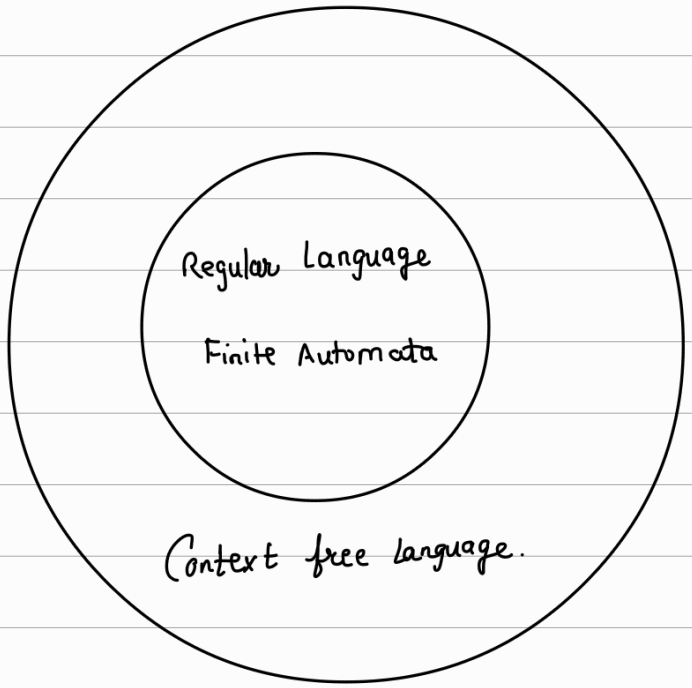
Context Free Language



Context Free Grammer ← Regular Expression

Push Down Automata.

- Context free Grammars are more expressive than finite automata.
- Language generated by context free Grammer is context free language.

⟶ gives

⟹ derives

Production is defining rule, always single line.

Derivation is deriving a rule,

Context free Grammer $G = (V, T, P, S)$ where

V is the finite set of variables,

T is the finite set of alphabets/ terminals

P is the finite set of productions

$S \in V$ is the start variable.

- Each production P is of the form $\underset{\uparrow}{A \to S}$ where $A \in V$ and $S \in (V \cup T)^*$
  (A gives S)

<sentence> ⟶ <noun_phrase> <predicate> ⟶ The Cat runs.

<noun_phase> ⟶ <article> <noun>

<predicate> ⟶ <verb>

<article> ⟶ the

<noun> ⟶ Cat

<verb> ⟶ runs

If you have productions:

$A \to \alpha_1$
$A \to \alpha_2$
$A \to \alpha_3$

you can write them as,

$A \to \alpha_1 \mid \alpha_2 \mid \alpha_3$

Design CFG for language $a^n b^n$ where $a \geq 0$.

$P \to S \to asb \mid \in$

$S \Rightarrow asb \mid \in$

$\Rightarrow aasbb$

$\Rightarrow a^2asbb^2$

$\Rightarrow a^3asbb^3$

$\cdots$

$\Rightarrow a^{n-1}asbb^{n-1}$

$\Rightarrow a^n sb^n$

↳ to remove S, we need s to become $\in$,

∴ we write

$S \Rightarrow asb \mid \in$

Design CFG means majorly designing productions

Final Answer:

$G = (\{S\}, \{a, b\}, P, S\}$

↑ Grammar

V is one step derivable from U denoted by $U \Rightarrow V$ (derived), if $U = x\alpha z$,
$V = x\beta z$
if $\alpha \rightarrow \beta$ gives

Productions in Grammar
$S \rightarrow AB$
$A \rightarrow aaA \mid \epsilon$
$B \rightarrow Bb \mid \epsilon$

$S \Rightarrow AB$
$\Rightarrow aaAB$
$\Rightarrow aa\epsilon B$
$\Rightarrow aaBb$
$\Rightarrow aa\epsilon b$
$\Rightarrow aab$

OR (Ma'am's way)

$\Rightarrow aaABb$
$\Rightarrow aa\epsilon Bb$
$\Rightarrow aa\epsilon Bb$
$\Rightarrow aa\epsilon\epsilon b$
$\Rightarrow aab$

In a step you have to apply only one production

Can we simultaneously remove ϵ and production B.

→ Yes

CFL:

given a Context Free Grammar $G = (V, T, P, S)$ the language generated from G is called context free Grammer.

- It is denoted from $L(G) = \{\omega \mid S \overset{*}{\Rightarrow} \omega\}$

CFL for strings of balanced parantheses {, (,), }

Terminals → {, (,), }

Production :

① $S \rightarrow SS \mid \in \mid (S) \mid \{S\}$

R - Rule

$S \Rightarrow SS$ ... R1

$\Rightarrow \{S\}\, S$ ... R4

$\Rightarrow \{(S)\}\, S$ ... R3

$\Rightarrow \{(\in)\}\, S$ ... R2

$\Rightarrow \{()\}\, S$ .... ∈ - removing

$\Rightarrow \{()\}\, (S)$ R3

$\Rightarrow \{()\}\, (\in)$ ... R2

$\Rightarrow \{()\}\, ()$ ... ∈ Removal

A language can have more than one grammar.

② Production :

$S \rightarrow (S)S \mid \{S\}S \mid \in$

$\{()\}\, ()$

$S \Rightarrow \{S\}S$

$\Rightarrow \{(S)\}S$

$\Rightarrow \{(\in)\}S$

$\Rightarrow \{()\}\,(S)S$

$\Rightarrow \{()\}\,(\in)S$

$\Rightarrow \{()\}\,()\in$

$\Rightarrow \{()\}\,()$

Design grammer for ww^R where $w \in \{a,b\}^*$

$S \rightarrow asa \mid bsb \mid \in \mid a \mid b$

abba

$S \Rightarrow asa$

$S \Rightarrow absba$

$S \Rightarrow ab \in ba$

$S \Rightarrow abba$.

A parse tree of a derivation is a tree in which each internal node is labelled with a variable.

If you have rule,

$A \rightarrow A_1\ A_2 \ldots A_n$

then you can draw parse tree

Derivation or yield is the final string obtained by concating labels of the leaves of tree from left to right ignoring ϵ

Suppose, you have rule:

$E \rightarrow E+E \mid a$

$E \Rightarrow E+E$

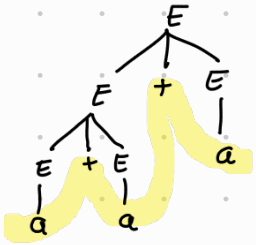
$\Rightarrow E+E+E$

$\Rightarrow E+E+a$

$\Rightarrow E+a+a$

$\Rightarrow a+a+a$

Parse Tree ⟹



→ Yield of parse tree.

Q. $S \rightarrow AB$

$A \rightarrow aaA \mid \epsilon$

$B \rightarrow Bb \mid \epsilon$

Find parse tree for aab

$S \Rightarrow AB$

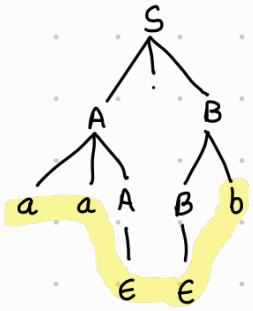
$\Rightarrow aaAB$

$\Rightarrow aaABb$

$\Rightarrow aa\epsilon Bb$

$\Rightarrow aa\epsilon\epsilon b$

$\Rightarrow aab$



Yield = $aa\epsilon\epsilon b$

but we ignore ϵ in yield,

∴ Yield = aab

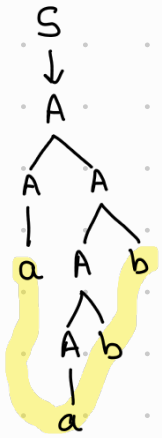
Q. $S \rightarrow A \mid AB$

$A \rightarrow \in \mid Ab \mid AA \mid a$

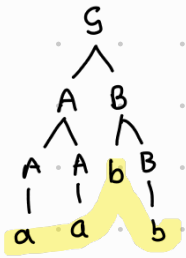
$B \rightarrow b \mid bc \mid Bc \mid bB$

for string aabb, can you draw more than one parse tree?

