CD Unit-2

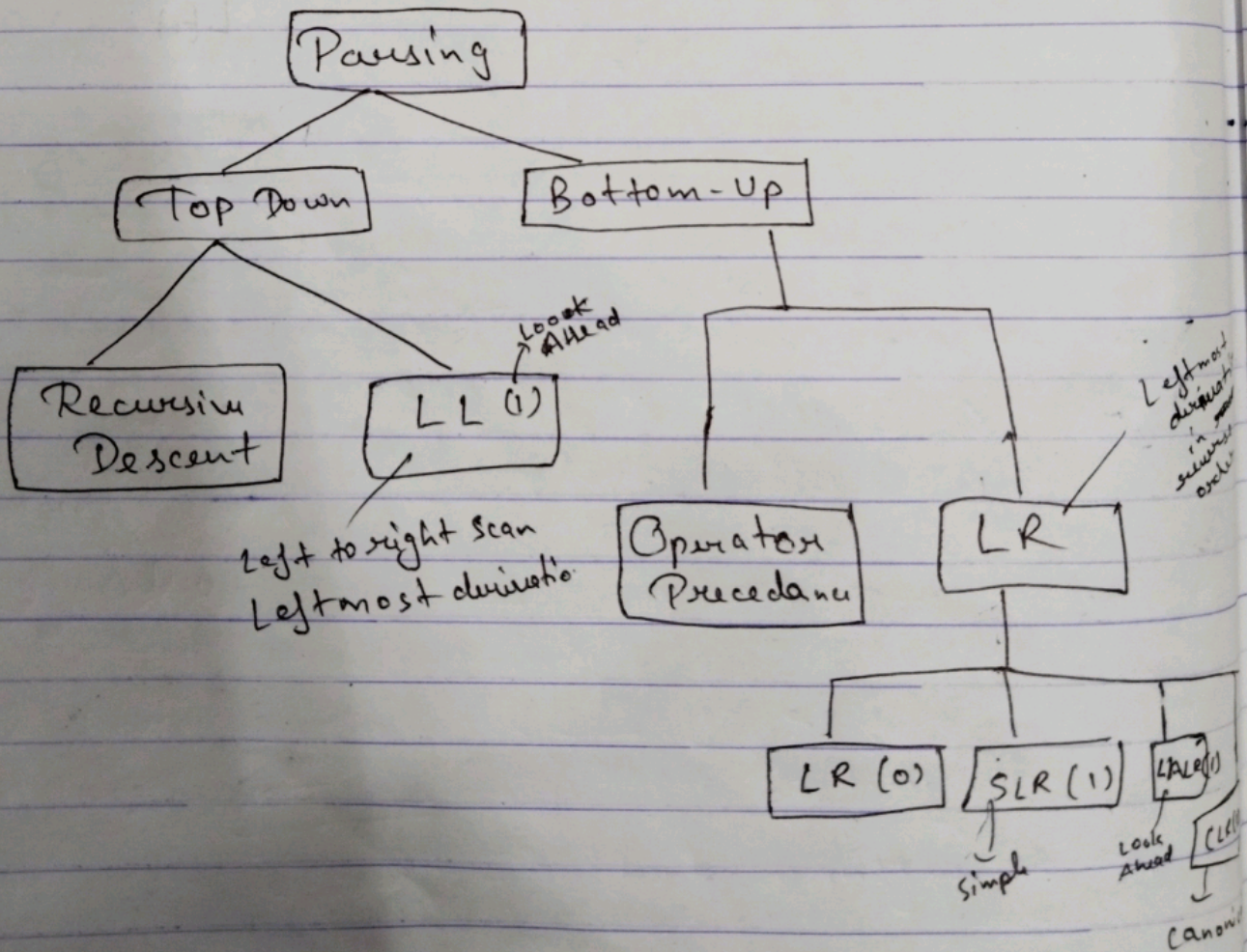
Parsing :-

Parsing can be works on the following principles -

(i) The Parser scan the input string from left to right & identifies that the derivation is leftmost or rightmost

(ii) The Parser makes use of production rules for choosing the appropriate derivation.

* The different parsing technique uses different approaches in selecting the appropriate rules for derivation & finally a parse tree is constructed



Top Down Parsing

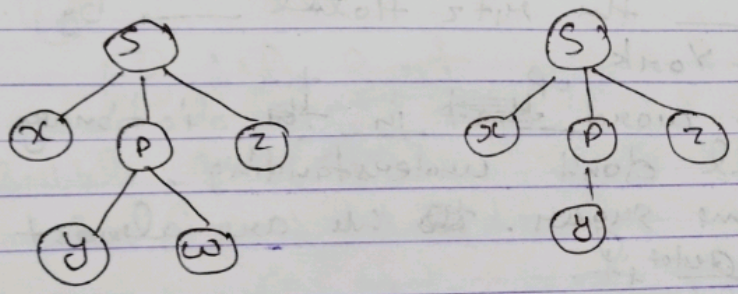
Problems in top down parsing :-

(1) **Back tracking** :- It is a technique in which for expansion of non-terminal symbol we choose one alternative & if some mismatch occurs then we try another alternative if any.

eg-

$S \rightarrow xPz$

$P \rightarrow yw \mid y$



(2) **Left Recursion** :-

$A \rightarrow A\alpha \mid B$

$A \rightarrow BA'$

$A' \rightarrow \alpha A' \mid \epsilon$

eg $A \rightarrow ABd \mid Aa \mid a$ Remove left recursion

$B \rightarrow Be \mid b$

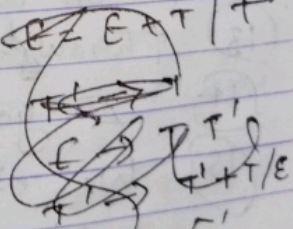
$A \Rightarrow B \rightarrow bB'$

$B' \rightarrow eB' \mid \epsilon$

$A \rightarrow aA'$

$A' \rightarrow BdA' \mid aA' \mid \epsilon$

eg $E \rightarrow E+T \mid T$



$E' \rightarrow +TE'$

$E' \rightarrow$

CD

Qny $E = E+T/T$ Remove left recursion

$\Rightarrow$ $E' \to TE'$

$E' \to +TE'/\epsilon$

(3) Left factoring :- is used when it is not cleared that which of the two alternatives is used to expend the non-terminal.

