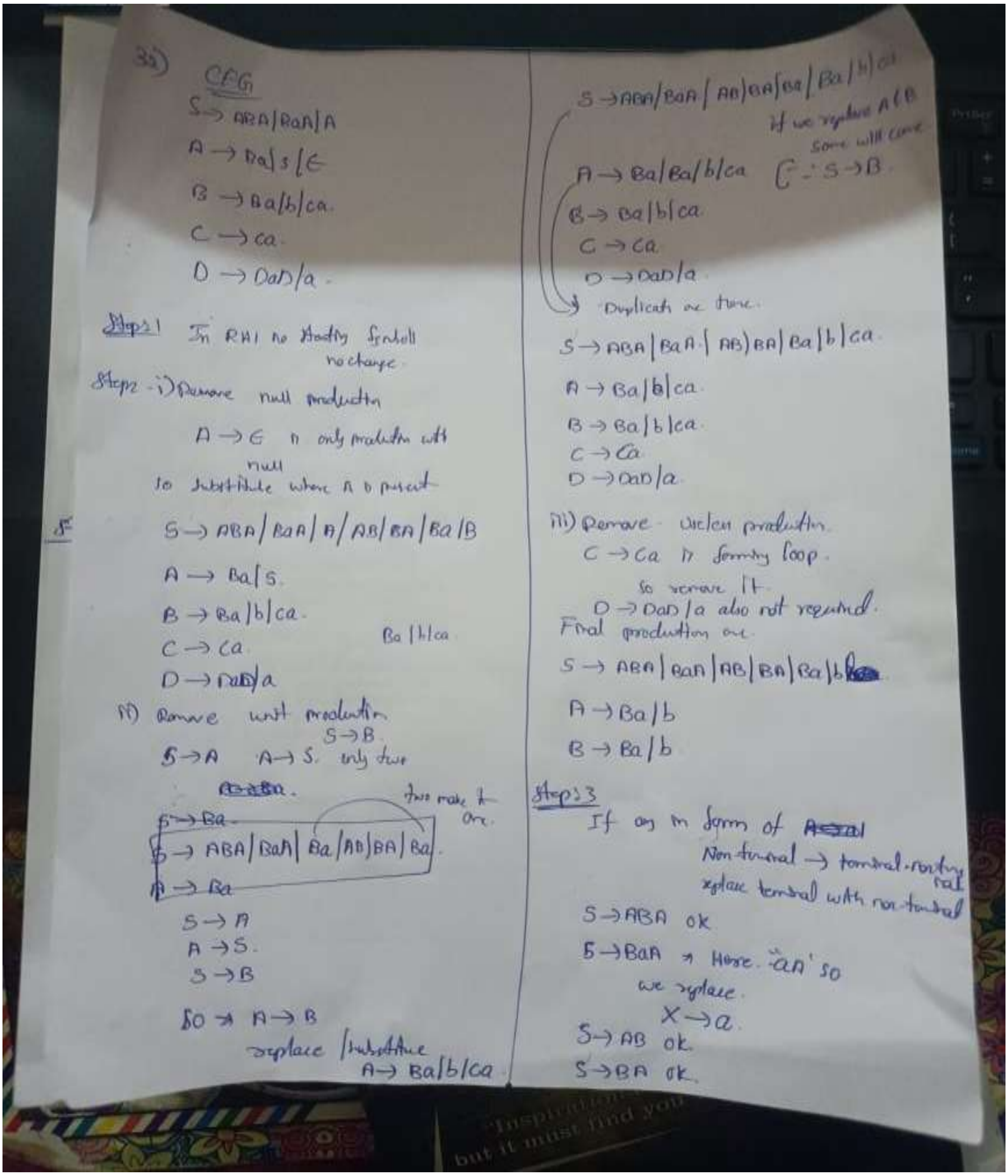
$\hookrightarrow$ Empty.

$\therefore$ Thus, 0104 accepted

32. Convert the following CFG into Chomsky's

Normal Form (CNF)
$S \rightarrow A\ B\ A \mid B\ a\ A \mid A$
$A \rightarrow B\ a \mid S \mid \varepsilon$
$B \rightarrow B\ a \mid b \mid Ca$
$C \rightarrow C\ a$

$D \rightarrow D\ a\ D \mid a$



32)

CFG

$S \rightarrow ABA \mid BaA \mid A$
$A \rightarrow Ba \mid S \mid \epsilon$
$B \rightarrow Ba \mid b \mid ca$
$C \rightarrow ca$
$D \rightarrow DaD \mid a$

Step 1 In RHS no starting symbol
no change

Step 2 i) Remove null production

$A \rightarrow \epsilon$ is only production with null
so substitute where A is present

$S \rightarrow ABA \mid BaA \mid A \mid AB \mid BA \mid Ba \mid B$
$A \rightarrow Ba \mid S$
$B \rightarrow Ba \mid b \mid ca$ Ba | b | ca
$C \rightarrow Ca$
$D \rightarrow DaD \mid a$

ii) Remove unit production

$S \rightarrow A$ $A \rightarrow S$ $S \rightarrow B$ only two ~~[illegible]~~.

two make it one.