① $S \Rightarrow A$
$\Rightarrow AA$
$\Rightarrow AAb$
$\Rightarrow AAbb$
$\Rightarrow Aabb$
$\Rightarrow aabb$



② $S \Rightarrow AB$
$\Rightarrow AAB$
$\Rightarrow AABB$
$\Rightarrow aABB$
$\Rightarrow aaBB$
$\Rightarrow aabB$
$\Rightarrow aabb$



Thus we can have multiple parse trees, these multiple parse trees cause the compiler to get confused. This is Ambiguous Grammar.

A partial derivation tree is a subtree of a parse tree such that either all of its children in subtree of none in the subtree.

- If a partial derivation tree contains a variable is called a sentential form.
- Left most derivation or LMD of a sedential form is one in which productions are applied always to the left most variables.

RMD - Replace Right Most

LMD

$S \Rightarrow \underline{A}B$
$\Rightarrow AAB$
$\Rightarrow aAB$
$\Rightarrow aaB$
$\Rightarrow aabB$
$\Rightarrow aabb$

$S \longrightarrow A \mid AB$
$A \rightarrow \in \mid Ab \mid AA \mid a$
$B \rightarrow b \mid bc \mid Bc \mid bB$

LMD

② $S \Rightarrow A$
$\Rightarrow AB$
$\Rightarrow AAB$
$\Rightarrow aAB$
$\Rightarrow aaB$
$\Rightarrow aabB$
$\Rightarrow aabb$

RMD:

$S \Rightarrow AB$
$\Rightarrow Ab$
$\Rightarrow Abb$
$\Rightarrow AAbb$
$\Rightarrow Aabb$
$\Rightarrow aabb$

aabb

$S \longrightarrow A \mid AB$
$A \rightarrow \in \mid Ab \mid AA \mid a$
$B \rightarrow b \mid bc \mid Bc \mid bB$

Ambiguous Grammar:

A Grammer G is ambiguous if a string belonging to grammar has at least two different:

1) Parse Trees
2) LMDs
3) RMDs

E.g. $S \rightarrow AS \mid \epsilon$

$A \rightarrow A1 \mid 0A1 \mid 01$

String : 0011

To prove grammar is ambiguous, take a string derive parse string.

① $S \Rightarrow AS$

$\Rightarrow 0A1S$

$\Rightarrow 0011S$

$\Rightarrow 0011\epsilon$

$\Rightarrow 0011\epsilon$

② $S \Rightarrow AS$

$\Rightarrow$

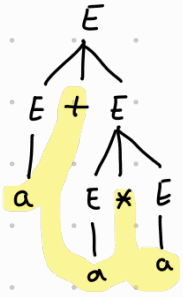
$=$

a) $E \rightarrow E+E \mid E*E \mid (E) \mid a$

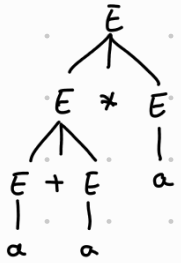
$\Sigma = \{a, +, *\}$

Show that language is ambiguous

$a+a*a$



$E \Rightarrow E+E$
$\Rightarrow a+E$
$\Rightarrow a+E*E$
$\Rightarrow a+a*E$
$\Rightarrow a+a*a$



$E \Rightarrow E*E$
$\Rightarrow E+E*E$
$\Rightarrow a+E*E$
$\Rightarrow a+a*E$
$\Rightarrow a+a*a$

- Operator Precedence and left associativity:

$F \rightarrow (E) \mid a$

$T \rightarrow T*F \mid F$

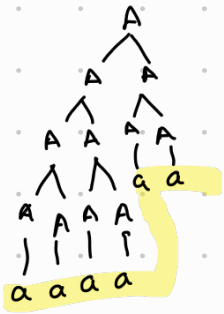
$E \rightarrow E+T \mid T$

$E \Rightarrow E+T$
$\Rightarrow E+T*F$
$\Rightarrow T+F*F$
$\Rightarrow F+F*F$
$\Rightarrow a+a*a$

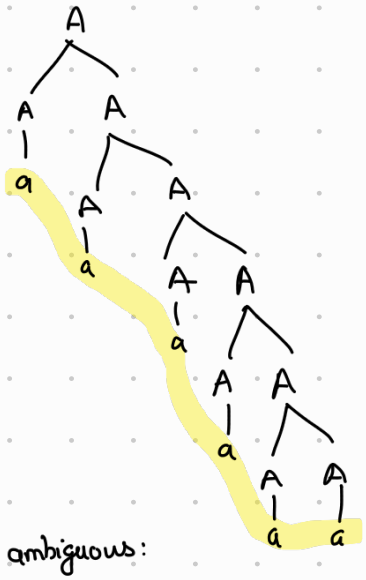
Q) $A \rightarrow AA \mid a$

String = aaaaaa

$A \Rightarrow AA$
$\Rightarrow AAA$
$\Rightarrow AAAA$
$\Rightarrow AAAAA$
$\Rightarrow AAAAAA$
$\Rightarrow aAAAAA$
$\Rightarrow aaAAAA$
$\Rightarrow aaaAAA$
$\Rightarrow aaaaAA$
$\Rightarrow aaaaaA$
$\Rightarrow aaaaaa$



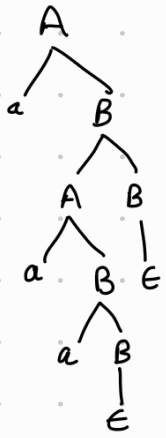
$A \Rightarrow AA$
$\Rightarrow AAA$
$\Rightarrow AAAA$
$\Rightarrow AAAAA$
$\Rightarrow AAAAAA$
$\Rightarrow aAAAAA$
$\Rightarrow aaAAAA$
$\Rightarrow aaaAAA$
$\Rightarrow aaaaAA$
$\Rightarrow aaaaaA$
$\Rightarrow aaaaaa$



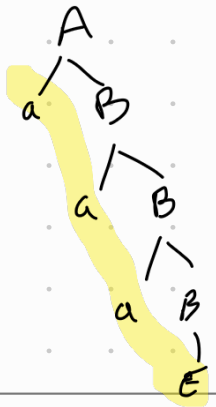
To make above language non ambiguous:

for string = aaa

$A \rightarrow aB$
$B \rightarrow AB \mid \epsilon$



$A \rightarrow aB$
$B \rightarrow aB \mid \epsilon$



Search up! → Right Recursion Removal | Left factoring

$A \longrightarrow \alpha B_1 \mid \alpha B_2$

left factoring

$A \rightarrow \alpha B$

$B \rightarrow B_1 \mid B_2$

Inherently Ambiguous Grammar:

Let 'L' be a context free language, if every Context free Grammer 'G' with language 'L' is ambiguous the 'L' is said to be inherently ambiguous.

$G = \{0^i 1^j 2^k \mid i=j \text{ or } j=k$

G_1 :

$S \rightarrow AB \mid CD$

$A \rightarrow 0A1 \mid 01$

$B \rightarrow 2B \mid 2$

$C \rightarrow 0C \mid 0$

$D \rightarrow 1D2 \mid 12$

V, T, P, S.