$A \to \alpha B_1 / \alpha B_2$

$A \to \alpha A'$

$A' \to B_1 / B_2$

Qny $S \to iEtS / iEtSeS / a$

$E \to b$

$\Rightarrow$ ~~$S \to iEtSS'$~~ ~~$iEtS$~~

~~$S \to iEtS A' / S'eS / a$~~

$S' \to E/eS$

~~$S \to a$~~

$E \to b$

$S \to iEtSS'/a$

$S' \to eS/e$

$E \to b$

Qny $A \to aAB / aA / a$

$B \to bB / b$

$\Rightarrow$ $A \to aAA'$

$A' \to AB / A / \epsilon$ $\Rightarrow$ $A' \to AA'' / \epsilon$

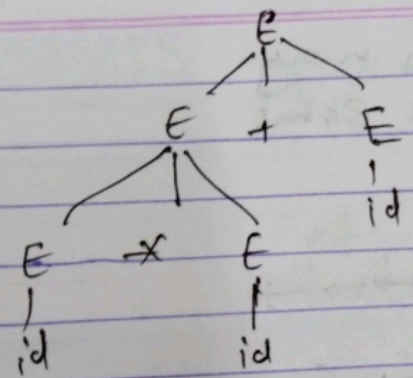
$B \to bB'$

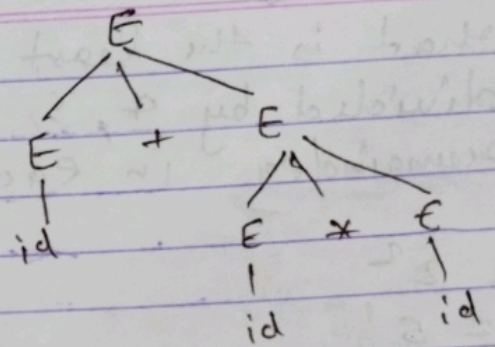
$B' \to BB / \epsilon$ $A'' \to B / \epsilon$

(7) Ambiguity :-

$E \longrightarrow E + E \mid E * E \mid id$

id + id * id





CD

Top Down Parsing :-

LL (1) Parser :-

first ( ) :-

first(α) is a set of terminal symbols that begin in strings derived from α

$A \longrightarrow abc \mid def \mid ghi$

$first(A) = \{a, d, g\}$

Rules for calculating first ( ) function :-

i) for a production rule.

$X \rightarrow \epsilon$

$first(X) \longrightarrow \{\epsilon\}$

ii) for any terminal symbols 'a'

$first(a) = \{a\}$

iii) for any production rule :-

$X \longrightarrow Y_1 Y_2 Y_3$

* if $\epsilon \notin first(Y_1)$ then first(X) = first(Y_1)

$X \rightarrow Y_1 Y_2 Y_3 \mid a \mid b \mid c$

$first(X) = \{ first(Y_1), a, b, c \}$

* if $\epsilon \in first(Y_1)$, then

$first(X) = \{ first(Y_1) - \epsilon \} \cup \{ first(Y_2, Y_3) \}$

Q1 $S \rightarrow aBDh$

$B \rightarrow cC$

$C \rightarrow bC / \epsilon$

$D \rightarrow \epsilon f$

$E \rightarrow g / \epsilon$

$f \rightarrow f(\epsilon$

Sol^n

$first(S) = \{a\}$

$first(B) = \{c\}$

$first(c) = \{\epsilon, b\}$

$first(D) = \{g, f, \epsilon\}$

$first(\epsilon) = \{g, \epsilon\}$

$first(f) = \{f, \epsilon\}$

Q2 $S \rightarrow A$

$A \rightarrow aB / Ad$

$B \rightarrow b$

$C \rightarrow g$

Sol^n

$first(S) =$

$first(A) =$

$first(B) = \{b\}$

$first(C) = \{g\}$

CD

ε is not in follow()

follow (α) :-

follow is a set of terminal symbols that appears immediately to the right of α'

Rules for ~~for~~ calculating follow() function:-

(i) for the start symbol S, follow (S) = $ (dollar)

(ii) for any production,

$A \to \alpha B$

follow (B) = follow (A)

e.g- $A \to BG$

$B \to fg / i$

$G \to e / i$

follow (A) = $

follow (B) = first (G) = {e, i}

follow (G) = $

(iii) for the production rule,

$A \to \alpha B \beta$

* if $\epsilon \notin$ first (B) then follow (B) = first (B)
* if $\epsilon \in$ first (B) then follow (B) = {first (B) − ε} ∪ follow (A)

Ex $S \to aBDh$

$B \to cC$

$C \to bC / \epsilon$

$D \to EF$

$E \to g / \epsilon$

$F \to f / \epsilon$

follow (S) = {$}

follow (B) = {g, f, h}

follow (C) = {g, f, h}

follow (D) = {h}

follow (E) = {f, h}

follow (F) = {h}

D → g, ε, f, ε {gf, f, ε} ∪ h gf, h

CD

Qy $S \rightarrow AaAb \mid BbBa$
$A \rightarrow \epsilon$
$B \rightarrow \epsilon$

$first(S) = \{a, b\}$
$first(A) = \{\epsilon\}$
$first(B) = \{\epsilon\}$

$follow(S) = \{\$\}$
$follow(A) = \{a, b\}$
$follow(B) = \{b, a\}$

Qy $S \rightarrow (L) / a$
$L \rightarrow SL'$
$L' \rightarrow SL' / \epsilon$

$first(S) = \{(, a\}$
$first(L) = \{(, a\}$
$first(L') = \{(, a, \epsilon\}$

$follow(S) = \{\$,), (, a,)\}$
$follow(L) = \{)\}$
$follow(L') = \{)\}$

Qy
$S \rightarrow ACB \mid CbB \mid Ba$
$A \rightarrow da \mid BC$
$B \rightarrow g \mid \epsilon$
$C \rightarrow h \mid \epsilon$

