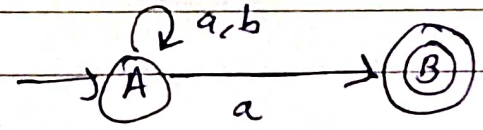
NFA to DFA

1) Make transition table for transition diagram of DFA
2) ~~Copy rows~~ Make transition table for ;

NFA to DFA

1) Make transition table for NFA
2) Make transition table for DFA by copying first row as it is.
3) In the first row find new element.
4) Fill ~~transition~~ next row for that element.
5) In second row find new element, then Step 4.
6) When there is no new element, stop.
7) Initial state will be same as that of NFA.
8) In DFA transition diagram wherever you spot final element of NFA, make that final state.

eg.



NFA

Sol^n)

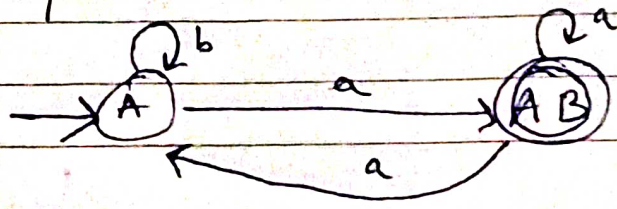
	a	b
A	{A,B}	{A}
B	—	—

TT for NFA

	a	b
A	{A,B}	{A}

	a	b
A	{A,B}	{A}
{A,B}	{A,B}	{A}

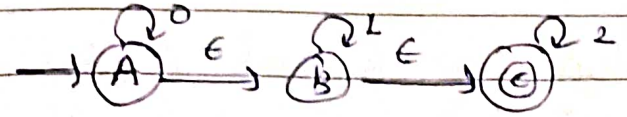
TT for DFA



ϵ-NFA to NFA

All the states that can be reached from a particular state only by seeing ϵ symbol is called ϵ closure ϵ^*

eg



ϵ^* of A: A, B, C
ϵ^* of B: B, C
ϵ^* of C: C

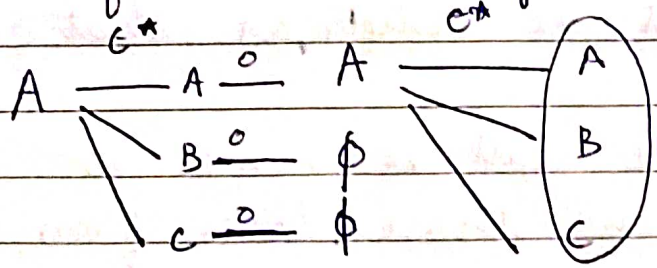
Step 1: Draw transition table for NFA

Step 2: For each ~~state~~

	0	1	2
A	{ABC}	{AB}	{C}
B			
C			

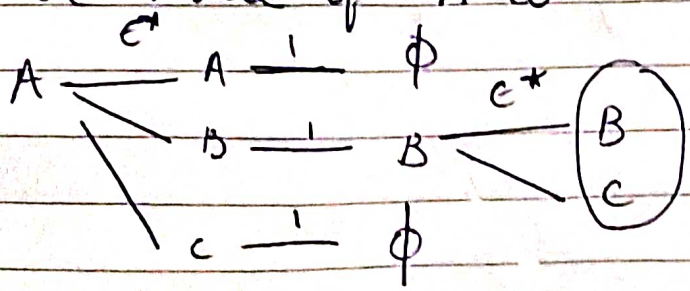
Step 2: For each state, for each input, do the following step