$\in \notin L$

① Useless Variables
② Unit Production
③ $A \to \in$

① Useless Symbol:

- A symbol 'x' is useful for a grammar $G = (V, T, P, S)$ if there is

$$S \overset{*}{\Rightarrow} \alpha X \beta \overset{*}{\Rightarrow} \omega \in T^*$$

if variable is not involved in any derivation of a language.

$S \to aSb \mid \in \mid A$

$A \to aA$ → here A is useless as once we go into A, there is no termination of string.

→ A Symbol 'x' is useful if
1. x is a generating variable.
2. x is reachable

} make sure to apply these rules in their order.

$S \to AB \mid a$
$A \to a$

Generating Variables: {S, A}
Non Generating {B}

① ∴ $S \to AB \mid a$
will become,

$S \to a$ ∵ AB has B which is non generating.
$A \to a$

② → $S \to a$ ∵ A is unreachable.

Procedure for eliminating non generating symbols:

1) Every symbol of T is generating.
2) If $a \to \alpha$ and α is generating, then a is generating.

} Calculate Generating Variables here.

$$\underset{NGV}{\text{Non Generating Variables}} = \text{Variables} - \underset{GV}{\text{Generating Variables}}$$

Q. $S \to aS \mid A \mid C$
$A \to a$
$B \to aa$
$C \to acb$

① $G.V. = \{S, A, B\}$
$NGV = \{C\}$

② $S \to aS \mid A$
$A \to a$
$B \to aa$

step 2: For production of form $C \to xDy$ create a dependency graph

(C) → (D)

Draw dependency graph for all production, if there is an edge not reaching variable x starting from start variable S, then x is not reachable.

(S) → (A) (B)

$\therefore$ $S \to aS \mid A$
$A \to a$ is the grammar.

$T \rightarrow aaB \mid abA \mid aaT$

$A \rightarrow aA$

$B \rightarrow ab \mid b$

$C \rightarrow ad$

Step 1:

① Non Generating symbols = $\{A\}$

Generating Symbols = $\{B, C\}$

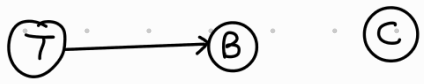
Production 1:

$T \rightarrow aaB \mid aaT$

$B \rightarrow ab \mid b$

$C \rightarrow ad$

Step ② Dependency Graph



Production 2:

$T \rightarrow aaB \mid aaT$

$B \rightarrow ab \mid b$

Production of type

'$S \rightarrow \epsilon$' are called ϵ production.

IF A in multiple derivations reaches ϵ, then A is nullable variable.

↳ $A \overset{*}{\Rightarrow} \epsilon$

① Find out nullable variables

② Add productions with nullable variables removed.

③ Remove ϵ productions and duplicates.

Algorithm to find nullable variable:

- $V \leftarrow$ set of variables.
- $N_0 \leftarrow \{A \mid A \rightarrow \epsilon$ in production and $A \in V\}$

Repeat

$N_i \leftarrow N_{i-1} \cup \{A \mid A \in V,\ A \rightarrow \alpha,\ \alpha$ is in $N_i\}$

until $N_i = N_{i-1}$

Q. $S \rightarrow XYX$

$X \rightarrow 0X \mid \epsilon$

$Y \rightarrow 1Y \mid \epsilon$

$NV = \{X, Y, S\}$

$S \rightarrow XYX$

$\rightarrow XYX \mid YX \mid XY \mid Y \mid XX \mid X$

$X \rightarrow 0X \mid 0$

$Y \rightarrow 1Y \mid 1$

$S \rightarrow AB$

$A \rightarrow aAA \mid \epsilon$

$B \rightarrow bBB \mid \epsilon$

$NV = \{S, A, B\}$

$S \rightarrow AB \mid A \mid B$

$A \rightarrow aAA \mid aA \mid a$

$B \rightarrow bBB \mid bB \mid b$

Grammar w/o epsilon production and useless symbols then follow sequence

1. Eliminate ϵ-production
2. Eliminate useless symbols.

Q)

$S \rightarrow a \mid aA \mid b \mid c$

$A \rightarrow aB \mid \epsilon$

$B \rightarrow aA$

$C \rightarrow aCD$

$D \rightarrow ddd$

Nullable variables $= \{A\}$

Production 1

P_1

$S \rightarrow a \mid aA \mid B \mid c$

$A \rightarrow aB$

$B \rightarrow aA \mid a$

$C \rightarrow aCD$

$D \rightarrow ddd$

Step 2: Eliminate useless symbols by finding generating variables.

Non-generating Variable $\Rightarrow \{C\}$

Generating Variable $\Rightarrow \{S, A, B, D\}$

P_2: Remove Non generating

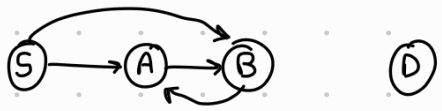
$S \rightarrow a \mid aA \mid B$

$A \rightarrow aB$

$B \rightarrow aA \mid a$

$D \rightarrow ddd$

Reachability



P_3

$S \rightarrow a \mid aA \mid B$

$A \rightarrow aB$

$B \rightarrow aA \mid a$

Productions of the form $A \rightarrow B$ where A and B are variables are known as unit productions:

Procedure to remove Unit

1. Add all non unit productions of P to P_1
2. For each unit production of form $A \rightarrow B$ add to P_1, $A \rightarrow \alpha$ if $B \rightarrow \alpha$ is a non-unit production in P.
3. Delete all unit productions.

If a grammar has $\in$ productions, useless symbols and unit productions the sequence of removal is

1. Remove $\in$ productions and get G_1
2. Remove unit productions from G_1 and get G_2
3. Remove useless from G_2 and get G_3

Q) $S \rightarrow 0A \mid 1B \mid C$

$A \rightarrow 0S \mid 00$

$B \rightarrow 1 \mid A$

$C \rightarrow 01$

P_1:

$S \rightarrow 0A \mid 1B \mid 01$

$A \rightarrow 0S \mid 00$

$B \rightarrow 1 \mid A$

Q. Simplify Grammar:

$S \rightarrow a \mid aA \mid B \mid C$

$A \rightarrow aB \mid \epsilon$

$B \rightarrow aA$

$C \rightarrow ccD$

$D \rightarrow ddd$

1. Remove Nullable variables: $\{A\}$

P_1:

$S \rightarrow a \mid aA \mid B \mid C$

$A \rightarrow aB$

$B \rightarrow aA \mid a$

$C \rightarrow cCD$

$D \rightarrow ddd$

2. Remove unit productions:

$S \rightarrow a \mid aA \mid \underbrace{\cancel{aA \mid a}}_{\text{already there}} \mid cCD$

$A \rightarrow aB$

$B \rightarrow aA \mid a$

$C \rightarrow cCD$

$D \rightarrow ddd$

3. Remove useless symbols:

Generating Variables = { A, B, D }

NGV = { C }

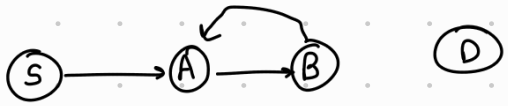
$S \rightarrow a \mid aA$

$A \rightarrow aB$

$B \rightarrow aA \mid a$

$D \rightarrow ddd$

4. Reachability



P_4 :

$S \rightarrow a \mid aA$

$A \rightarrow aB$

$B \rightarrow aA \mid a$

$D \rightarrow ddd$

Chomsky Normal Form:

Every non empty CFL without $\in$ has a grammar with productions in CNF, if all productions are of the form,

$S \rightarrow AB$

$A \rightarrow \alpha$

$S, A, B \in V$

$\alpha \in T^+$

Step 1: Simplify grammar by eliminating useless symbols, $\in$-production and unit production.