$first(S) = \{d, g, h, \epsilon, b, a\}$
$first(A) = \{d, g, h, \epsilon\}$
$first(B) = \{g, \epsilon\}$
$first(C) = \{h, \epsilon\}$

$follow(S) = \{\$\}$
$follow(A) = \{h, g, \$\}$
$follow(B) = \{\$, a, h, g\}$
$follow(C) = \{g, \$, b, h\}$

Q. $E \to E + T / T$ $\Rightarrow$ $E \to TE'$, $E' \to +TE' / \epsilon$

$T \to T * F / F$ $\Rightarrow$ $T \to FT'$, $T' \to *FT' / \epsilon$

$F \to (E) / id$

$A = A\alpha / \beta$

$A = \beta A'$, $A' = \alpha A' / \epsilon$

first(E) = { (, id }

first(T) = { (, id }

first(E') = { +, ϵ }

first(T') = { *, ϵ }

first(F) = { (, id }

follow(E) = { $,) }

follow(T) = { +, $,) }

follow(E') = { $,) }

follow(T') = { +, $,) }

follow(F) = { *, $, +,) }

LL(1) Parse Table :-

	id	+	*	(	)	$
E	$E \to TE'$			$E \to TE'$		
E'		$E' \to +TE'$			$E' \to \epsilon$	$E' \to \epsilon$
T	$T \to FT'$			$T \to FT'$		
T'		$T' \to \epsilon$	$T' \to *FT'$		$T' \to \epsilon$	$T' \to \epsilon$
F	$F \to id$			$F \to (E)$		

Row → non-terminal

Colum → terminal

Stack	Input	Action
$E	id + id * id $	
$E'T	id + id * id $	$E \to TE'$
$E'T'F	id + id * id $	$T \to FT'$
$E'T'id	id + id * id $	$F \to id$
$E'T'	+ id * id $	
$E'	+ id * id $	$T' \to \epsilon$
$E'T+	+ id * id $	$E \to +TE'$
$E'T	id * id $	
$E'T'F	id * id $	$T \to FT'$
$E'T'id	id * id $	$F \to id$
$E'T'	* id $	
$E'T'F*	* id $	$T' \to *FT'$
$E'T'F	id $	
$E'T'id	id $	$F \to id$
$E'T'	$	
$E'	$	$T' \to \epsilon$
$	$	$E' \to \epsilon$

help of co

CD

Qus $S \rightarrow AaAb \mid BbBa$
$A \rightarrow \epsilon$
$B \rightarrow \epsilon$

Show that above grammar is LL(1) or not

first(S) = {a, b}
first(A) = {ε}
first(B) = {ε}

follow(S) = {#}
follow(A) = {a, b}
follow(B) = {b, a}

	a	b	#
S	S → AaAb	S → BbBa	
A	A → ε	A → ε	
B	B → ε	B → ε	

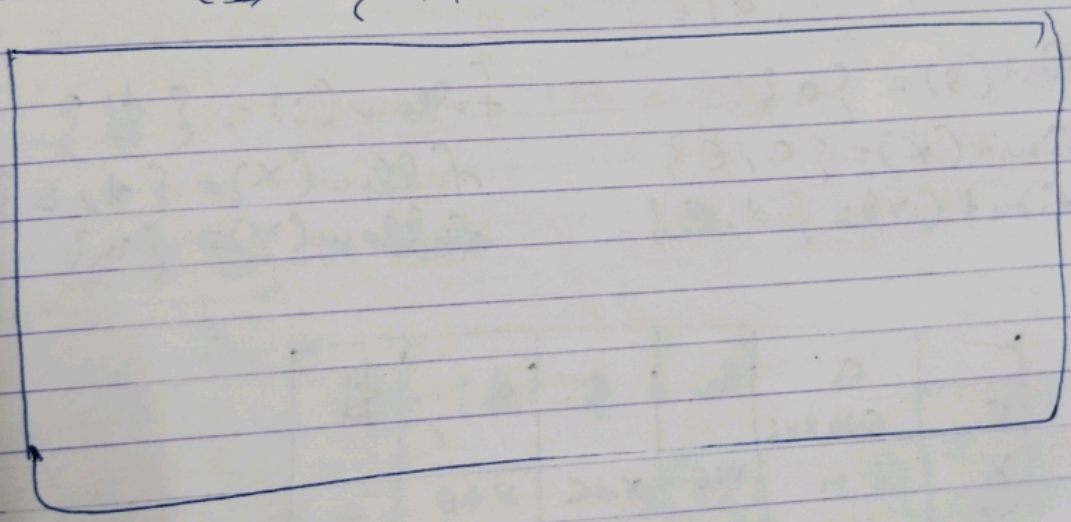
Qus $A \rightarrow AcB \mid cC \mid C$
$B \rightarrow bB \mid id$
$C \rightarrow CaB \mid BbB \mid B$

$\Rightarrow$ $A \rightarrow cCA' \mid CA'$
$A' \rightarrow CBA' \mid \epsilon$

$\Rightarrow$ $C \rightarrow BbBC' \mid BC'$
$C' \rightarrow aBC' \mid \epsilon$

first(A) = { c, b, id }
first(A') = { c, ∈ }
first(B) = { b, id }
first(C) = { b, id }
first(C') = { a, ∈ }

follow(A) = { $ }
follow(A') = { $ }
follow(B) = { $, b, a }
follow(C) = { c, $ }



Qn S → iCtSA / a
A → eS / ∈
C → b

first(S) = { i, a }
first(A) = { e, ∈ }
first(C) = { b }

follow(S) = { $, e }
follow(A) = { $, e }
follow(B) = { t }

	i	t	e	a	b	$
S	S→iCtSA			S→a		
A			A→eS			A→ϵ
C					C→b	

It is not a LL(1) parsel

Q) S → aXYb
X → c | ϵ
Y → d | ϵ

first(S) = {a}
first(X) = {c, ϵ}
first(Y) = {d, ϵ}

follow(S) = {$}
follow(X) = {d, b}
follow(Y) = {b}

	a	b	c	d	$
S	S→aXYb				
X		X→ϵ	X→c	X→ϵ	
Y		Y→ϵ		Y→d	

CD

Recursive Descent Parsing :-

A parser that uses collection of recursive procedure for parsing the given input string is called recursive descent parser.

