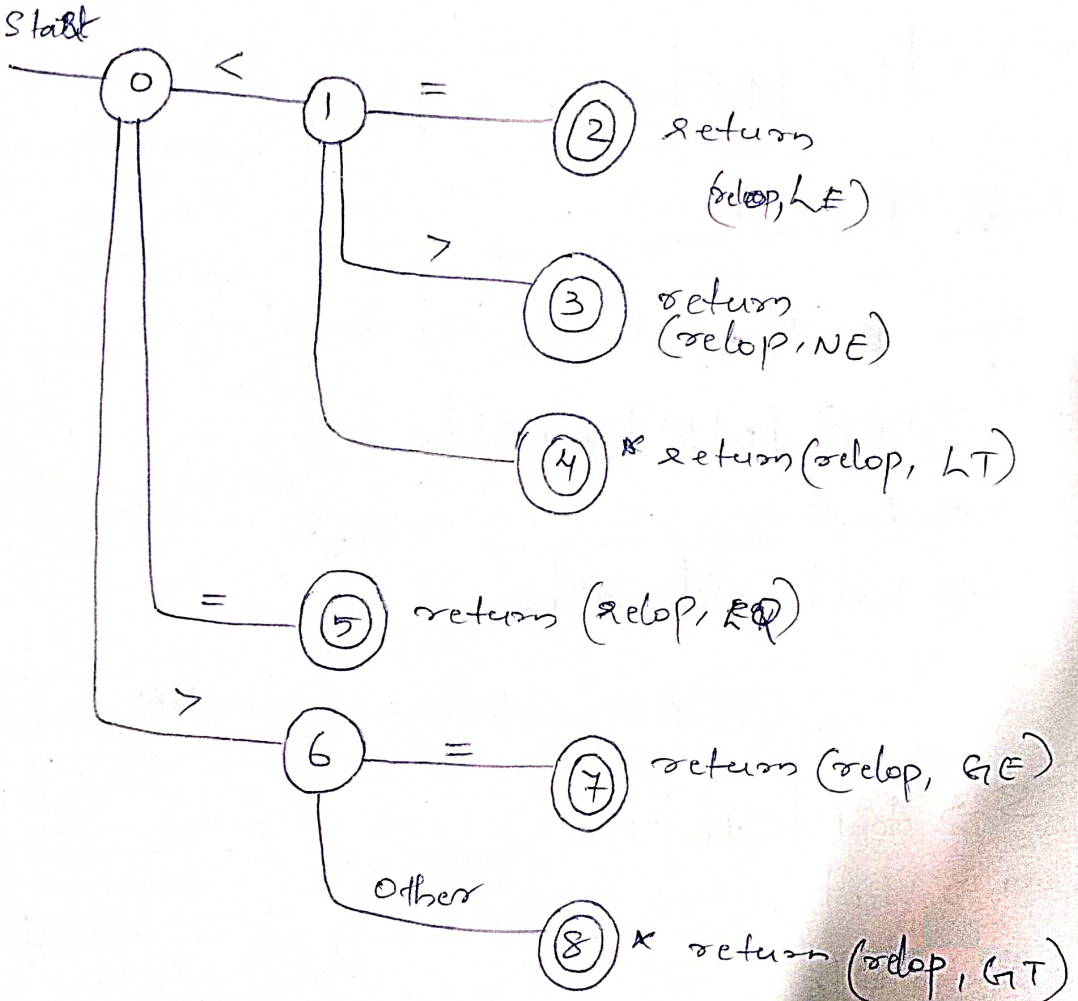
CST 302

COMPILER DESIGN

Part A

1)



5) $A \rightarrow Ba \mid b \mid p \mid q$

$B \rightarrow Bc \mid Ad \mid \in$

$B \rightarrow \underset{\alpha_1}{Bc} \mid \underset{\alpha_2}{Bad} \mid \underset{\beta_1}{bd} \mid \underset{\beta_2}{pd} \mid \underset{\beta_3}{qd} \mid \in$

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

$A \rightarrow \alpha A'$

$A' \rightarrow \beta A' \mid \in$

$B \rightarrow BcB' \mid adB'$

$B' \rightarrow bdB' \mid pdB' \mid qdB' \mid \epsilon$

$A \rightarrow Ba \mid b \mid p \mid q$

$A \rightarrow \underbrace{Ada}_{\alpha} \mid \underbrace{b}_{\beta_1} \mid \underbrace{p}_{\beta_2} \mid \underbrace{q}_{\beta_3}$