2: Sim - ll- RHS of production

A. Add all productions of the form $A \rightarrow BC$ / $A \rightarrow a$

B. Consider a production $A \rightarrow X_1 X_2 \ldots X_n$

e.g. if X_i is a terminal 'a' then you add a production CAI gives A and

replace X_i with CA_i in the A production.

e.g. $A \to aBC$

new variable 'D' is to be introduced,

$D \to a$

rewrite A as $A \to DBC$.

Step 3: Consider a production $A \to X_1 X_2 \ldots X_n$ where $n \geq 3$, and all X_i's are variables. Introduce new productions

$A \to X_1 C_1$

$C_1 \to X_2 C_2$

$C_{n-2} \to X_{n-1} X_n$

Suppose: $A \to BCDE$

$C_1 \to BC$

$C_2 \to DE$

$A \to C_1 C_2$

} Wrong Approach

Do this ⇒

$A \to BC_1$

$C_1 \to CC_2$

$C_2 \to DE$

e.g.

$S \to aAD$

$A \to aB \mid bAB$

$B \to b$

$D \to d$

P_i:

$B \to b$

$D \to d$

$C_a \to a$

$S \to C_a AD$

⇓

$S \to C_a C_1$

$C_1 \to AD$

$A \to C_a B$

$A \rightarrow bAB$

$A \rightarrow BAB$

$\Downarrow$

$A \rightarrow BC_2$

$C_2 \rightarrow AB$

Grammar: (

$\{ A, B, C_a, C_1, C_2, D, S \}$,

$\{ a, b, d \}$,

P_1,

S)

1) To reach a sendential form of length 'n' you require 'n-1' steps. To reach a string of terminals of length 'n' you need 2n-1 derivation.

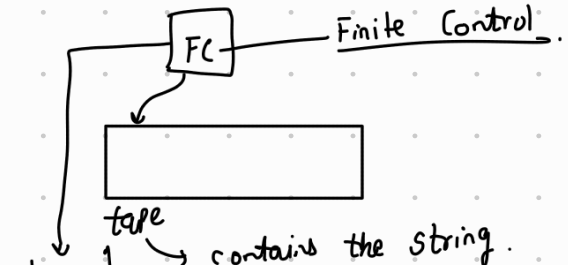
2) Grammar in CNF is not ambiguous.

Push Down Automata (PDA)

There are 2 types of PDA

1) NPDA - Non-deterministic PDA
2) DPDA - Deterministic PDA

· By default all are NPDA.



An NPDA is a 7 tuple, $(Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$

Γ → stack alphabet

Z_0 → Start Stack Symbol

$L = a^n b^n \mid n \geq 0$

$\delta = Q \times ((\Sigma \cup \{\epsilon\}) \times \Gamma) \rightarrow Q \times \Gamma^*$

Moves:
① Consuming Input
② Episilon Move.

Configuration:

$(p, a, t) \rightarrow (q, u)$

Cases:

1) Skip operation.

$u = t$

· Consume a terminal but without changing the stack.

$(p, a, t) = (q, u)$

2) Pop operation:

$(p, a, t) = (q, \epsilon)$

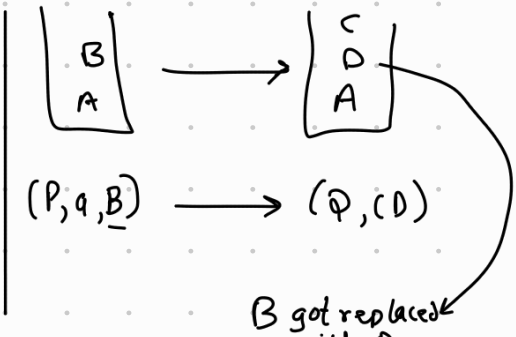
· Consuming input and popping.

3) Push operation

if $u = wx$,

$(p, a, t) \rightarrow (q, wx)$

· t is replaced with x, w is pushed into stack.



Q. $\{a^n b^n \mid n \geq 0\}$

$(q_0, a, Z_0) \longrightarrow (q_0, AZ_0)$

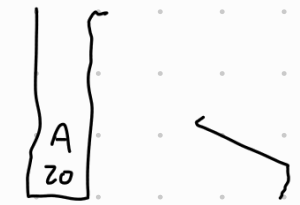
$(q_0, a, A) \longrightarrow (q_0, AA)$

$(q_0, b, A) \longrightarrow (q_1, \epsilon)$

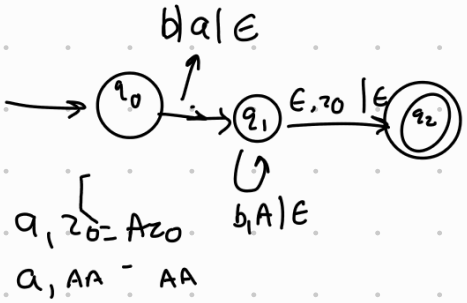
$(q_1, b, A) \longrightarrow (q_1, \epsilon)$

$(q_1, \epsilon, Z_0) \longrightarrow (q_2, \epsilon)$

PDA M has 3 states $\{q_0, q_1, q_2\}$



Any other moves → PDA stops.



Instantaneous Description (ID): or (Configuration):

- Describes execution status of a PDA, anytime.
- Represent by δ-

q- current state of PDA
ω: Unread part of the input string.
α- Stack content written as a string with left most symbol at top of stack.

* Id like $\hat{\delta}$ but for PDA.

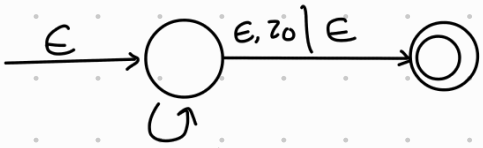
Q) Show moves of PDA for input aabb

Input string aabb
$(aabb, z_0) \vdash$
↑
Represents move of the PDA.

$(q_0, aabb, z_0) \vdash (q_0, abb, Az_0)$
$\vdash (q_0, bb, AAz_0)$
$\vdash (q_1, b, Az_0)$
$\vdash (q_1, \epsilon, z_0)$
$\vdash (q_1, \epsilon, z_0)$
$\vdash (q_2, \epsilon, \epsilon)$

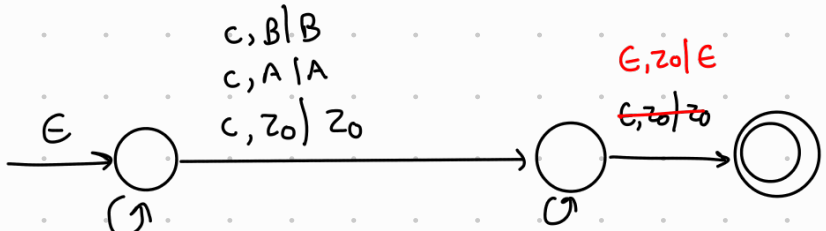
Q) $L = \{\omega \in \{a,b\}^* \mid |\omega|_a = |\omega|_b\}$