$E \rightarrow iE'$

$E' \rightarrow +iE' / \epsilon$

```
E()
{ if (l == 'i')
  { match('i');
    E'();
  }
}
```

```
E'()
{ if (l == '+')
  { match('+')
    match('i');
    E'()
  }
  else
    return;
}
```

```
match(char t)
{
  if (l == t)
    l = getchar();
  else
    printf("error");
}
```

```
main()
{ E();
  if (l == '$')
    printf("parsing success");
}
```

Bottom-up Parsing :-

Handle Pruning :-

In bottom up parsing to find the substring that could be reduced by appropriate non-terminal such a substring is called handle pruning

$E \rightarrow E + E$

$E \rightarrow id$

Right sential form	Handle	Production
id + id + id	id	$E \rightarrow id$
E + id + id	id	$E \rightarrow id$
$E + E$ + id	id	$E \rightarrow id$
$E + E + E$	$E+E$	$E \rightarrow E+E$
$E + E$	$E+E$	$E \rightarrow E+E$
E		

CRT

quantity

CD

Shift Reduce Parser

$E \to E+E$
$E \to E*E$
$E \to id$

id + id * id $

Stack	Input Buffer	Parsing Action
$	id + id * id $	shift
$ id	+ id * id $	E → id
$ E	+ id * id $	shift
$ E +	id * id $	shift
$ E + id	* id $	E → id
$ E + E	* id $	shift
$ E + E *	id $	shift
$ E + E * id	$	E → id
$ E + E * E	$	E → E * E
$ E + E	$	E → E + E
$ E	$	

Operator Precedence Parser

Operator precedence parsing can be done with operator precedence grammar only. The grammar 'G' is said to be operator precedence grammar if it posses the following properties :-

(i) No production on the right side is ∈.
(ii) There should not be any production rule possesing two adjacent non-terminals at the right hand side.

eg- $S \to bCB / Ae / BeC$
$A \to f / e$
$B \to d / c$
$C \to f / g$

~~A to~~ It is not operator precedence grammar

$E \to EAE / (E) / -E / id$

$A \to + / - / * / / / \uparrow$

$E \to E+E / E-E / E*E / E/E / E \uparrow E$

$E \to (E) / -E / id$

P & q are terminals

* $p < \cdot q \Rightarrow$ p का preference q से कम है।
* $p > \cdot q \Rightarrow$ p का preference q से ज्यादा है।
* $p = \cdot q \Rightarrow$ p का preference q के बराबर है।

Q. id + id * id

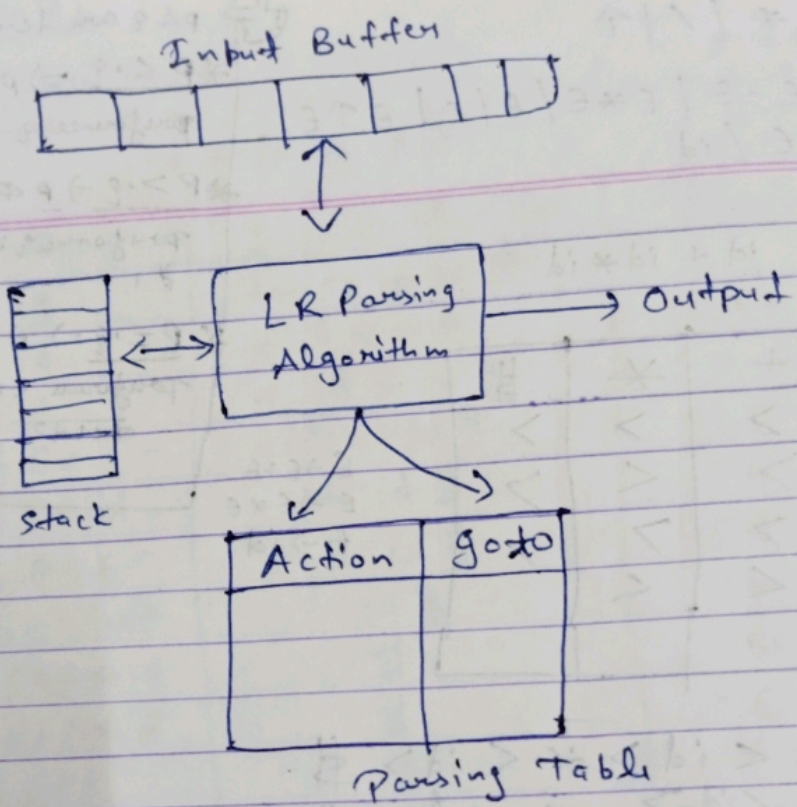
	id	+	*	$
id		>	>	>
+	<	>	<	>
*	<	>	>	>
$	<	<	<	

$E \to E+E$
$E \to E*E$
$E \to id$

$ < id > + < id > * < id > $
$ E + < id > * < id > $
$ E + E * < id > $
$ E + E * E $
$ < + < * > $
$ < + > $
$ $

LR Parser

- LR(0)
- SLR(1) — Canonical collection of LR(0) items
- LALR
- CLR(1) — Canonical collection of LR(1) items