~~$S \rightarrow Ba$~~
$S \rightarrow ABA \mid BaA \mid Ba \mid AB \mid BA \mid Ba$
~~$A \rightarrow Ba$~~

$S \rightarrow A$
$A \rightarrow S$
$S \rightarrow B$

so $A \rightarrow B$
replace / substitute
$A \rightarrow Ba \mid b \mid ca$

$S \rightarrow ABA \mid BaA \mid AB \mid BA \mid Ba \mid Ba \mid b \mid ca$
If we replace A & B same will come
$A \rightarrow Ba \mid Ba \mid b \mid ca$ $\because S \rightarrow B$
$B \rightarrow Ba \mid b \mid ca$
$C \rightarrow Ca$
$D \rightarrow DaD \mid a$
Duplicate are there.

$S \rightarrow ABA \mid BaA \mid AB \mid BA \mid Ba \mid b \mid ca$
$A \rightarrow Ba \mid b \mid ca$
$B \rightarrow Ba \mid b \mid ca$
$C \rightarrow Ca$
$D \rightarrow DaD \mid a$

iii) Remove useless production
$C \rightarrow Ca$ is forming loop.
so remove it.
$D \rightarrow DaD \mid a$ also not required.
Final production are

$S \rightarrow ABA \mid BaA \mid AB \mid BA \mid Ba \mid b$
$A \rightarrow Ba \mid b$
$B \rightarrow Ba \mid b$

Step 3

If any in form of ~~[illegible]~~
Non terminal → terminal. non terminal
replace terminal with non-terminal

$S \rightarrow ABA$ ok
$S \rightarrow BaA$ → Here 'aA' so
we replace
$X \rightarrow a$.
$S \rightarrow AB$ ok.
$S \rightarrow BA$ ok.

$S \to Ba \to$ Not ok

we alread replace
a with
$X \to a$

So $S \to BX$

$S \to b$ ok

$A \to Ba \Rightarrow X \to a$

$B \to Ba \Rightarrow X \to a$.

final production are

$S \to ABA \mid BXA \mid AB \mid BA \mid BX \mid b$.

$A \to BX \mid b$

$B \to BX \mid b$.

Step (4)

Reduce number of non-terminal only two allowed.

let ~~BA~~ $\Rightarrow$

$Y \to BA \quad \underset{Z}{\cancel{XA}} \to XA$

then

$S \to BY \mid \cancel{BZ} \mid AB \mid BA \mid BX \mid b$.

$A \to BX \mid b$

$B \to BX \mid b$.

$Y \to BA$. //

$Z \to XA$

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:
  ($Q, \Sigma, \Gamma, \delta, q_0, Z_0, F$)
  Where:
  - **Q**: Set of states
  - **Σ**: Input alphabet
  - **Γ**: Stack alphabet
  - **δ**: Transition function
  - **q_0**: Initial state
  - **Z_0**: Initial stack symbol
  - **F**: Set of accepting states
- **Transition Function**:
  $\delta(q, a, X) \rightarrow$ *set* of possible (next state, stack action)
  Allows multiple options (nondeterministic).
- **Language Power**:
  Recognizes **context-free languages (CFLs)**.
- **Example Language**:
  $L = \{ a^n b^n c^n \mid n \geq 0 \}$
  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:
  ($Q, \Sigma, \Gamma, \delta, q_0, Z_0, F$)
  But with the restriction that δ is **deterministic**.
- **Transition Function**:
  $\delta(q, a, X) \rightarrow$ *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 = { $a^n b^n$ | n ≥ 0 }
  This language can be accepted **deterministically** by matching each "a" with a "b".