$L = \{\epsilon, ab, ba, aabb, abab, baba \ldots\ldots\}$



a, z0 | Az0
b, z0 | Bz0
a, A | AA
b, B | BB
a, B | ε
b, A | ε

Q) $L = \{\omega \in \{a,b\}^* \mid \omega c \omega^R\}$



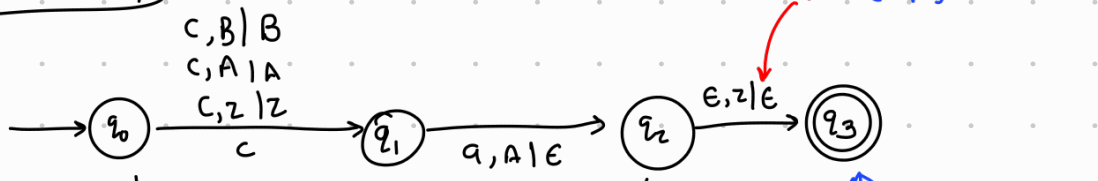
a, z0 | Az0
a, A | AA
b, z0 | Bz0
b, B | BB
a, B | AB
b, A | BA

it may happen that you get a on top of b

a, z0 | ε
a, A | ε
b, z0 | ε
b, B | ε

Not acceptable

Ma'am's Soln:



a, z | Az
a, A | AA
a, B | AB
b, z | Bz
b, A | BA
b, B | BB

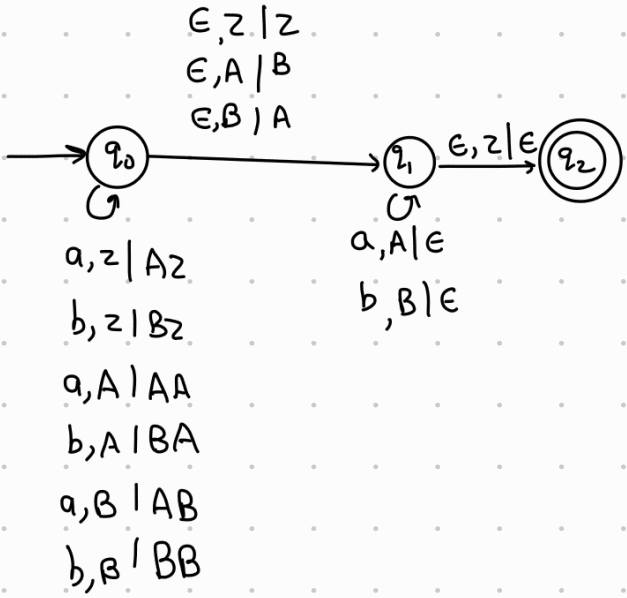
b, B | ε
a, A | ε

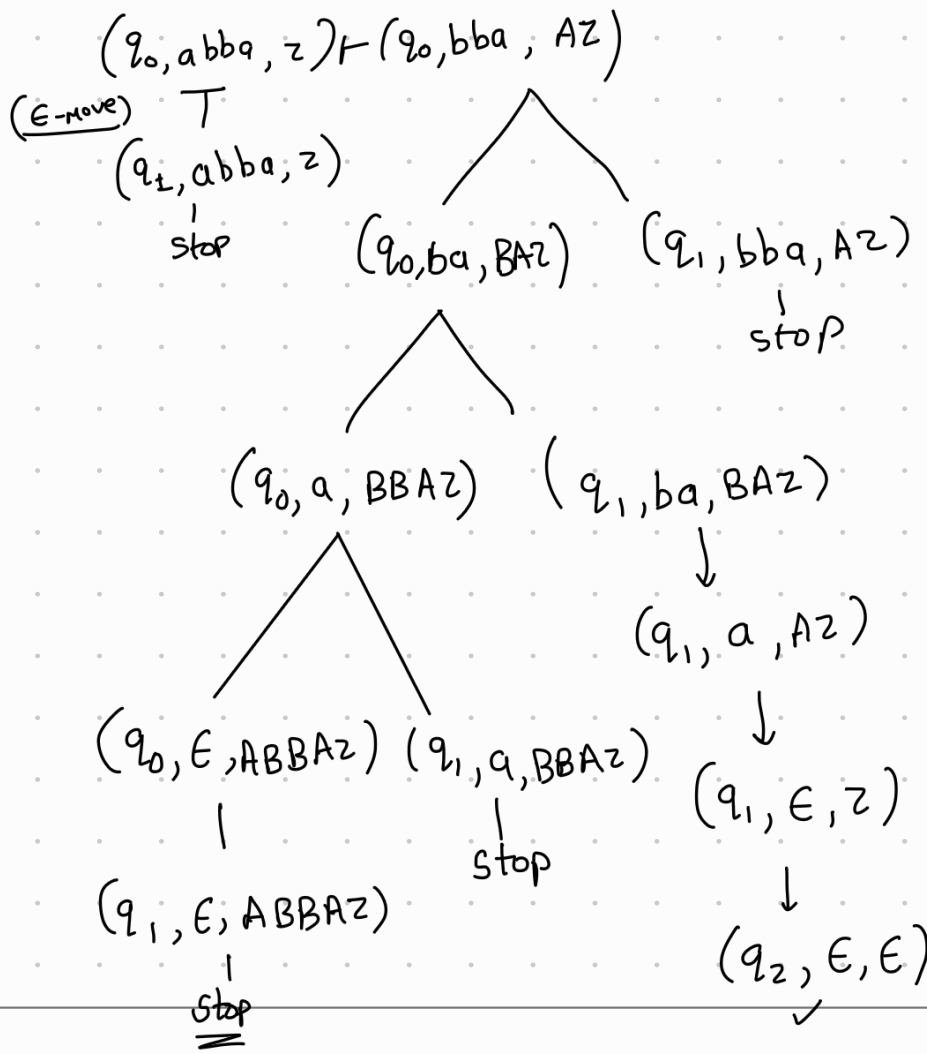
$L = \{ ww^R \mid w \in (a+b)^* \}$

Non Determinism → we need ϵ move.