To find value of A at 0

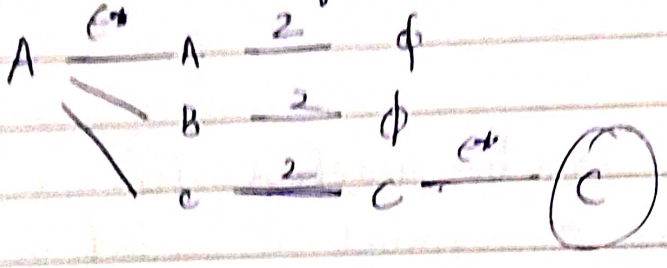


Fill this value in TT

For Value of A at 1



For value of Λ at 2



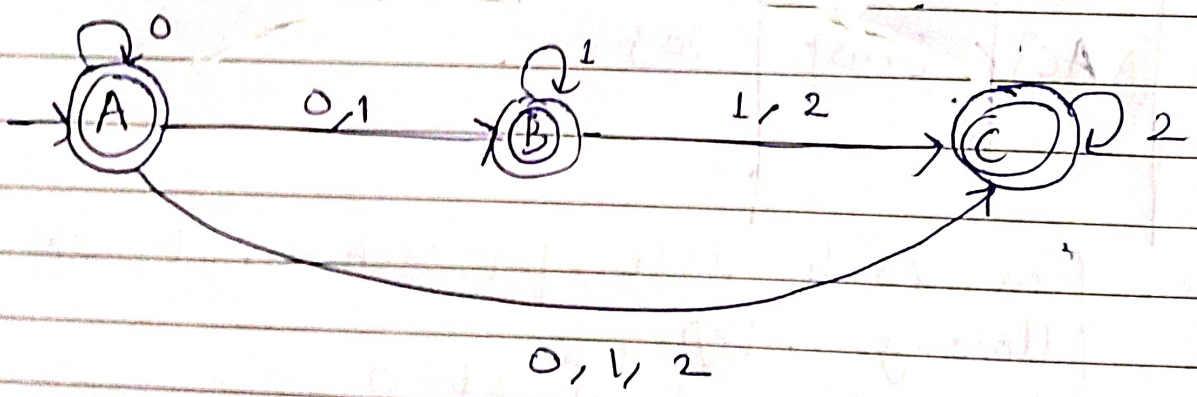
Fill it in TT

Step3 Similarly, find values for B and C at 0, 1, 2 and fill it in TT.

	0	1	2
A	{A, B, C}	{B, C}	{C}
B	{ }	{B, C}	{C}
C	{ }	{ }	{C}

TT for NFA

Step 1 Draw Transition diagram for NFA



- Initial state will be same as that of ϵ NFA
- Final ~~so~~ ~~set~~ state will be same as that of ϵ NFA and those states from where we could reach final state by reading ϵ in diagram of ϵ NFA.

Finite Automata to Regular Expression

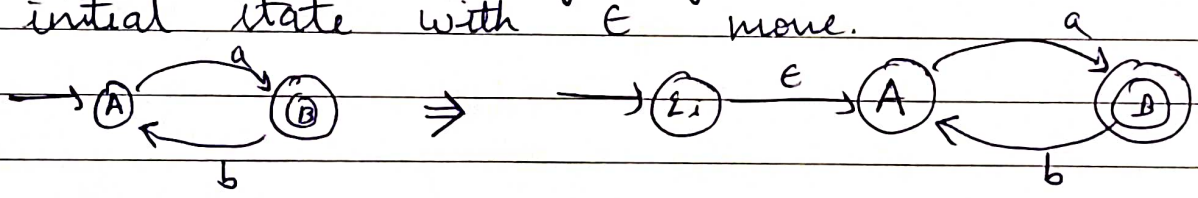
1. Arden's Theorem

If P and Q are two regular expression over Σ and if P does not contain ϵ then the following equation in R given by $\boxed{R = Q + RP}$