$A \rightarrow daA'$

$A' \rightarrow bA' \mid pA' \mid qA' \mid \epsilon$

So final productions:-

$B \rightarrow BcB' \mid adB'$

$B' \rightarrow bdB' \mid pdB' \mid qdB' \mid \epsilon$

$A \rightarrow daA'$

$A' \rightarrow bA' \mid pA' \mid qA' \mid \epsilon$

4)

$S \rightarrow ACB \mid Cbb \mid Ba$

$A \rightarrow da \mid BC$

$B \rightarrow g \mid \epsilon$

$C \rightarrow h \mid \epsilon$

$First(S) = First(A)$

$First(A) = \{d, First(B)\} \Rightarrow A \rightarrow da \mid BC$

$First(B) = \{g, \epsilon\} \Rightarrow A \rightarrow da \mid C$

$First(C) = \{h, \epsilon\} \Rightarrow A \rightarrow da \mid \epsilon$

$\therefore First(S) = \{d, g, h, \epsilon\}$

First(A) = { d, g, g, ϵ

First(A) = { d, First(B) }

= { d, g, h, ϵ }

First(B) = { g, ϵ }

First(C) = { h, ϵ }

Follow

Follow(S) = { $ }

Follow(A) = First(C)

= { h, ϵ } ⟹ S → AB | Cbb | Ba

⟹ S → ACB | Cbb | Ba

= First(B)

= { g, ϵ } ⟹ S → A | Cbb | Ba

Follow(A) = Follow(S)

= { $ }

ie, Follow(A) = { h, g, $ }

Follow(B) = Follow(S)

= { $ }

S → ACB | Cbb | Ba

Follow(B) = First(a)

= { a }

Follow(B) = First(C)

= { h, ϵ } ⟹ A → da | B

A → da | BC

Follow(B) = Follow(A)

= { h, g, $ }

∴ Follow(B) = { $, a, h, g }

Follow(c) = First(B) $\quad S \to ACB \mid Cbb \mid Ba$

$= \{g, \epsilon\} \Rightarrow S \to Ac \mid Cbb \mid Ba$

Follow(c) = Follow(S)

$= \{\$\}$

Follow(c) = First(b) $\quad S \to ACB \mid Cbb \mid Ba$

$= \{b\}$

Follow(c) = Follow(A) $\quad A \to da \mid BC$

$= \{b, g, \$\}$

$\therefore$ Follow(c) = $\{g, \$, b, b\}$

6) $S \to a \mid ab \mid abc \mid abcd \mid e \mid f$

$A \to \alpha\beta_1 \mid \alpha\beta_2 \mid \alpha\beta_3 \mid \ldots \mid \gamma_1 \mid \gamma_2$

$A \to \alpha A' \mid \gamma$

$A' \to \beta_1 \mid \beta_2 \mid \beta_3 \ldots \mid \beta_n$

$S \to \underset{\alpha}{a} \mid \underset{\beta_1}{ab} \mid \underset{\beta_2}{abc} \mid \underset{\beta_2}{abcd} \mid \underset{\gamma_1}{e} \mid \underset{\gamma_2}{f} \qquad \alpha = a$

$\therefore \quad S \to aS' \mid e \mid f$

$S' \to \epsilon \mid b \mid bc \mid bcd$

again, $S \to \underset{\gamma}{\epsilon} \mid \underset{\alpha}{b} \mid \underset{\beta_1}{bc} \mid \underset{\beta_2}{bcd} \qquad \alpha = b$

$\therefore \quad S' \to bS'' \mid \epsilon$

$S'' \to \epsilon \mid c \mid cd$

again, $S'' \rightarrow \underset{\gamma}{\epsilon} \mid \underset{\alpha}{c} \mid \underset{\beta 1}{cd}$ $\quad \alpha = c$

$\therefore S'' \rightarrow cS''' \mid \epsilon$

$S''' \rightarrow \epsilon \mid d$

$\therefore$ Final productions:-

$S \rightarrow aS' \mid e \mid f$

$S' \rightarrow bS'' \mid \epsilon$

$S'' \rightarrow cS''' \mid \epsilon$

$S''' \rightarrow \epsilon \mid d$

3) • Symbol tables are data structures that are used by compilers to hold information about source program constructs.

• It is used to store information about the occurrence of various entities such as, objects, classes, variable names, functions etc,

• It is used by both analysis phase and synthesis phase.

Symbol table used for following purposes:-

→ It is used to store the name of all entities in a structured form at one place.

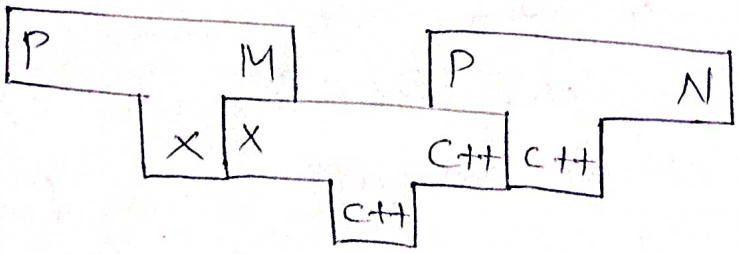
→ It is used to verify if a variable has been declared.