Comparison Table

Feature	PDA	DPDA
Determinism	Nondeterministic	Deterministic
Transitions	Multiple possible transitions	Exactly one transition
Language Class	Context-free languages (CFLs)	Deterministic CFLs (DCFLs)
Power	More powerful	Less powerful
Example Language	{ $a^n b^n c^n$ }	{ $a^n b^n$ }

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

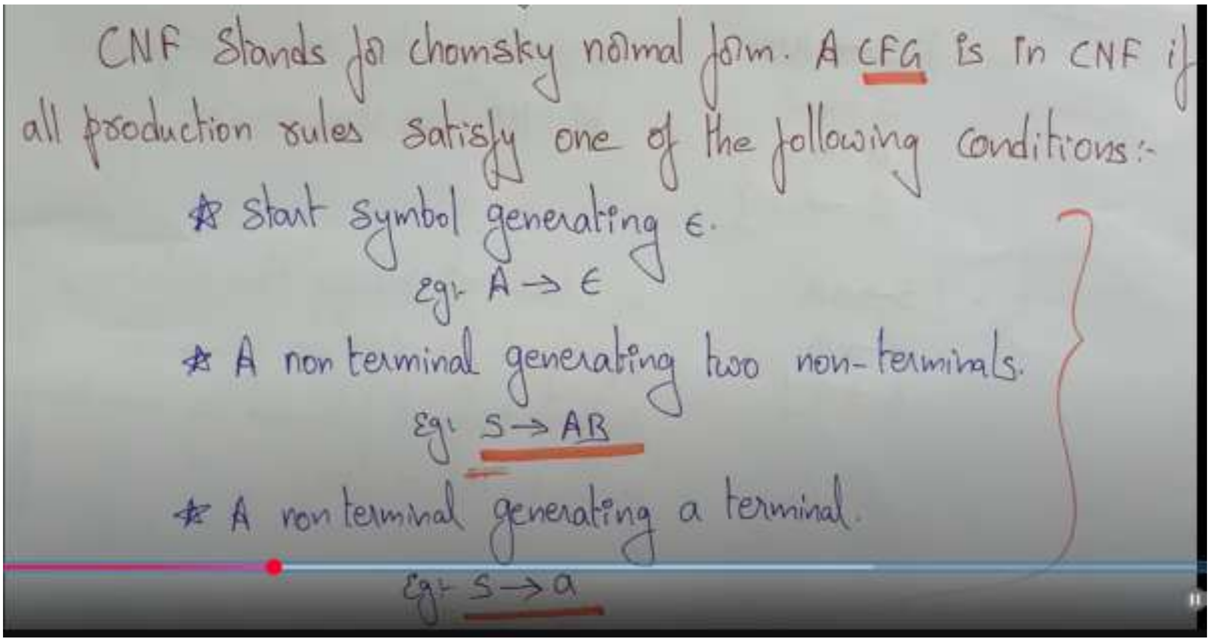
27. Enumerate normal forms for context free language

Ans:

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

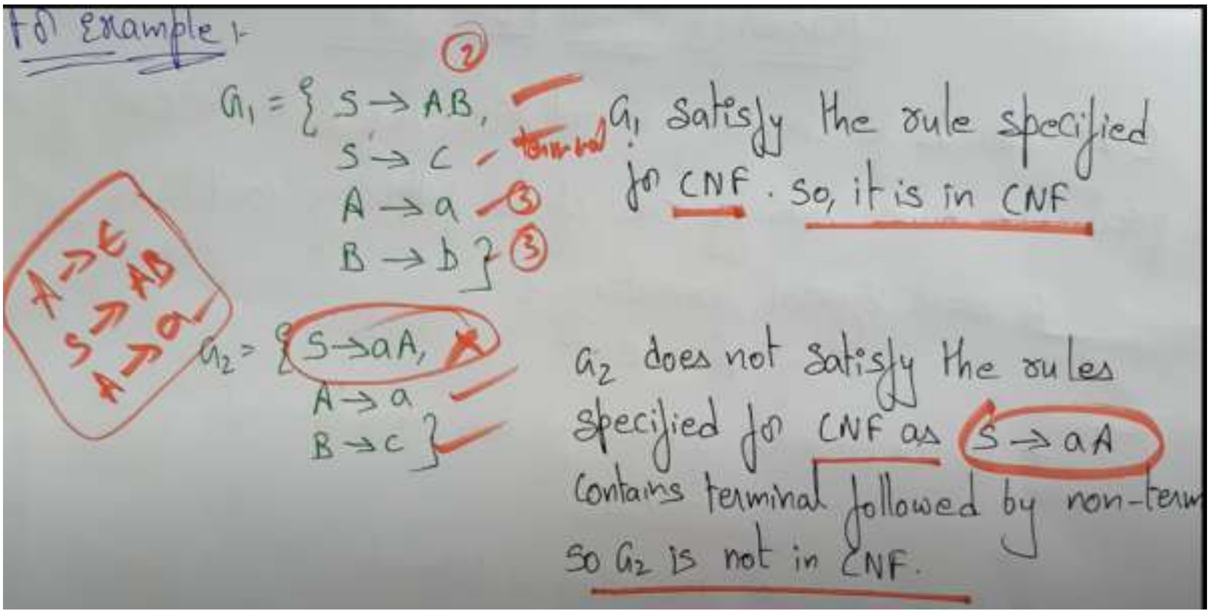
1. **Chomsky Normal Form (CNF)**
2. **Greibach Normal Form (GNF)**