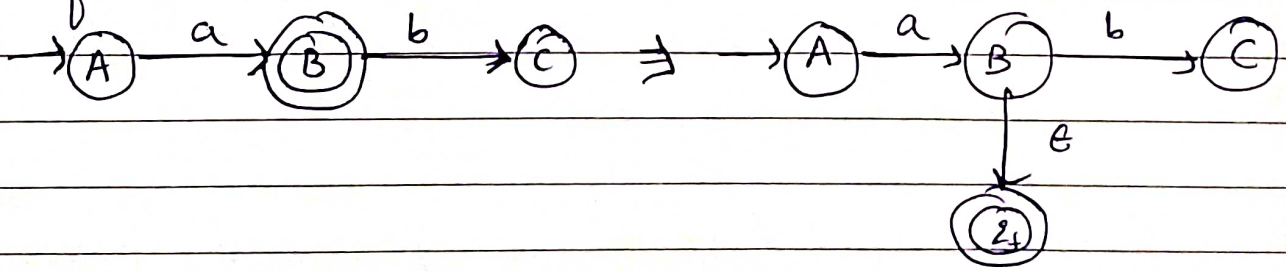
has unique solution $\boxed{R = QP^*}$

2. State Elimination Method

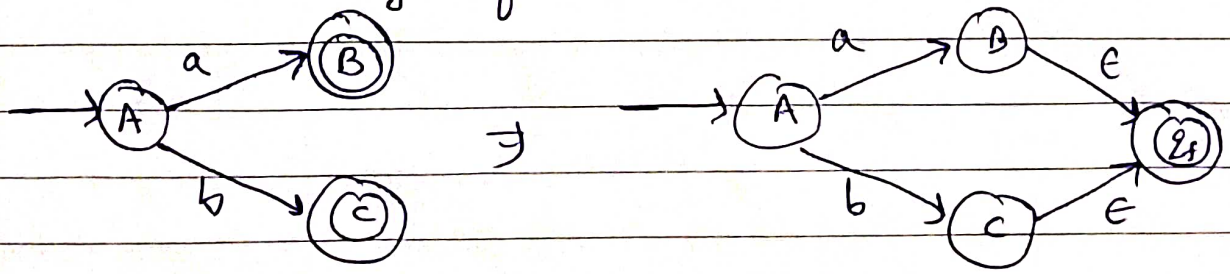
1) If there is incoming edge on initial state create new initial state with ϵ move.



2) If there is outgoing edge on final state, create new final state with ϵ move.



3) If there are more than one final states, create single final state with ϵ moves.



4) Eliminate all states one by one except initial and final state.