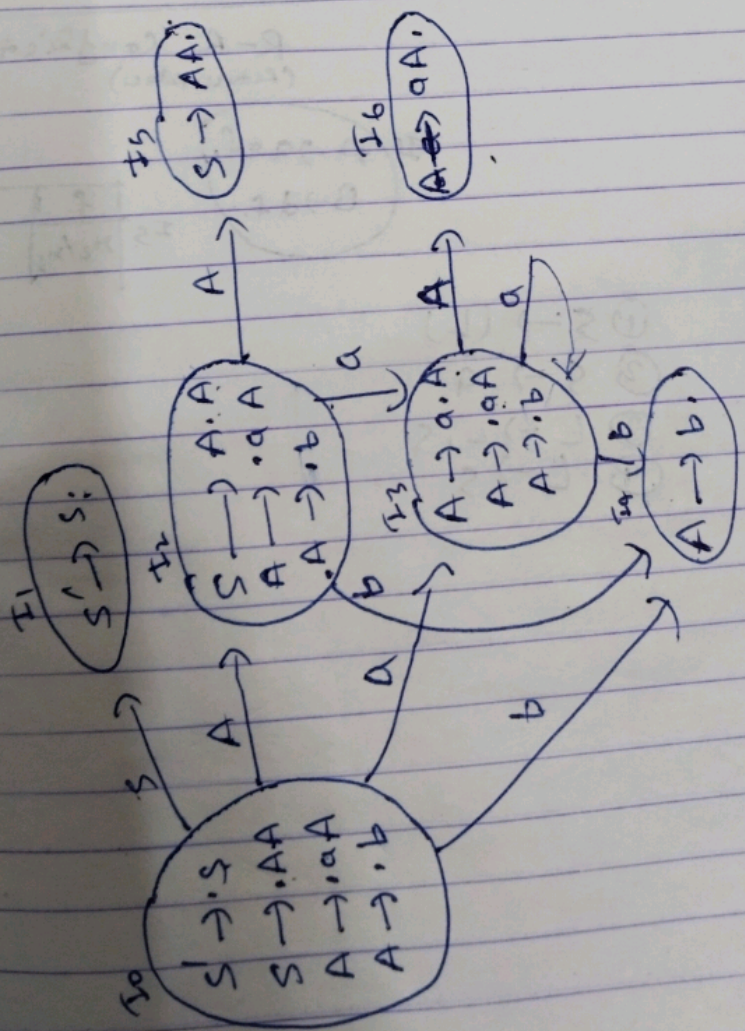
LR(0) Parser :-

1. $S \rightarrow AA$
2. $A \rightarrow aA$
3. $A \rightarrow b$

Augmented Grammar:
$S' \rightarrow .S$

~~Action~~

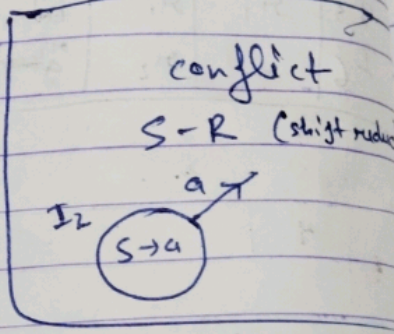
State	Action			Goto	
	a	b	$	S	A
I_0	s_3	s_4		1	2
I_1			accept		
I_2	s_3	s_4			5
I_3	s_3	s_4			6
I_4	r_3	r_3	~~accept~~ r_3		
I_5	r_1	r_1	~~accept~~ r_1		
I_6	r_2	r_2	~~accept~~ r_2		



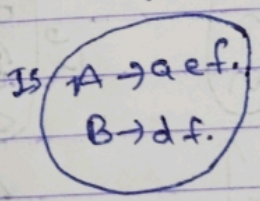
SLR(1) :-

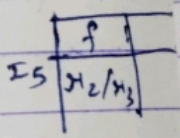
State	Action			Goto	
	a	b	$	S	A
I_0	S_3	S_4		1	2
I_1			Accept		
I_2	S_3	S_4			5
I_3	S_3	S_4			6
I_4	R_3	R_3	R_3		
I_5			R_1		
I_6	R_2	R_2	R_2		

① S → AA
② A → aA
③ A → b



R-R conflict
(Reduce-Reduce)