1.Chomsky Normal Form(CNF)



CNF Stands for chomsky normal form. A CFG is in CNF if all production rules satisfy one of the following conditions:-

- Start symbol generating ϵ.
  eg:- $A \rightarrow \epsilon$
- A non terminal generating two non-terminals.
  eg:- $S \rightarrow AB$
- A non terminal generating a terminal.
  eg:- $S \rightarrow a$



For Example:-

$G_1 = \{ S \rightarrow AB,$ (2)
$S \rightarrow c$ terminal
$A \rightarrow a$ (3)
$B \rightarrow b \}$ (3)

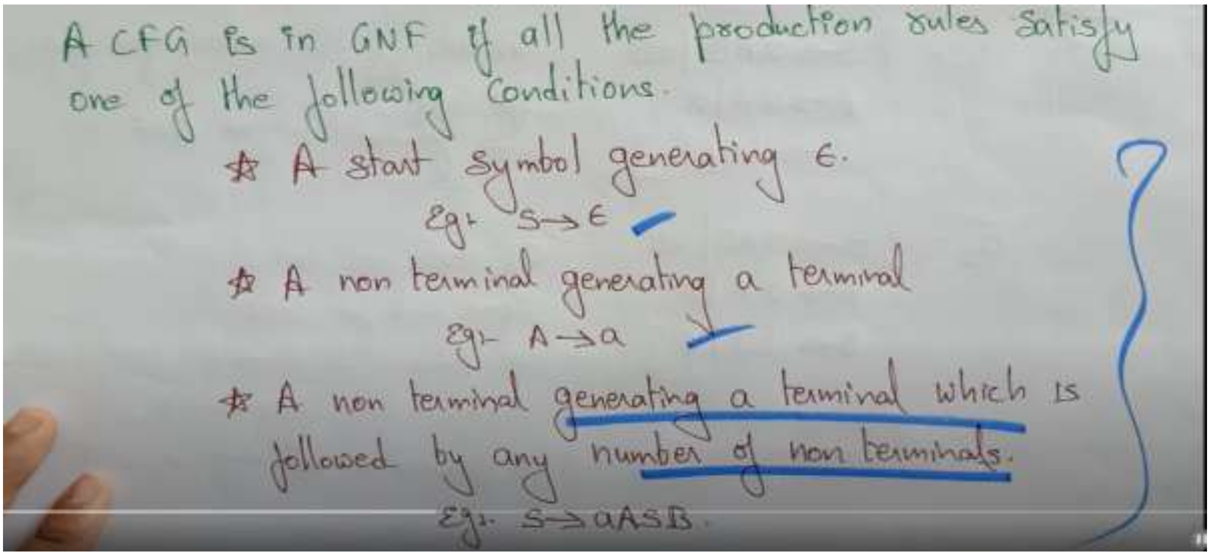
G_1 satisfy the rule specified for CNF. So, it is in CNF

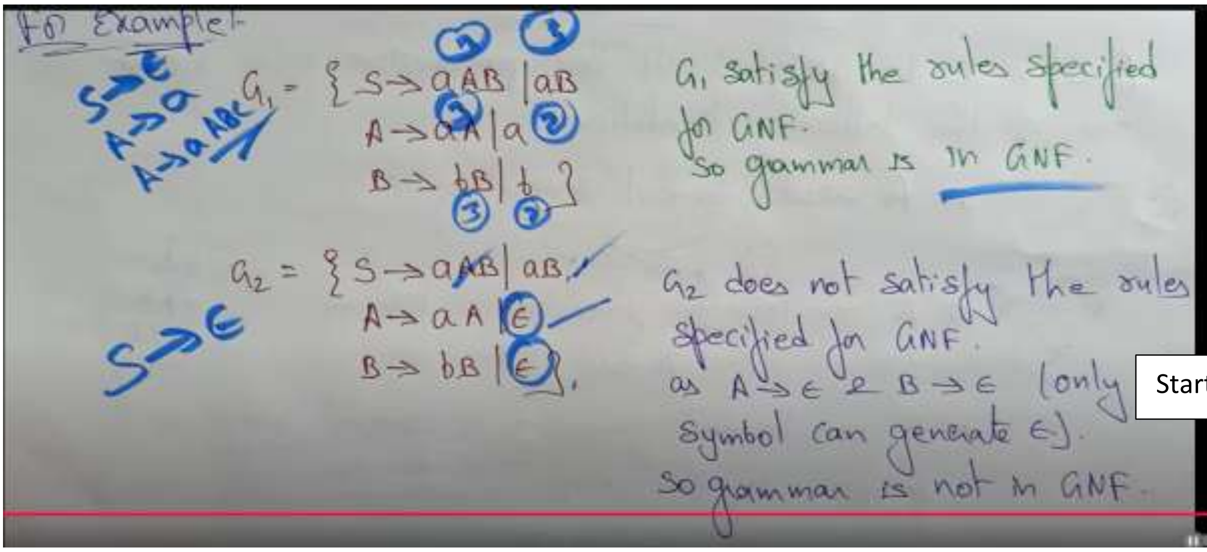
$A \rightarrow \epsilon$
$S \rightarrow AB$
$A \rightarrow a$