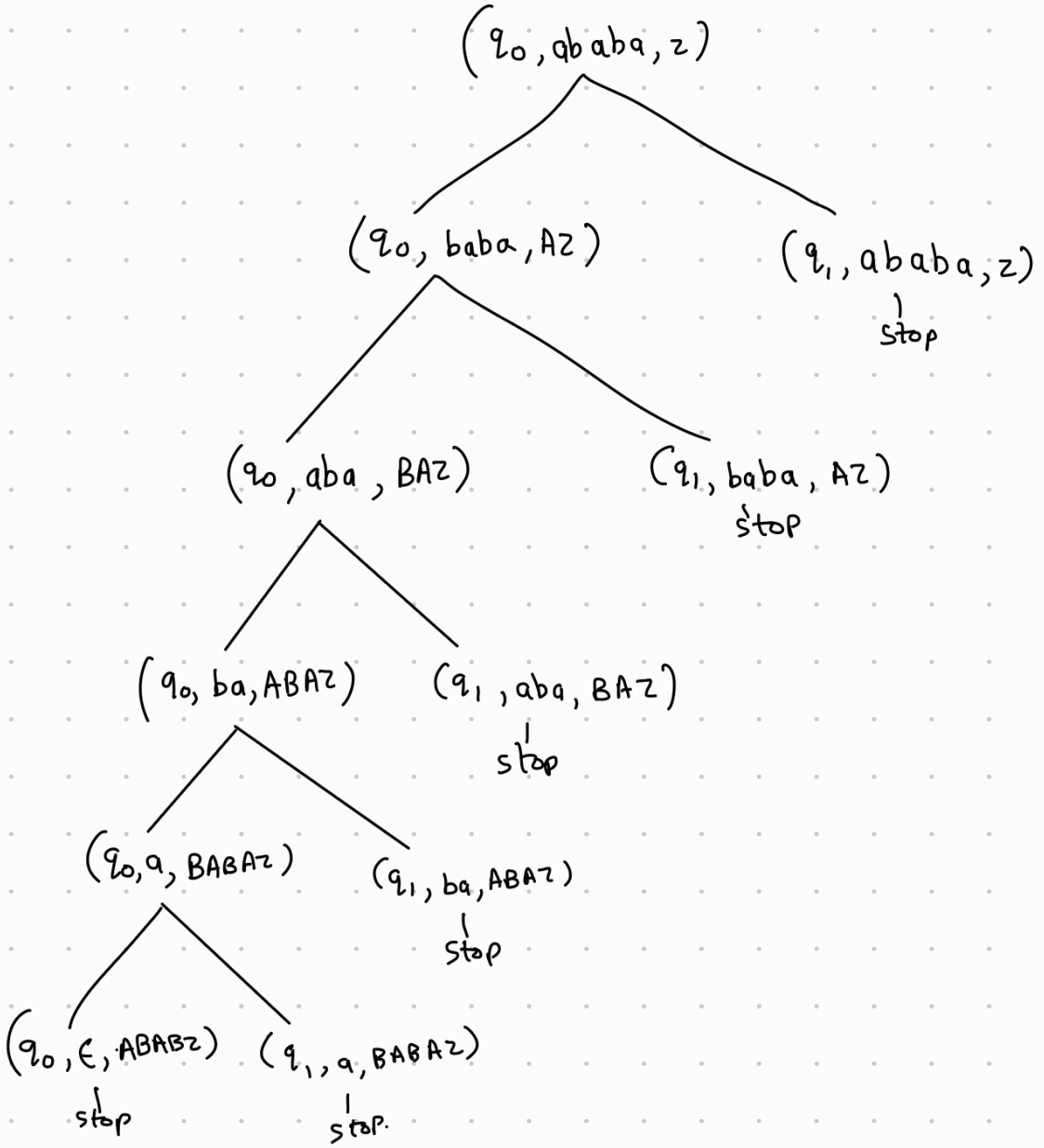
odd palindrome — Even palindrome

└ two logics



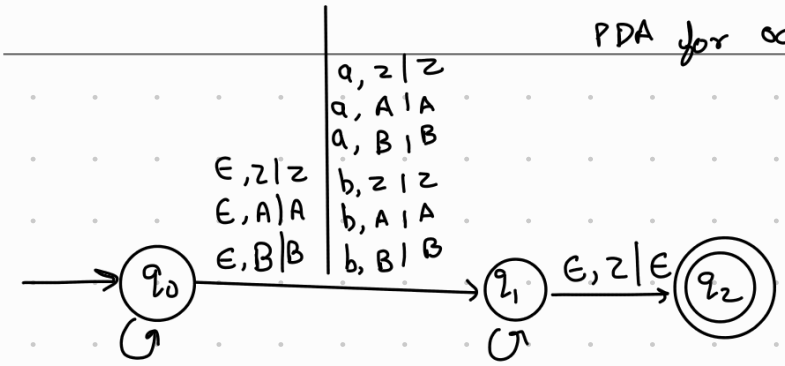


$(q_0, ababa, z) \vdash$



Won't work for odd Palindrome.

PDA for odd palindrome.



a, z | Az

b, z | Bz

a, A | AA

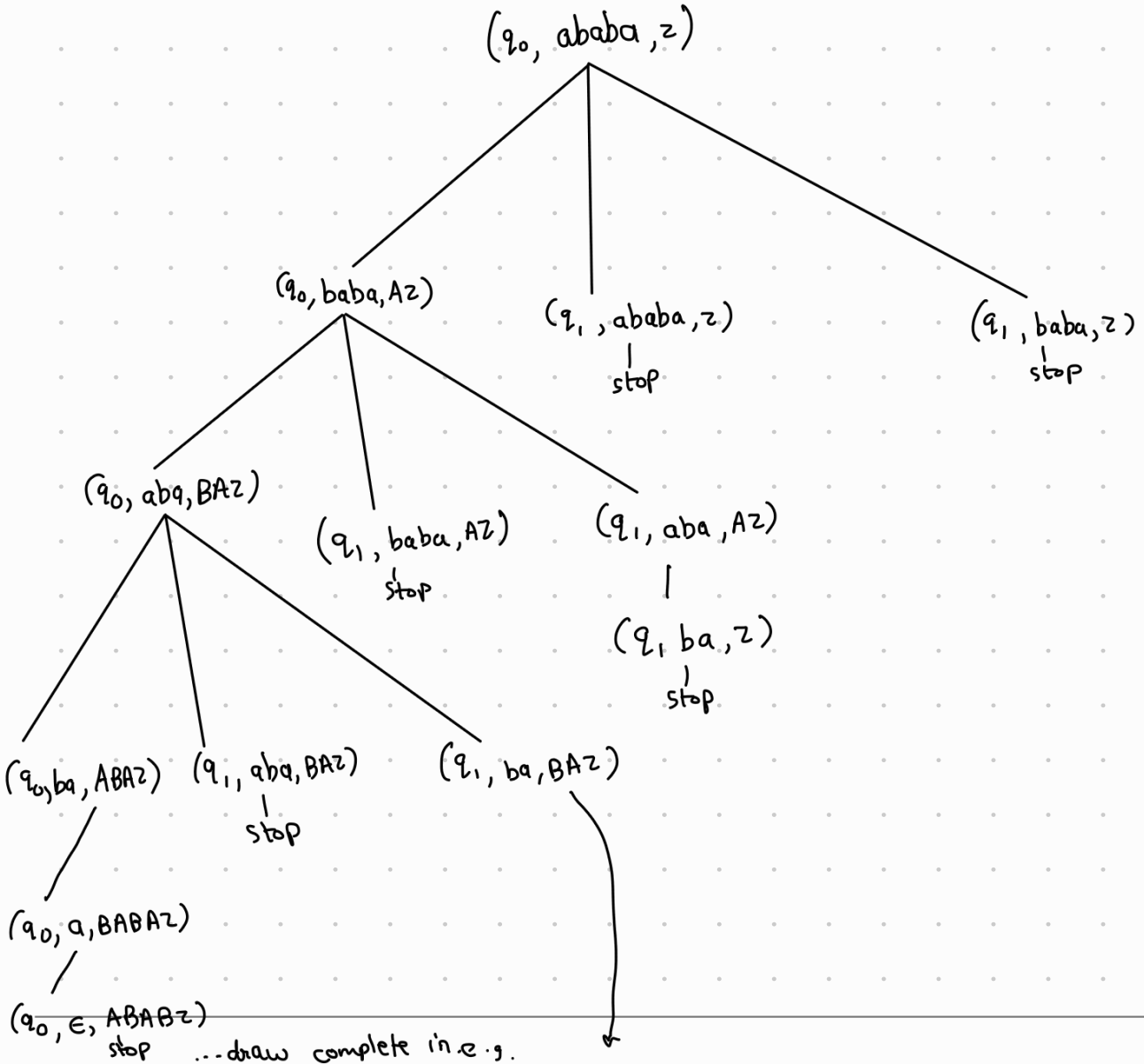
b, A | BA

a, B | AB

b, B | BB

a, A | ϵ

b, B | ϵ



...draw complete i.e.g.

(q_1, ba, BAz)

(q_1, a, Az)

(q_1, ϵ, z)

$(q_2, \epsilon, \epsilon)$ accepted.