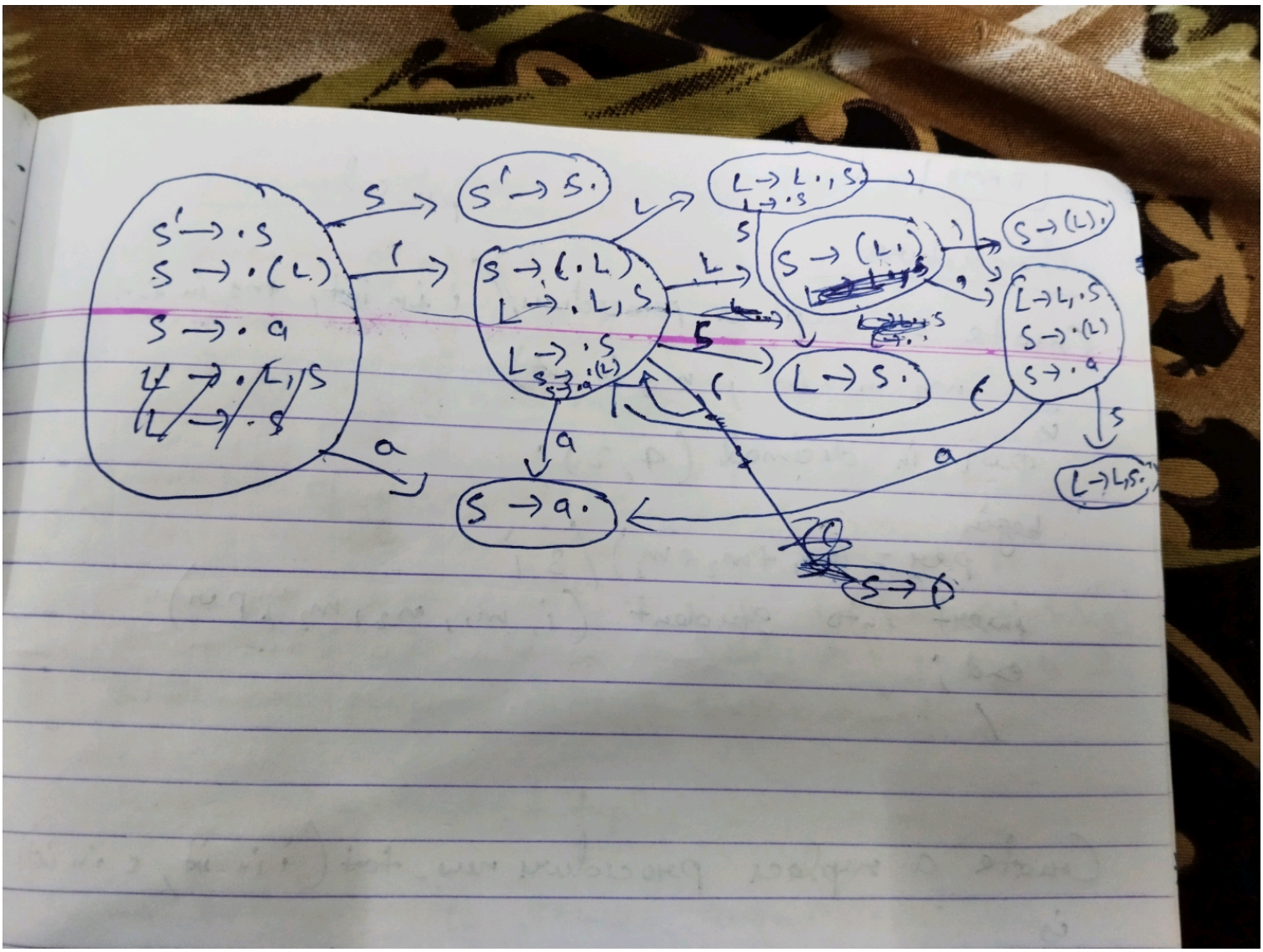




Ex S → (L)/a
L → L,S/S

S' → .S
S → .(L)
S → .a
L → .L,S
L → .S

① S → (L)
② S → a
③ L → L,S
④ L → S

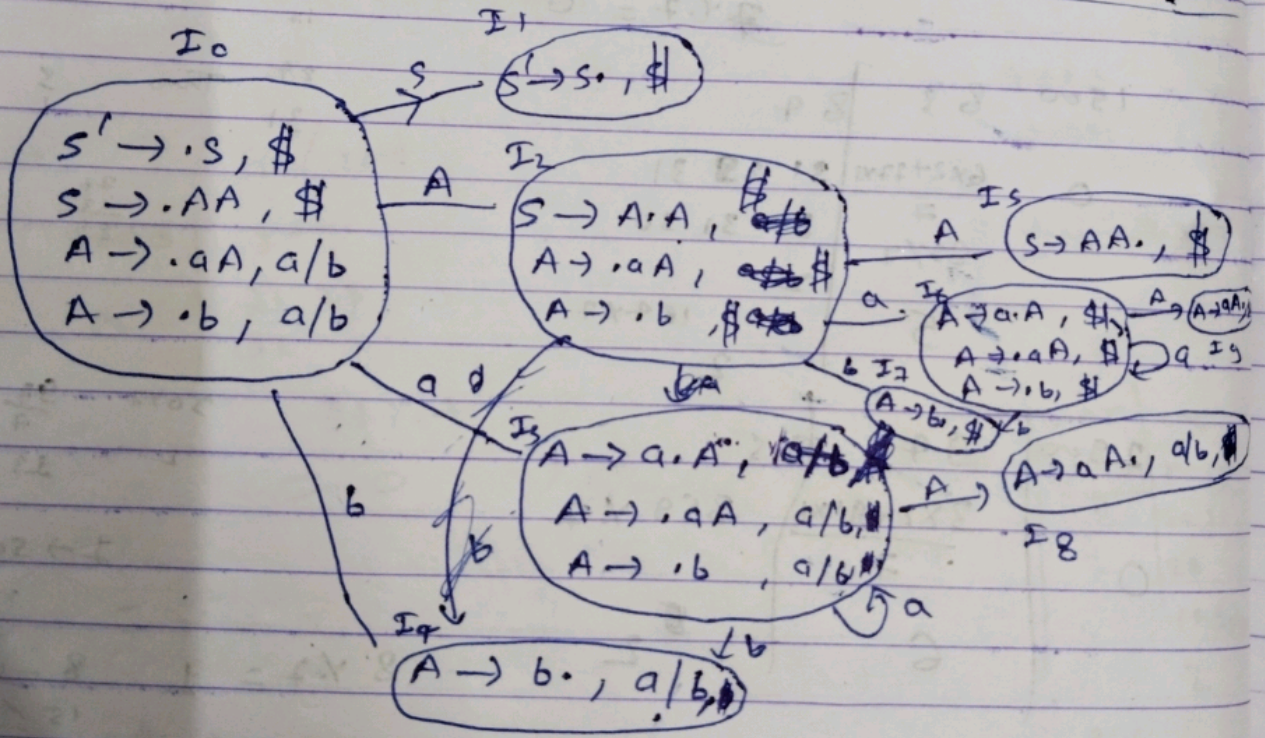

S' → .S
S → .(L)
S → .a
L → .L,S
L → .S
S
(
a
S' → S.
S → (.L)
L → .L,S
L → .S
S → .(L)
S → .a
L
S
(
a
L → L.,S
L → .S
S → (L.)
L → S.
S → a.
S → (L).
L → L,.S
S → .(L)
S → .a
L → L,S.