$G_2 = \{ S \rightarrow aA,$
$A \rightarrow a$
$B \rightarrow c \}$

G_2 does not satisfy the rules specified for CNF as $S \rightarrow aA$ contains terminal followed by non-term
So G_2 is not in 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

3. Convert the following CFG into GNF.

$S \to AB$
$A \to a$
$B \to CA$
$C \to AB/b$

Ans A CFG is in Greibach Normal form, if the production are in the following forms.

$A \to b$
$A \to bC_1C_2/C_n$

$A_1, C_1 \ldots C_n$ are non-terminals and b is a terminals.

Steps to convert CFG to GNF

i) Convert the grammar into CNF
ii) Change the names of the non-terminals in some ascending order.
iii) for each production check the following rules:
a. if i<j -> don't change production
b. if i>j -> replace A with its production
c. if i==j -> Eliminate left recursion.

Solution

i) Grammar is in CNF as it has 2 non-terminals or terminal on its production.

ii) Change names:

$S \to A_1$

$A \to A_2$

$B \to A_3$

$C \to A_4$

$\to A_1 \to A_2 A_3$

$A_2 \to a$

$A_3 \to A_4 A_2$

$A_4 \to A_2 A_3 / b$

Consider $\underset{i}{A_1} \to \underset{j}{A_2}, A_3$ ($i<j$) { don't change production

$A_2 \Rightarrow a$

Consider

$\underset{i}{A_4} \to \underset{j}{A_2} A_3 / b$ ($i>j$) change A_2 with its production.

$A_4 \to aA_3 / b$ } it is in GNF { $x \to a$ }

CMRT

30. Construct a PDA for accepting a language

{L=a^nb^n | n ≥ 1}

Construct PDA for accepting a language $L = a^n b^n \mid n \geq 1$

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

Step 1 : Initially push all a's onto the stack.

Step 2 : Whenever 'b' occur change the state & pop 'a' from stack.

Step 3 : Repeat step 2 until stack is empty.

$\therefore L = \{ab, aabb, aaabbb \ldots\}$

Let input string : | a | a | a | b | b | b | ε |

$\delta(q_0, a, z_0) = (q_0, a z_0)$

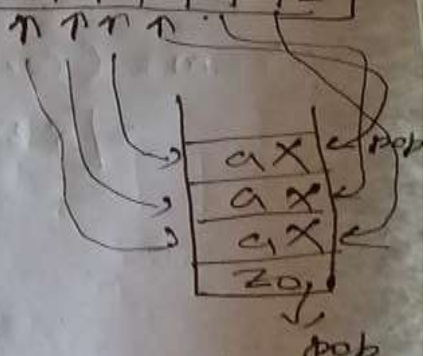
$\delta(q_0, a, a) = (q_0, aaz_0)$

Ignore for a, a, a

$\delta(q_0, b, a) = (q_1, \varepsilon)$

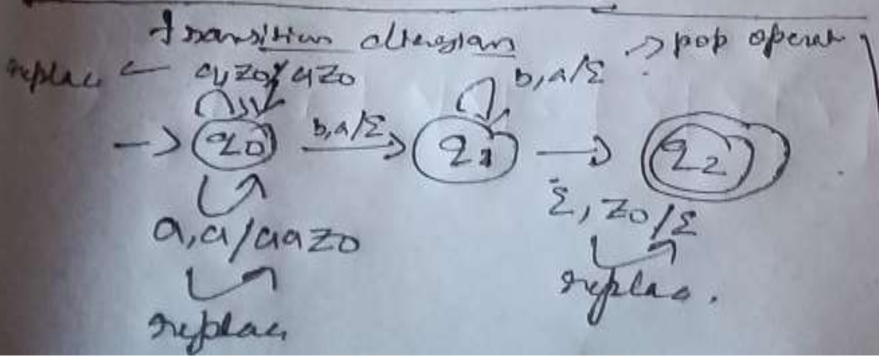
$\delta(q_1, b, a) = (q_1, \varepsilon)$

$\delta(q_1, \varepsilon, z_0) = (q_2, \varepsilon)$



Now it's empty so will accept it

Transition diagram



Lets take any String for input

↳ aabb

∴ $(q_0, \underline{a}abb, \underline{Z_0})$

↳ $(q_0, \underline{a}bb, \underline{a}Z_0)$

↓

$(q_0, \underline{b}b, \underline{a}aZ_0)$

→ pop dir[illegible]

↓

$(q_1, \underline{b}, \underline{a}Z_0)$

↓

~~(q_1, ε, Z_0)~~ (q_1, ε, Z_0)

↓

(q_2, ε)

final State

Stack is Empty.