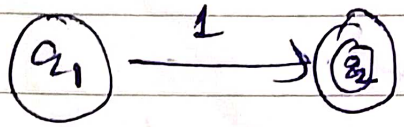
Regular Expression to Finite Automata

eg. RE = $(1+0).0^*$

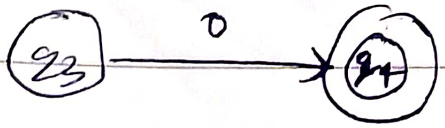
Ans) Write RE for each element

$R_1 = 1$ $R_2 = 0$ $R_3 = 0$

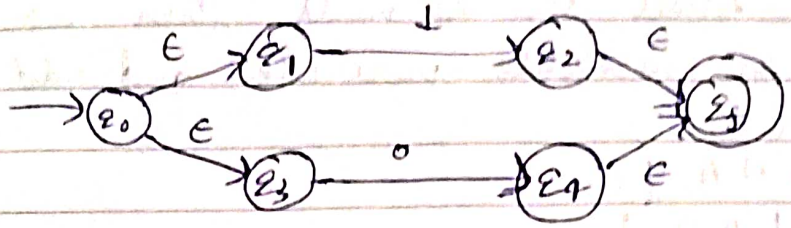
R_1



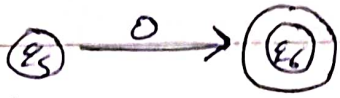
R_2



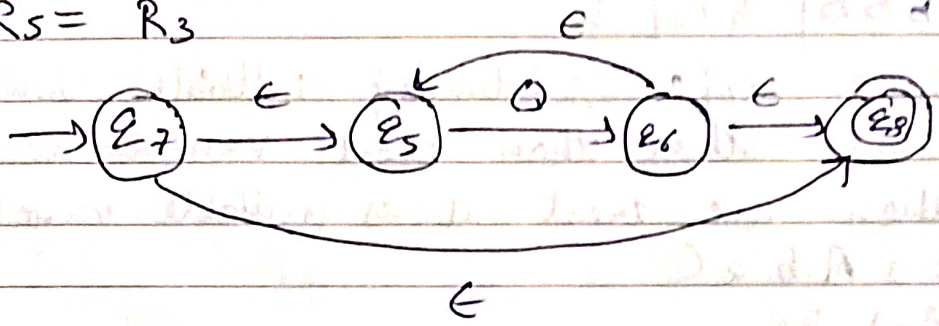
$R_4 = R_1 + R_2$



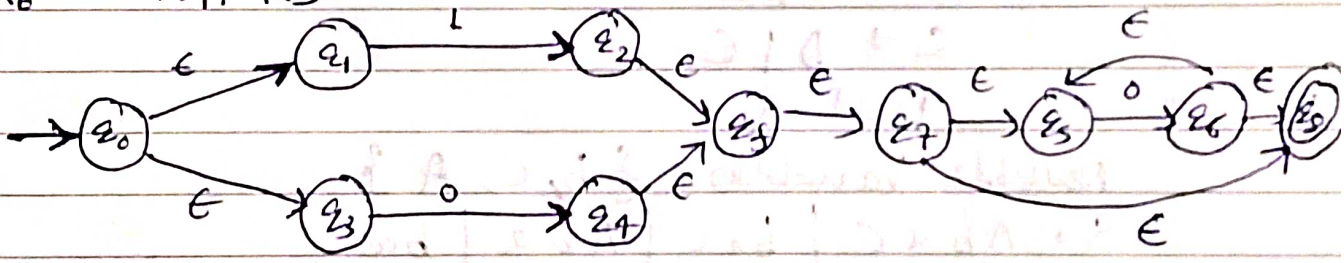
R_3



$R_5 = R_3^*$



$R_6 = R_4 . R_5$



~~Removal~~

Simplification of CFG

1. Removal of Null Production

$S \rightarrow aAB$

$A \rightarrow aAA \mid \epsilon$

$B \rightarrow bBB \mid \epsilon$

Identify nullable variables

Nullable Variable = {A, B}

$S \rightarrow aAB \mid aB \mid aA \mid a$

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

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

If after replacing nullable variables with null, start derives null then we take S as nullable variable and add ∈ to it.

eg. $S \to AB$
$A \to aAA \mid \in$
$B \to bBB \mid \in$

Nullable variables = {A, B, S}

$S \to AB \mid \in \mid A \mid B$
$A \to aAA \mid aA \mid a$
$B \to bBB \mid bB \mid b$

If ~~variable~~ after substituting nullable variables, a variable other than start variable derives null, then we treat it as nullable variable

eg. $S \to AbaC$
$A \to BC$
$B \to b \mid \in$
$C \to D \mid \in$
$D \to d$

Nullable Variables = {B, C, A}

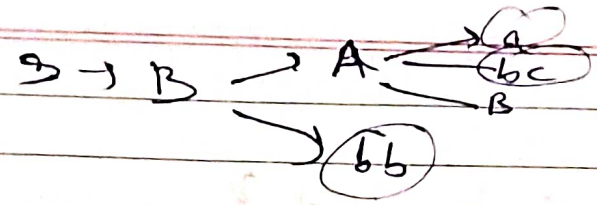
$S \to Abac \mid bac \mid Aba \mid ba$
$A \to BC \mid B \mid C$
$B \to b$
$C \to B$
$D \to d$

2. Removal of Unit Production

eg.

(1) $S \to Aa \mid B$
$B \to A \mid bb$
$A \to a \mid bc \mid B$

First we wrote productions other than unit production (length 2)



B derives bb, a, bc

Because B derives B doesn't make sense

S → Aa | bb | a | bc

B → bb | a | bc

A → a | bc | bb

(2) S → AB

A → a

B → b | C

C → D

D → E

E → a

Write productions other than unit production

S → AB

A → a

B → b

~~B → C~~ C →

D →

E → a

Now B → C → D → E → a hence B also derives a

S → AB

A → a

B → b | a

C → a

D → a

E → a

3. Removal of Useless Production

Useful Symbols = All terminals + Variables that derive terminals + Variable deriving combination of current useful symbol

S → AB | a

A → BC | b

$B \to aB \mid C$

$C \to aC \mid B$

Useful Symbols = { a, b, S, A }

Useless symbols = { B, C }

Useless production contains ~~any~~ one or more useless symbols

Here $S \to AB$, $A \to BC$, $B \to aB$, $B \to C$, $C \to aC$, $C \to B$ are useless productions