CD

CLR(1)

① $S \to AA$
② $A \to aA$
③ $A \to b$

Canonical collection of LR(1) items
⇓
Canonical collection of LR(0) items + Look Ahead



Roman:-

CD

CLR(1) table

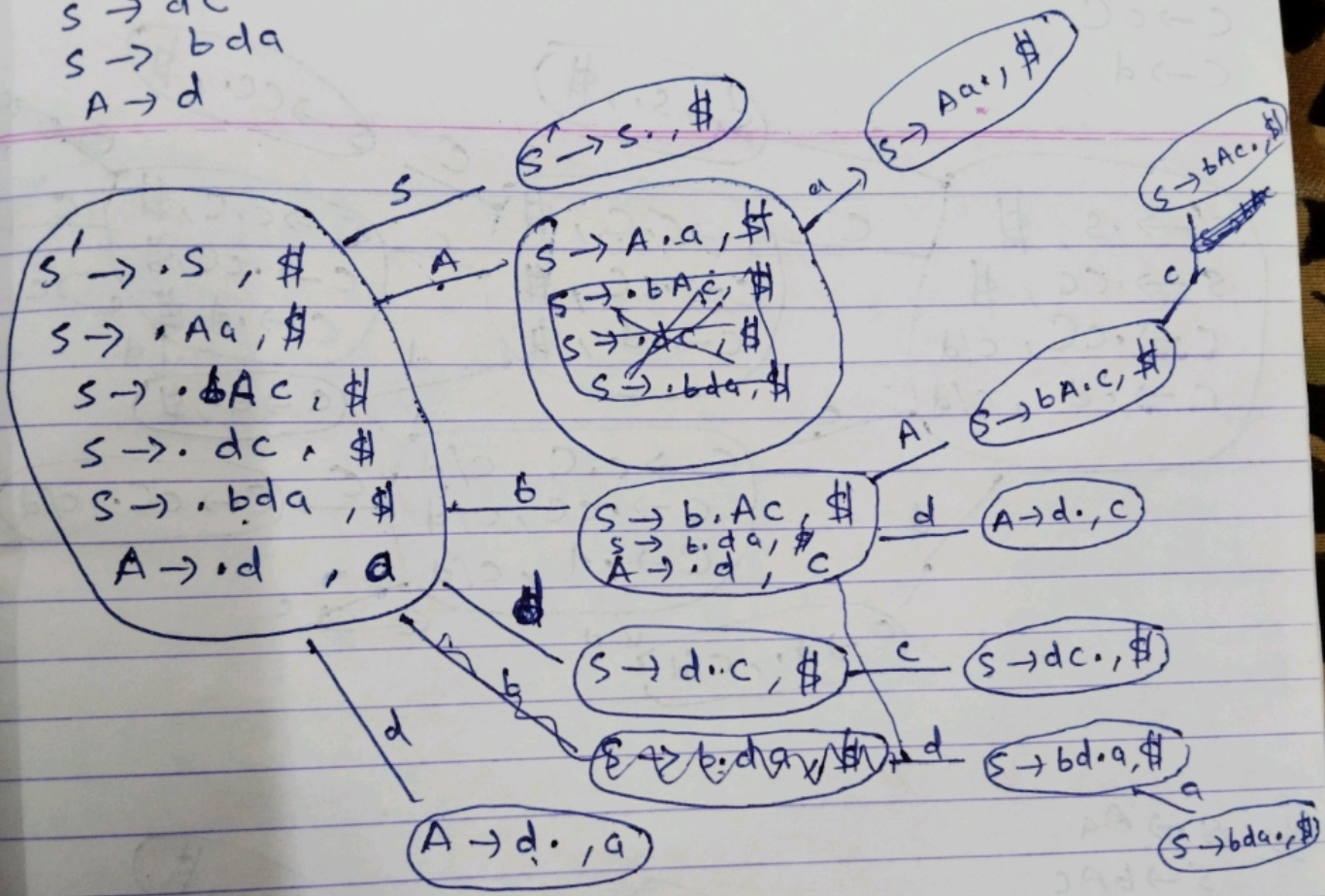
State	a	b	$	S	A
I_0	s_3	s_4		1	2
I_1			Accept		
I_2	s_6	s_7			5
I_3	s_3	s_4			8
I_4	r_3	r_3	r_3		
I_5			r_1		
I_6	s_6	s_7			9
I_7			r_3		
I_8	r_2	r_2			
I_9			r_2		

LALR(1)

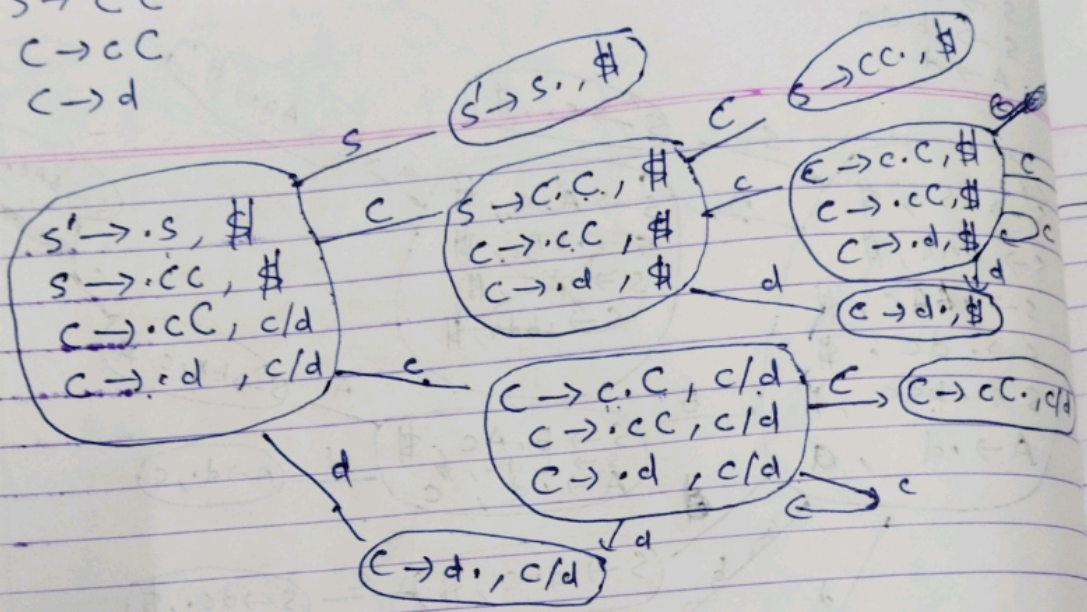
State	a	b	$	S	A
I_0	s_{36}	s_{47}		1	2
I_1			Accept		
I_2	s_{36}	s_{47}			5
I_{36}	s_{36}	s_{47}			89
I_{47}	r_3	r_3	r_3		
I_5			r_1		
I_{89}	r_2	r_2	r_2		