Languages Accepted by a PDA:

① Acceptance by final State

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

$L(P) = \{ w \mid (q_0, w, z_0) \vdash^* (q, \epsilon, \alpha)$

$q \in F$

$\alpha \in \Gamma^*$

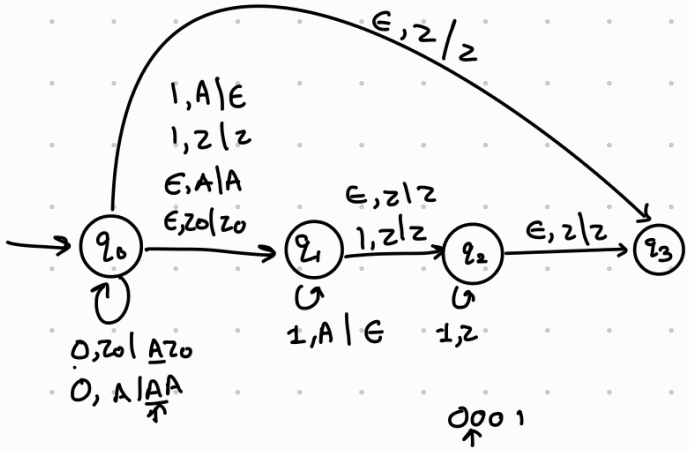
② Acceptance by empty stack → Do not mark final state

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

$N(P) = \{ w \mid (q_0, w, z_0) \} \vdash (q, \epsilon, \epsilon)$

where $q \in Q$

$L = \{ 0^i 1^j \mid 0 \le i \le j \}$



Acceptance by Empty Stack.

IF $L = N(P_N)$ for a PDA, $P_N = (Q, \Sigma, \Gamma, \delta_n, q_0, z_0)$ then there is a PDA P_F such that $L = L(P_F)$.

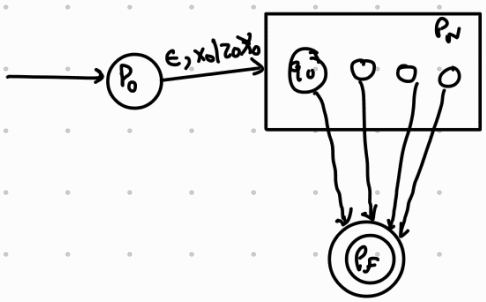
$P_N = (Q, \Sigma, \Gamma, \delta_N, q_0, z_0)$

$P_F = (\{Q \cup \{p_0, p_f\}\}, \Sigma, \Gamma \cup \{X_0\}, \delta_F, p_0, X_0, \{p_f\})$

$\delta_F(p_0, \epsilon, X_0) = \{(q_0, z_0, X_0)\}$

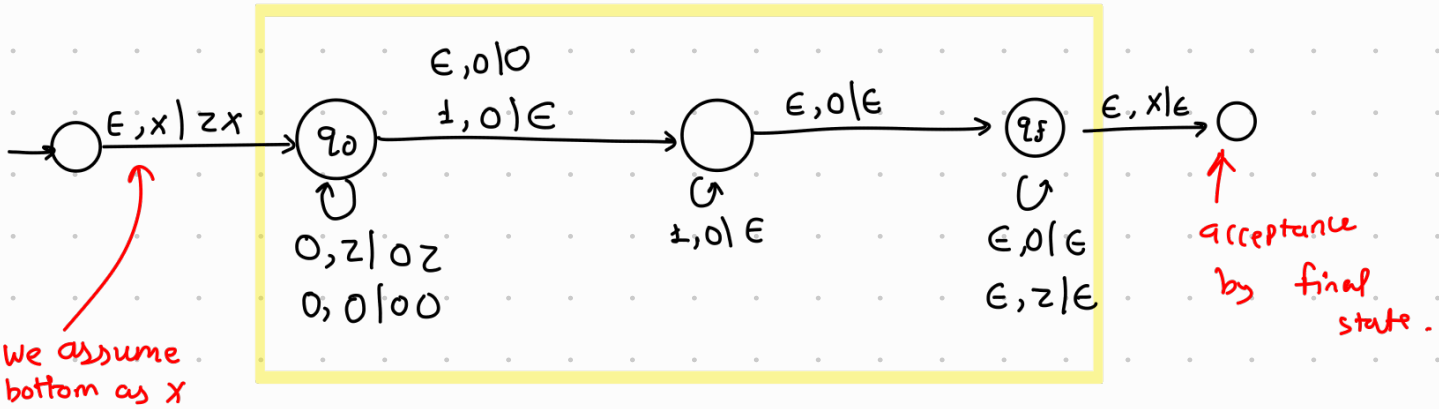
$\delta_F(q, a, y) = \delta_N(q, a, y)$ $a \in \Sigma$ or $a \in \epsilon$ and $y \in \Gamma$

$\delta_F(q_1, \epsilon, X_0) = \{(p_f, \epsilon)\}$

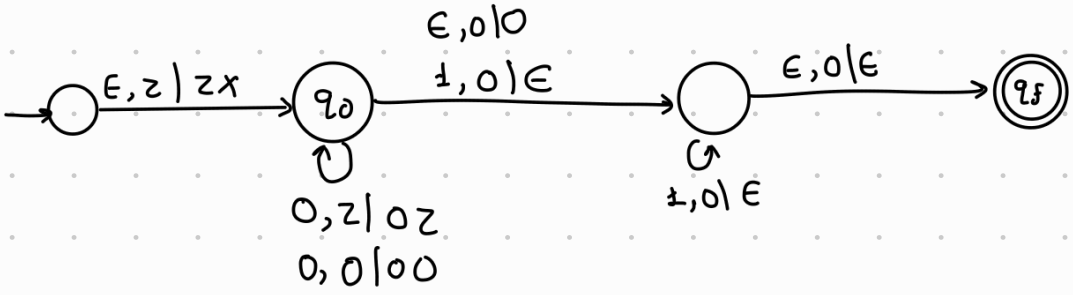


$L = \{0^n 1^m \mid m < n\}$

acceptance by final state



Acceptance by empty state.



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

1. $\delta(q, a, X)$, $q \in Q$, $a \in \Sigma$, and $X \in \Gamma$

2. if (q, a, X) is not empty $a \in \Sigma$, $\delta(q, \epsilon, x)$ must be empty.

→ i.e. $\delta(q, a, x)$ has at most 1 member $q \in Q$, $a \in \epsilon$ and $X \in \Gamma$

→ 2. If (q, a, x) is not embty for some $a \in \Sigma$, $\delta(q, \epsilon, X)$ must be empty.

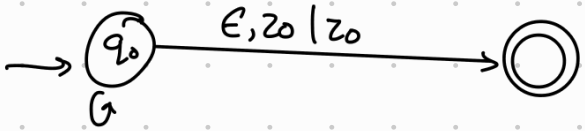
DPDA

WCW^R is a DPDA. how to identify. Search up.

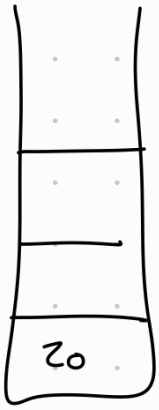
Design a PDA accepting strings with number of a's more than b's.
i.e. $a^i b^j \mid i > j$

Q.) design PDA to accept balanced strings. { }

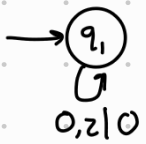
{ { } { { } }



{, z0 | A z0
{, A | AA
}, A | ϵ



$L = \{a^{2n} b^{3n} \mid n \geq 1\}$



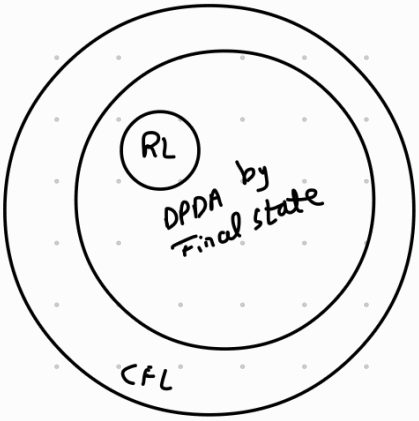
Deterministic CFL:

Language accepted by a DPDA.

Mode of acceptance for a DPDA to recognize Reg. Lang.