∴ $m = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$

$= [\{q_0, q_1, q_2\}, \{a, b\}, \{Z_0, a\}, \delta, q_0, Z_0, q_2]$

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

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∧B¯ $F=A\wedge\bar{B}$, is satisfiable, because A = TRUE and B = FALSE makes F = TRUE.
- G=A∧A¯ $G=A\wedge\bar{A}$, is unsatisfiable, because:

AA	A¯$\bar{A}$	GG
TRUE	FALSE	FALSE
FALSE	TRUE	FALSE

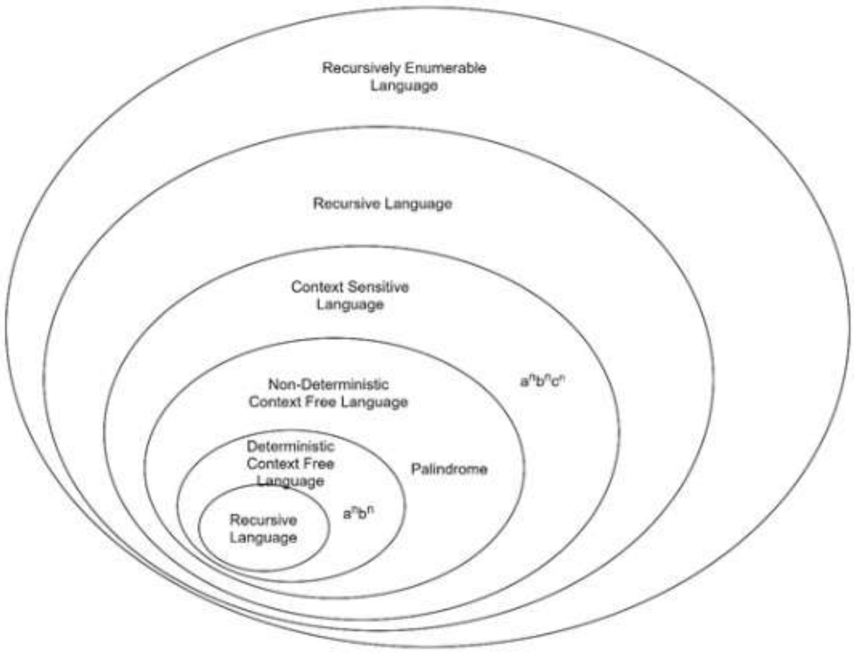
41. Illustrate the process of Recursive languages and Recursively enumerable Languages with suitable examples?

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 $L=\{a^n b^n c^n | n \geq 0\}$ L={anbncn|n≥0}. This language consists of strings where the number of a's, b's, and c's are equal

- **RE Language** – We can build a Turing machine that starts at the beginning of the string and systematically checks if the number of 'a's, 'b's, and 'c's are equal. If they are, it accepts the string. However, if the string is not of this form, the machine may never halt, potentially looping forever. This makes L a recursively enumerable language.
- **Recursive Language** – We can also construct a Turing machine that checks if the number of 'a's, 'b's, and 'c's are equal. If they are, it accepts the string. If they are not, it reaches a "reject" state and halts. This makes L a recursive language as well.

Closure Properties of Recursive Languages

Recursive languages possess an interesting property called closure. This means that certain operations performed on recursive languages result in another recursive language.

Here are some key closure properties –

Union

If L1 and L2 are two recursive languages, their union (L1 ∪ L2) is also recursive. Imagine a machine that checks if a string belongs to L1 or L2; if it does, it accepts. Since both machines for L1 and L2 will eventually halt, this combined machine will also halt, making the union recursive.

Concatenation

If L1 and L2 are two recursive languages, their concatenation (L1.L2) is also recursive. Imagine a machine that first checks if the first part of the string belongs to L1, and if it does, it checks the remaining part of the string for L2. Since both L1 and L2 machines halt, this combined machine will also halt.

Kleen Closure

If L1 is a recursive language, its Kleen closure (L1*) is also recursive. This means the language including all possible combinations of strings from L1 concatenated together, including the empty string, is also recursive.