→ It is used to determine the scope of a name.

→ It is used to implement type checking by verifying assignments and expressions in the source code are semantically correct.

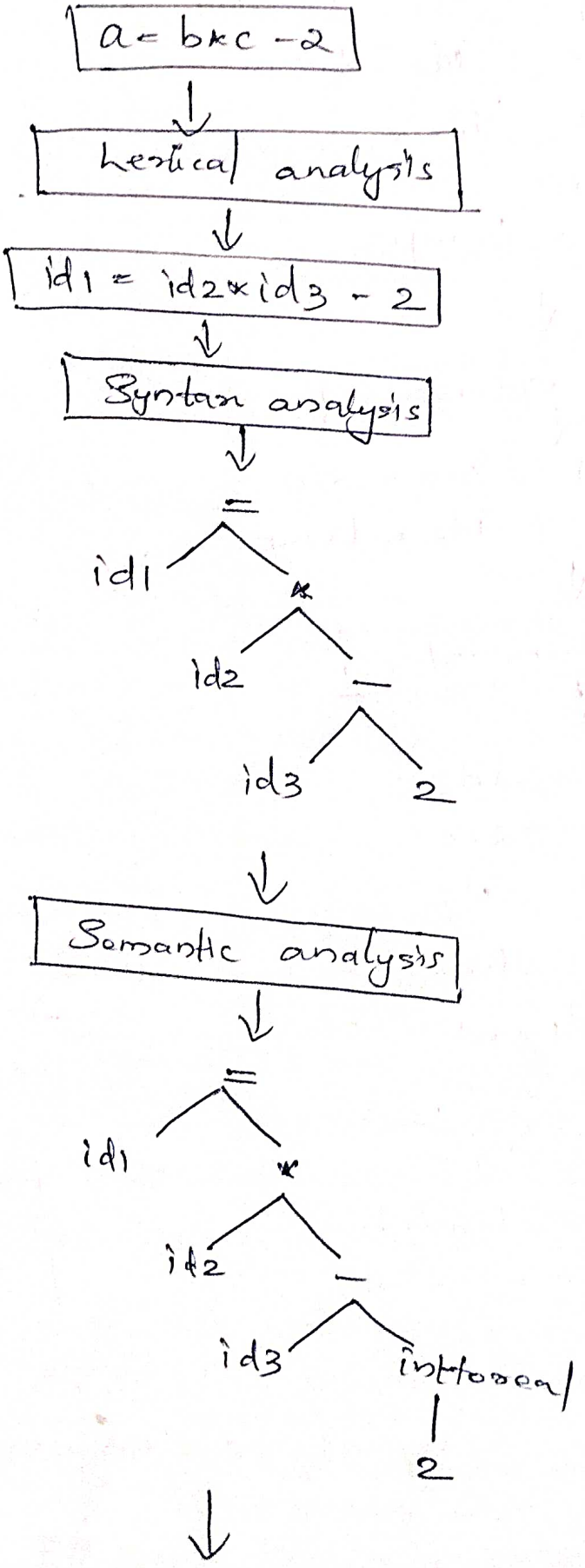
- A symbol table can either be linear or a hash table.
- It maintains the entry for each name as, <symbol name, type, attribute>
  eg: <static, int, salary>
- Use of symbol table:-
  → Symbol table information is used by the analysis and synthesis phases.
  → To generate intermediate or target code.
  → To verify that used identifiers have been defined (declared).
  → To verify that expressions and assignments are semantically correct - type checking.

2)



Part B

7) (a) a - b * c - 2



↓

Intermediate code generation

↓

```
temp1 = inttoreal(2)
temp2 = id3 - temp1
temp3 = id2 * temp2
~~temp4 =~~
id1 = temp3
```

↓

Code optimizer

↓

```
temp1 = id3 - 2.0
id1   = id2 * temp1
```

↓

Code Generator

↓

```
MOV   R2 , id3
SUB   R2, 2.0
MOV   R1 , id2
MUL   R1, R2
MOV   id1 , R1
```