① If L is a regular language, we can design DPDA to accept by final state

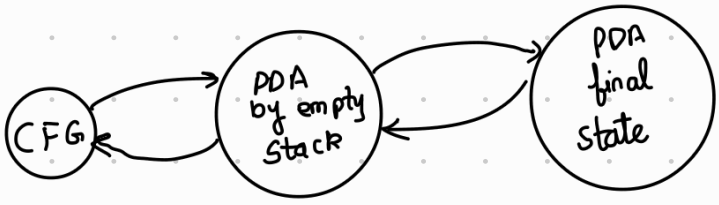
- PDA includes the stack but it is always inactive.



② For Reg. Lang to accept by empty stack, the regular language should have prefix property

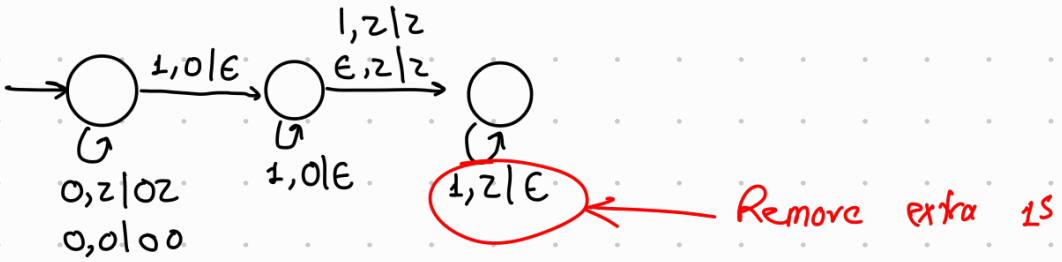
Prefix Property of Language L:

If x and y ∈ L, then x should not be prefix of y.



$0^n 1^m \mid n \leq m$

aabbb n=1

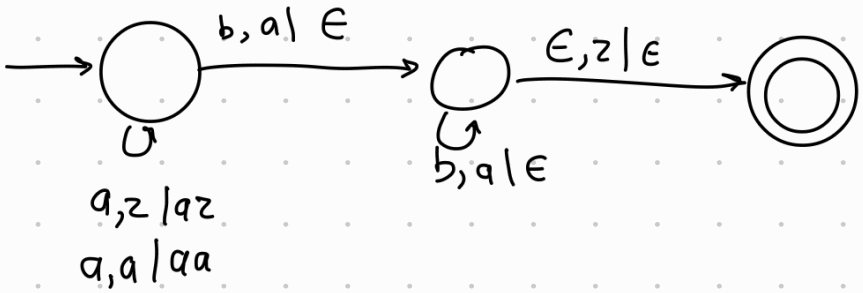


DPDA accepts unambiguous grammars

IF L is Language accepted by

IF L has unambiguous grammar there is a DPDA for L.

Ex. $S \rightarrow 0S0 \mid 1S1 \mid \epsilon$ is ambiguous.



9) Create unambiguous CFG, convert to lang with prefix property

Create an unambiguous CFG, you have to convert the language to have prefix property

Divide 01010 by LMD

$S \to 0S1 \mid A$

$A \mid 1A0 \mid S \mid \epsilon$

$S \Rightarrow 0S1$

$\Rightarrow 01A01$

$\Rightarrow 01S01$

$\Rightarrow 010S101$

$\Rightarrow 010A101$

$\Rightarrow 010101$

$\Rightarrow (q, 010101, S) \vdash (q, 010101, 0S1) \vdash (q, 10101, S1)$

$\vdash (q, 10101, A1) \vdash (q, 0101, 1A01)$

$\vdash^* (q, 01, 01S101) \vdash (q, 01, S101)$

$\vdash (q, \epsilon, \epsilon)$

PDA simulates the LMD from CFG.

If the PDA has an ID $(q, y, A\alpha)$ and if there is a production $A \to \beta$ then the PDA will change to ID

$$A \to \beta \implies (q, y, \beta\alpha)$$

$$\delta(q, \epsilon, A) = \{(q, \beta)\}$$

Terminals on stack top are matched i/p.

$$\delta(q, a, a) = \{(q, \epsilon)\}$$

PDA to CFG

$M = (Q, \Sigma, \Gamma, \delta, z_0, z_0)$ - ... Acceptance by empty stack.

$G_1 = (V, \Sigma, P, S)$

$V = \{S, [p x q]\}$ $p, q \in Q$, $x \in \Gamma$

$[pxq]$ pop x

$\Rightarrow [pxq]$ Intended to pop 'x' from stack while going from p to q.

P

①

$S \rightarrow [q_0 z_0 p]$ for all $p \in Q$.

① Strings accepted by PDA are those that allow PDA to erase z from top of stack from state q_0 and ending on any state p belonging to Q.

② Let $\delta(q, a, x) = \{(r, r_1, r_2, r_k)\}$

$a \in \Sigma$ or $a \in \epsilon$

k can be 0 or more

$r, r_1, r_2 \ldots r_k$ are list of states.

Q.) 1. $\delta(P,0,z) = \{P \times z)\}$

2. $\delta(1,0,x) = \{(P,xx)\}$

3. $\delta(P,1,x) = \{(q,\epsilon)\}$

4. $\delta(q,1,x) = \{(q,\epsilon)\}$

5. $\delta(q,\epsilon,x) = \{(q,\epsilon)\}$

6. $\delta(q,\epsilon,z) = \{(q,\epsilon)\}$

$[pzp] \leftarrow A$ $\quad [pzq] \leftarrow B$

$[qzp] \leftarrow C$ $\quad [qzq] \leftarrow D$

$[pxp] \leftarrow E$ $\quad [pxq] \leftarrow F$

$[qxp] \leftarrow G$ $\quad [qxq] \leftarrow H$

$[pzp] \rightarrow 0[pxp][pzp]$

$[pzp] \rightarrow 0[pxq][qzp]$

$[pzq] \rightarrow 0[pxp][pzq]$

$[pzq] \rightarrow 0[pxq][qzq]$

$A \rightarrow 0EA \mid 0FC$

$B \rightarrow 0EB \mid 0FD$

② $\delta(P,0,x) = \{(P,xx)\}$ ...replace q's below q

$[pxp] \rightarrow 0[pxp][pxp]$

$[pxp] \rightarrow 0[pxq][qxp]$

$[pxq] \rightarrow 0[pxp][pxq]$

$[pxq] \rightarrow 0[pxq][qxq]$

③ $\delta(P,1,x) = \{(q,\epsilon)\}$

$[pxq] \rightarrow 1$

$F \rightarrow 1$

$E \rightarrow 0EE \mid 0FG$

$F \rightarrow 0EF \mid 0FG$

④ $\delta(q, 1, X) = \{(q, \epsilon)\}$

$[qXq] \rightarrow 1$

$A \rightarrow 1$

⑤ $\delta(q, \epsilon, X) = \{(q, \epsilon)\}$

$[qXq] \rightarrow \epsilon$

$H \rightarrow \epsilon$

⑥ $\delta(q, \epsilon, Z) = \{(q, \epsilon)\}$

$[qZq] \rightarrow \epsilon$

$NG = \{G, C\}$

$E, A \rightarrow$ looping

$S \rightarrow B$

$B \rightarrow 0FD$

$F \rightarrow 0FH \mid 1$

$H \rightarrow 1 \mid \epsilon$

$D \rightarrow \epsilon$

Ruleman book
Old man

00011

$S \Rightarrow B$

$\Rightarrow 0FD$

$\Rightarrow 00FHD$

$\Rightarrow 000FHHD$

$\Rightarrow 00011\epsilon D$

$\Rightarrow 000111\epsilon\epsilon$

$\Rightarrow 00011$