Intersection

If L1 and L2 are two recursive languages, their intersection (L1 ∩ L2) is also recursive. Imagine a machine that checks if a string belongs to both L1 and L2. Since both L1 and L2 machines halt, this combined machine will also halt.

Complement

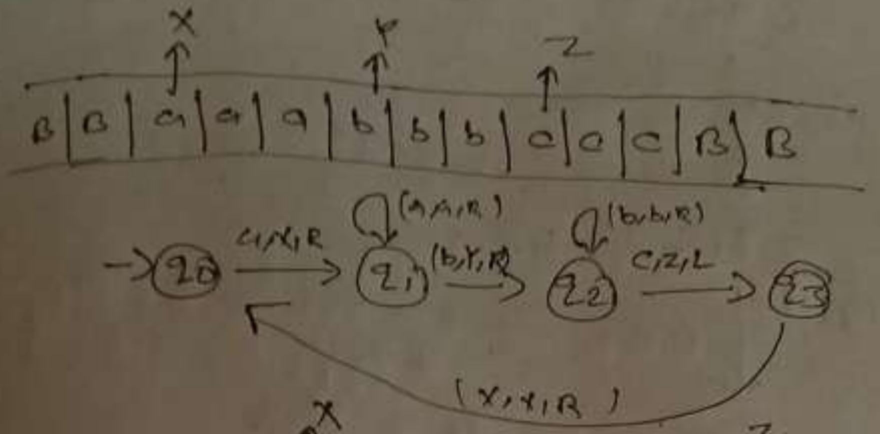
If L1 is a recursive language, its complement (L1') is also recursive. This means the language containing all strings *not* in L1 is also recursive.