$S \to a$

$A \to b$ are useful productions

Now since we can't reach A from start

Hence $A \to b$ is also useless production and A is useless symbol

Hence only $\boxed{S \to a}$ is useful production

(2) $S \to AB \mid AC$

$A \to aAb \mid bAa \mid a$

$B \to bbA \mid aaB \mid AB$

$C \to abCA \mid aDb$

$D \to bD \mid aC$

Useful Symbols = { a, b, A, B, S }

Useless Symbols = { C, D }

$S \to AB$

$A \to aAb \mid bAa \mid a$

$B \to bbA \mid aaB \mid AB$

are useful productions

DFA Minimization

1) Remove all states that are unreachable from initial state
2) Divide in two sets [Set of Non Final States] [Set of Final States].
3) If elements of a set don't have same behaviour divide the set.
4) If set cannot be divided further, make DFA.
5) Initial state = set containing q_i
   Final state = set containing q_f

Language to Grammar

1. Language to RLG
   1) Draw finite automata
   2) For state transition $S_1 \xrightarrow{a} S_2$, grammar is $S_1 \rightarrow a S_2$
   3) For final state, we add ϵ

2. Language to LLG
   1) Draw the FA
   2) Reverse the FA by: ~~Initial state to~~ Convert initial state to final state and vice versa
      Reverse the arrows
   3) ~~Follow steps 2 and 3 only~~
   3) $S_1 \xrightarrow{a} S_2 = S_1 \rightarrow a S_2$. Add ϵ to final state.
   4) Reverse the grammar

Chomsky Normal Form

If all the production are from $A \to BC$ or $A \to a$ where A, B, C are variables and 'a' is a terminal.

Advantage:

(i) Length of each production is restricted

(ii) Derivation tree or parse tree obtained from CNF is always Binary Tree.

(iii) The no. of steps required to derive a string of length $|w|$ is $(2|w|-1)$

(iv) It is easy to apply CYK algorithm

CFG to CNF conversion

We convert all those productions to CNF who are not in CFF

eg. $S \to ABa$

$A \to aab$

~~$B \to Ac$~~

$B \to Ac$

Ans) Let $X \to a$

$Y \to b$

$Z \to c$

$\Rightarrow$ $S \to ABX$

$A \to XXY$

$B = AZ$

$X \to a$

$Y \to b$

$Z \to c$

Let $T \to AB$

and $W \to XX$

$\Rightarrow$ $S \to TX$

$A \to WY$

$B \to AZ$

$T \to AB$, $W \to XX$

$X \to a$, $Y \to b$, $Z \to c$

CNF

Parsing LL(K) Grammar

- In input string we read 1 symbol at a time
- leftmost derivation is used (leftmost non terminal will be deduced first)
- We read input string from left to right

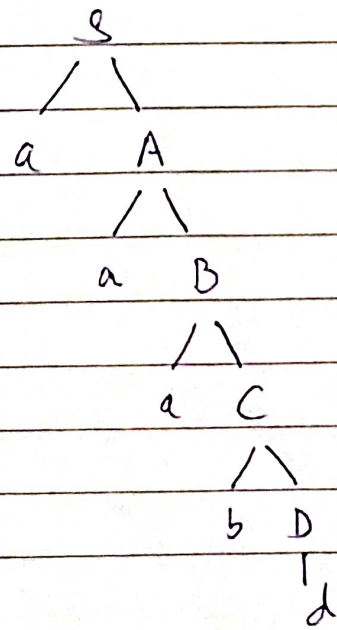
LL grammar is used for Top Down Parsing

eg. Determine if given grammar is LL(1) or not
Consider ω = a a a b d

$S \rightarrow aA \mid bB$
$A \rightarrow aB \mid cB$
$B \rightarrow bC \mid aC$
$C \rightarrow bD$
$D \rightarrow d$

Note: If it is not deterministic to select a single production then that is not LL(1) grammar

Ans)



Yes, it is LL(1) grammar

eg. $S \to abB \mid aaA$

$B \to d$

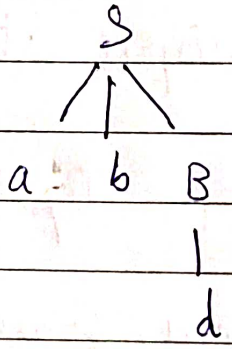
$A \to c \mid d$

here $w = abd$

Ans) It is not deterministic to read single symbol in input string.