3 & 6 = 36
4 & 7 = 47
8 & 9 = 89

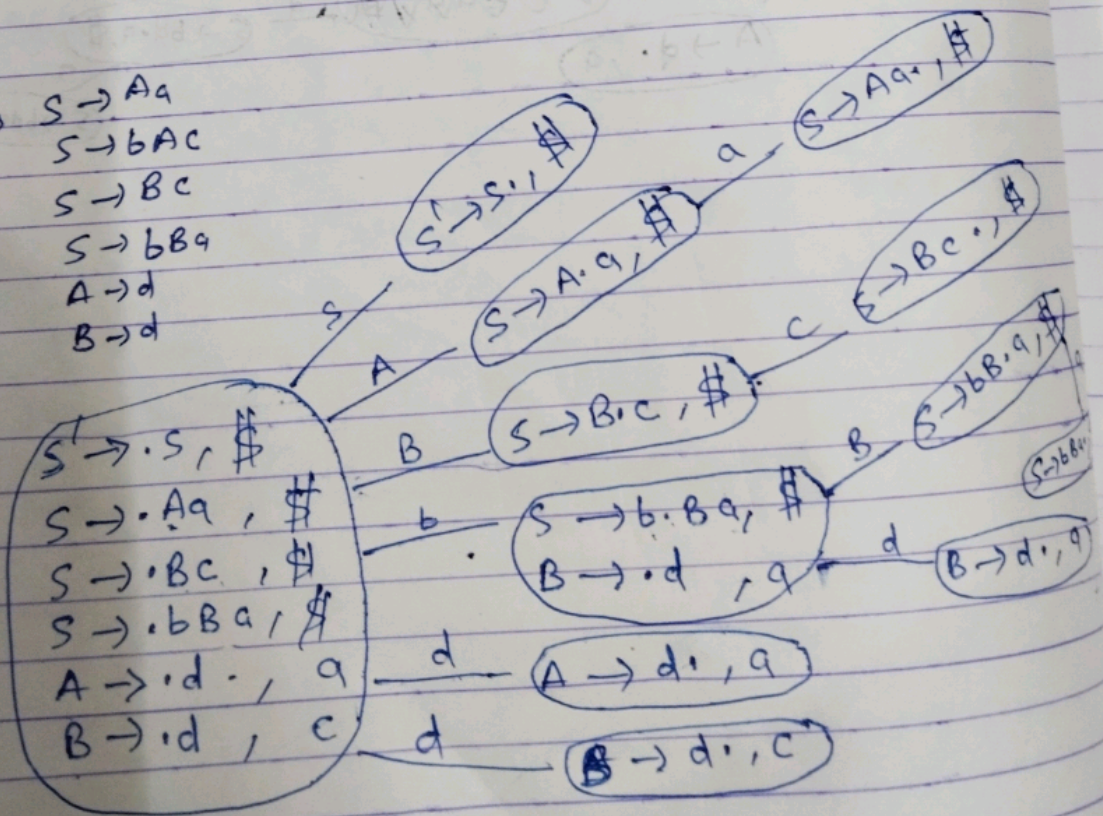
$S \to Aa$
$S \to bAc$
$S \to dc$
$S \to bda$
$A \to d$



$S \to CC$
$C \to cC$
$C \to d$

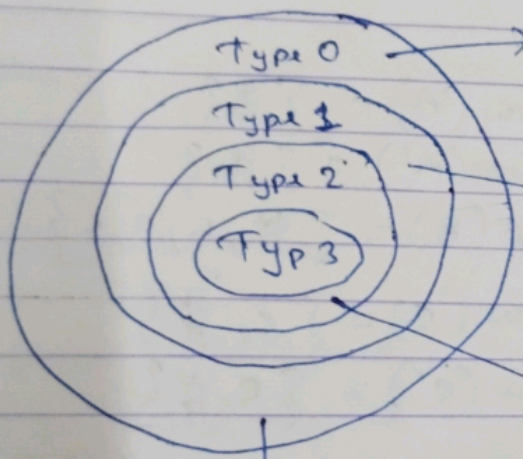


(a) $S \to Aa$
$S \to bAc$
$S \to Bc$
$S \to bBa$
$A \to d$
$B \to d$



$C \rightarrow cC_1, \$$

Chomsky Hierarchy



Type 0 → Unrestricted Grammar, Recursively Enumerable lang. (Recognised by Turing machine)

Type 1 → Context sensitive Grammar, Context sensitive lang. (Accepted by linear Bound Automata)

Type 2 → Context free Grammar, Context free lang. (Accepted by Push down Automata)

Type 3 → Regular Grammar, Regular lang. (Accepted by finite Automata)

$A \rightarrow AcB \mid cC \mid \epsilon$

$B \rightarrow bB \mid id$

$C \rightarrow CaB \mid BbB \mid B$

$A \rightarrow .A\alpha \mid \beta$

$A \rightarrow \beta A'$

$A' \rightarrow \alpha A' \mid \epsilon$

$A \rightarrow cCA' \mid cA'$

$A' \rightarrow cBA' \mid \epsilon$

$B \rightarrow bB \mid id$

$C \rightarrow BbBC' \mid BC'$

$C' \rightarrow aBC' \mid \epsilon$

$first(A) = \{c, b, id\}$	$follow(A) = \{\$\}$
$first(A') = \{c, \epsilon\}$	$follow(A') = \{\$\}$
$first(B) = \{b, id\}$	$follow(B) = \{\$, b, a, c\}$
$first(C) = \{b, id\}$	$follow(C) = \{c, \$\}$
$first(C') = \{a, \epsilon\}$	$follow(C') = \{c, \$\}$

	id	a	b	c	$
A	$A \rightarrow cA'$		$A \rightarrow cA'$	$A \rightarrow cA'$	
A'				$A' \rightarrow cBA'$	$A' \rightarrow \epsilon$
B	$B \rightarrow id$		$B \rightarrow bB$		
C	$C \rightarrow BbBC'$		$C \rightarrow BC'$		
C'		$C' \rightarrow aBC'$		$C' \rightarrow \epsilon$	$C' \rightarrow \epsilon$