42. Construct a Turing Machine

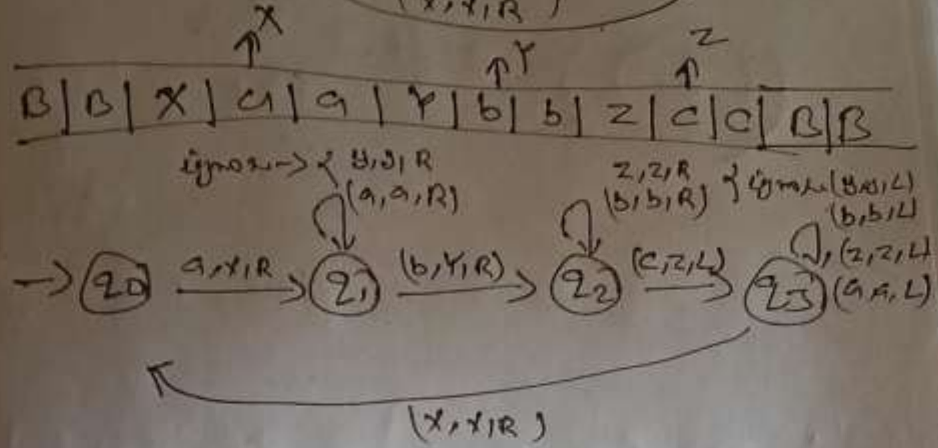
$L=\{ a^n b^n c^n \mid n \geq 1 \}$

$\{a^n b^n c^n \mid n >= 1\}$

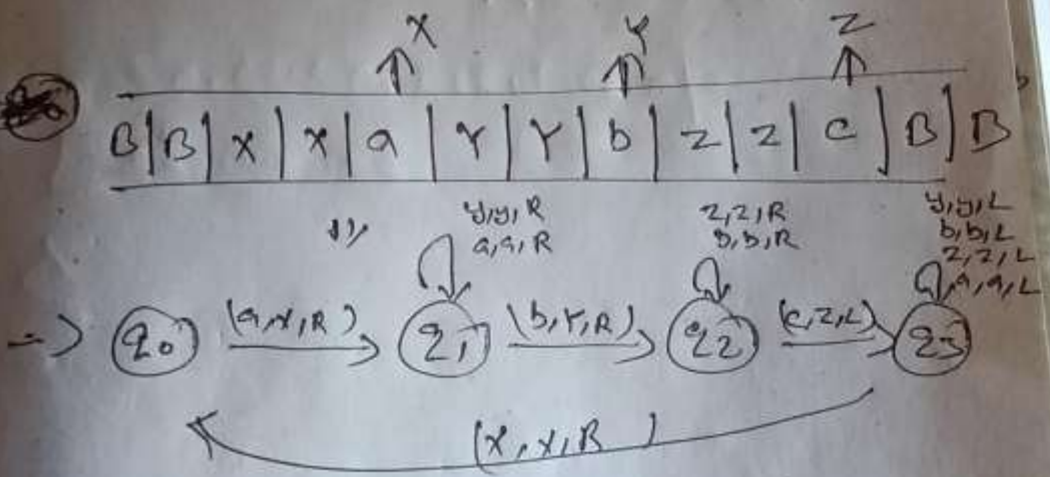
i) =>



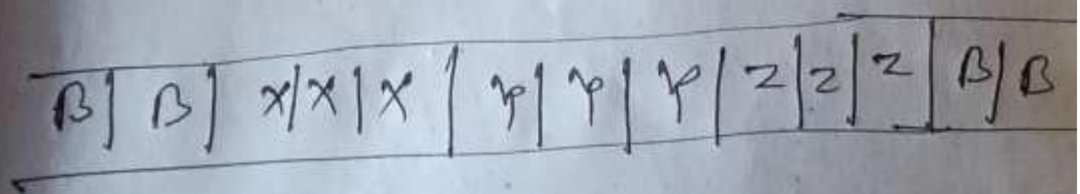
ii) =>

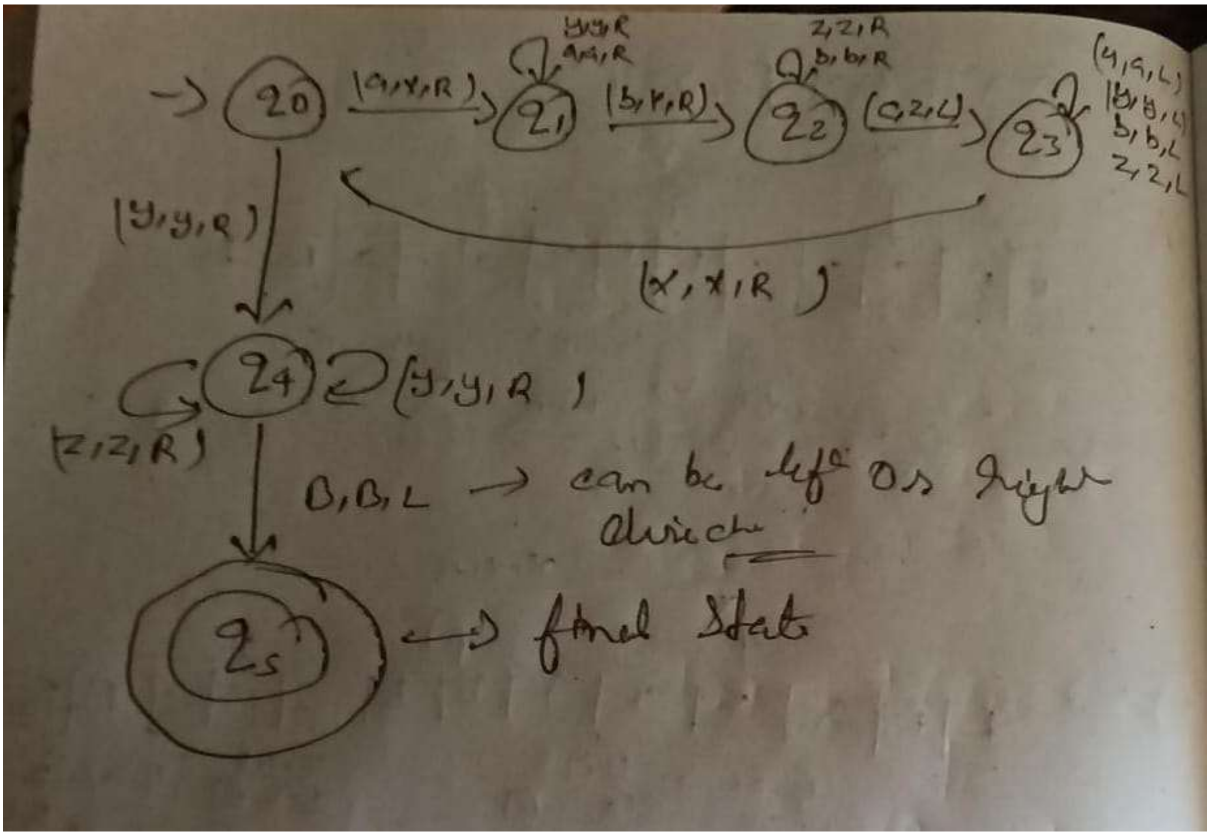


iii) =>



iv)




q0
(a,X,R)
q1
Y,Y,R
a,a,R
(b,Y,R)
q2
Z,Z,R
b,b,R
(c,Z,L)
q3
(a,a,L)
(Y,Y,L)
b,b,L
Z,Z,L
(X,X,R)
(Y,Y,R)
q4
(Y,Y,R)
(Z,Z,R)
B,B,L → can be left or right direction
q5 → final State