So, It is not LL(1) grammar.

We read two symbols at a time



So, it is LL(2) grammar

Note: $LL(K) \subset LL(K+1)$

Means, if a grammar is LL(K), then it is also LL(K+1), LL(K+2), LL(K+3), LL(K+4), ... and so on.

If a grammar is not LL(K) then it can be LL(K+1)

Greibach Normal Form (GNF)

If all the production are of form $A \to a\alpha$ where $\alpha \in V^*$ then it is called GNF.

Advantages

(i) The no of steps required to generate a string of length $|w|$ is $|w|$

(ii) GNF is useful to convert CFG to PDA

~~Closure Properties of CFL~~

CFG to PDA

Case 1: CFG is not in GNF form (TV*)

Case 2: CFG is in GNF form (TV*)

Case 1: CFG is not in GNF form

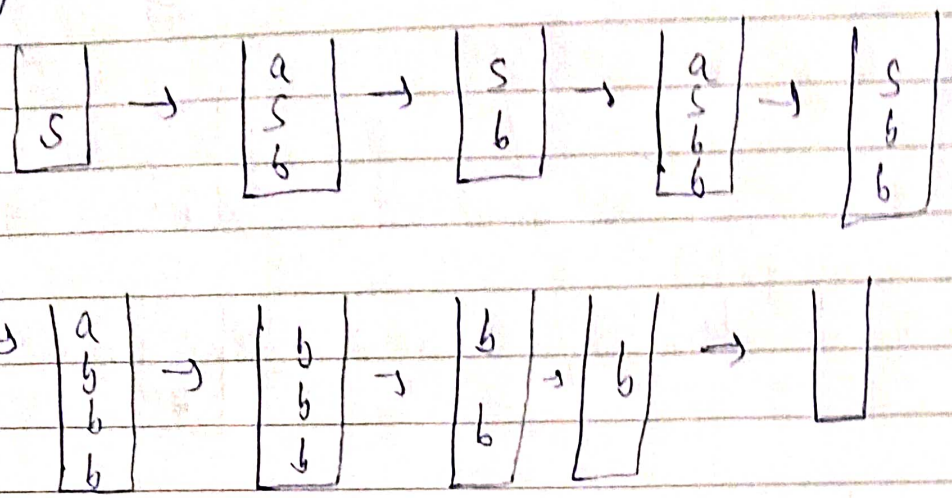
* stack start symbol will be S
* Top of the stack can be Variable of Terminal
* No final state
* If input and top of the stack is same then we move the pointer forward.
* If top of the stack is variable then we don't move the pointer.

eg. $S \to aSb$
$S \to ab$

		#Rule
$\delta(q, \epsilon, S) = \delta(q, aSb)$	}	For Variable
$\delta(q, \epsilon, S) = \delta(q, ab)$	}	
$\delta(q, a, a) = \delta(q, \epsilon)$	}	For Terminal
$\delta(q, b, b) = \delta(q, \epsilon)$	}	

eg. String $w =$ aaabbb

eg. a a a b b b



Case 2: CGF is in GNF form

* GNF form → TV* where T is terminal and V is variable
* Stack symbols are only variables
* We move the pointer in each case
* Stack start symbol is S
* Input symbol is terminal

eg.

S → 0BB

B → 0S | 1S | 0

#Rule

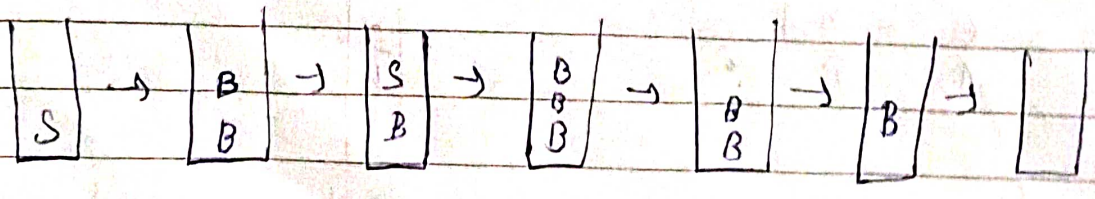
δ(q, 0, S) = δ(q, BB)

δ(q, 0, B) = δ(q, S)

δ(q, 1, B) = δ(q, S)

δ(q, 0, B) = δ(q, ∈)

eg. 0 1 0 0 0 0



1. $\delta(q_0, a, z_0) = (q_0, Xz_0)$

2. $\delta(q_0, a, X) = (q_0, XX)$

3. $\delta(q_0, b, X) = (q_1, \epsilon)$ $a^n b^n | n \geq 1$

4. $\delta(q_1, b, X) = (q_1, \epsilon)$

5. $\delta(q_1, \epsilon, z_0) = (q_1, \epsilon)$

$M = (\underset{Q}{\{q_0, q_1\}}, \underset{\Sigma}{\{a, b\}}, \delta, \underset{\Gamma}{\{z_0, X\}}, q_0, z_0, \underset{F}{\phi})$

N.T.

$S \cup [q, A, p] \quad q, p \in Q, A \in \Gamma$

1) $S \rightarrow [q_0, z_0, p]$ for each p.

2) if $\delta(q, x, A) = (p, B_1 B_2 \ldots B_m)$

$[q, A, q_{m+1}] \rightarrow x[p, B_1, q_2][q_2, B_2, q_3] \ldots [q_m, B_m, q_{m+1}]$

3) if $\delta(q, x, A) = (p, \epsilon)$

$[q, A, p] \rightarrow x \quad x \in (\Sigma \cup \{\epsilon\})$

$q(V, \Sigma, P, S)$ ←

① ✓ A

✓ $S \to [q_0, z_0, q_0]$, $S \to [q_0, z_0, q_1]$

① ✗ $[q_0, z_0, q_0] \to a\,[q_0, x, q_0]\,[q_0, z_0, q_0]$

✗ $[q_0, z_0, q_0] \Rightarrow a\,[q_0, x, q_1]\,[q_1, z_0, q_0]$ ✗

✗ $[q_0, z_0, q_1] \to a\,[q_0, x\ q_0]\,[q_0, z_0, q_1]$

② $[q_0, z_0, q_1] \to a\,[q_0, x, q_1]\,[q_1, z_0, q_1]$ N.T.

② ✓ $[q_0, x, q_0] \to a\,[q_0, x, q_0]\,[q_0, x, q_0]$ 1)

✗ $[q_0, x, q_0] \to a\,[q_0, x, q_1]\,[q_1, x, q_0]$ 2) ✗

✗ $[q_0, x, q_1] \to a\,[q_0, x, q_0]\,[q_0, x, q_1]$

③ $[q_0, x, q_1] \to a\,[q_0, x\ b]\,[b, x, q_1]$

1. $\delta(q_0, a, z_0) = (q_0, x z_0)$ ✓
2. $\delta(q_0, a, x) = (q_0, xx)$ ✓ ✗ push
3. $\delta(q_0, b, x) = (q_1, \epsilon)$ ✓
4. $\delta(q_1, b, x) = (q_1, \epsilon)$
5. $\delta(q_1, \epsilon, z_0) = (q_1, \epsilon)$ ← $a^n b^n | n \geq 1$

$M = (\underset{Q}{\{q_0, q_1\}}, \underset{\Sigma}{\{a, b\}}, \delta, \underset{\Gamma}{\{z_0, x\}}, q_0, z_0, \underset{F}{\phi})$

3. $[q_0, x, q_1] \to b$ 4 (B)
4. $[q_1, x, q_1] \to b$ 5 (C)
5. $[q_1, z_0, q_1] \to \epsilon$ 6. (D)

$= 1 + |\Gamma| \times |Q|^2$ (Q) (T)