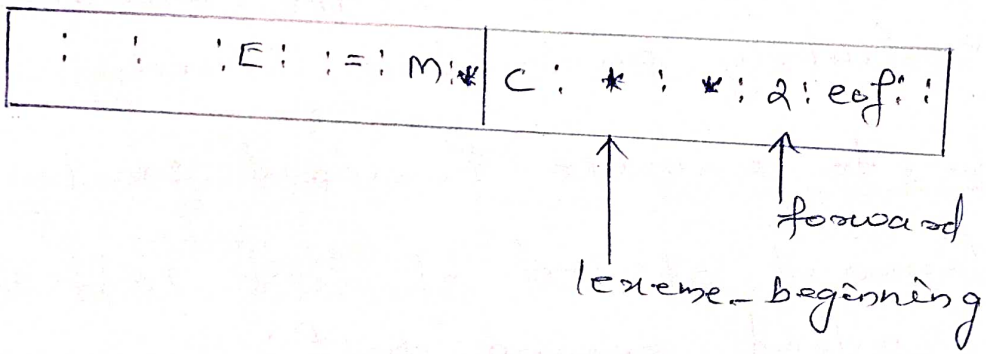
7) (b) Input Buffering :-

1. lexeme_beginning point
2. forward point

- A block of characters to be read into the buffer using 1 system call.

- Scanning of source code in compilers can be speeded up using input buffering by using a two buffer scheme to handle large look-aheads safely.
- Buffer is divided into two N-character halves where N is the number of characters on one disk block.
- Read N input characters into each half of the buffer with one system read command.

eg:



- If fewer than N characters remain in the input then a special character eof marks the end of the source file.
- Two pointers to the input buffer are maintained.
- String of characters between the two pointers is the current lexeme.
- Initially both pointers point to the first character of the next lexeme to be found.
- One called the forward pointer scans ahead until a match for a pattern is found.
- Once the next lexeme is determined the forward pointer is set to the character

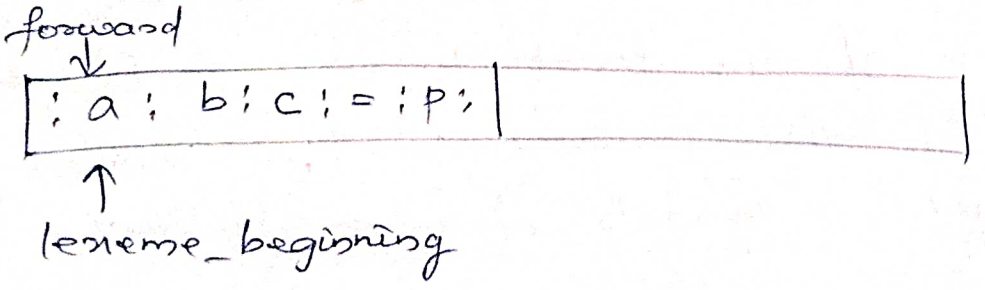
as its right end.

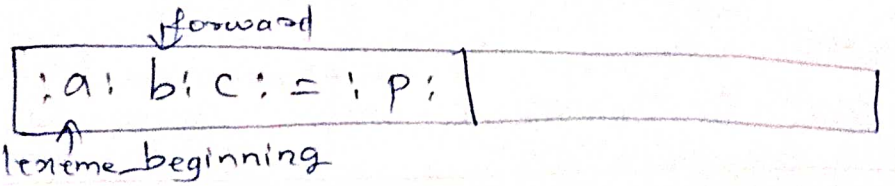
- If the forward pointer is about to move past the halfway mark, right half is filled with N new input characters.
- If the forward pointer is about to move past the right end of the buffer, the left half is filled with N new characters and the forward pointer wraps around to the beginning of the buffer.
- After the lexeme is processed, both pointers are set to the character immediately past the lexeme.

Code to advance Forward pointer :-

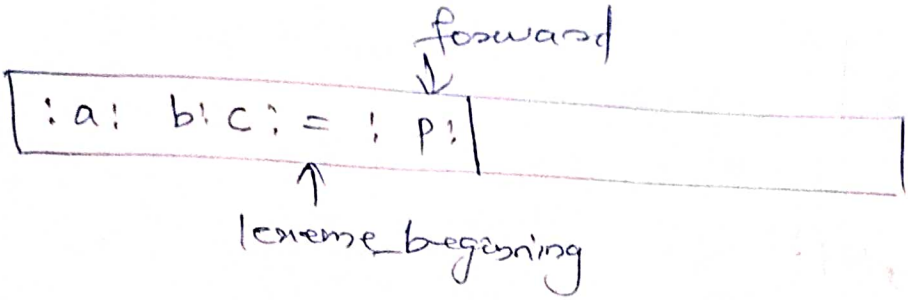
```
if forward at end of first half then begin
        reload second half;
        forward := forward + 1
end
else if forward at end of second half then begin
        reload first half;
        move forward to beginning of first half
end
else forward := forward + 1;
```

eg: abc = pqr * xyz

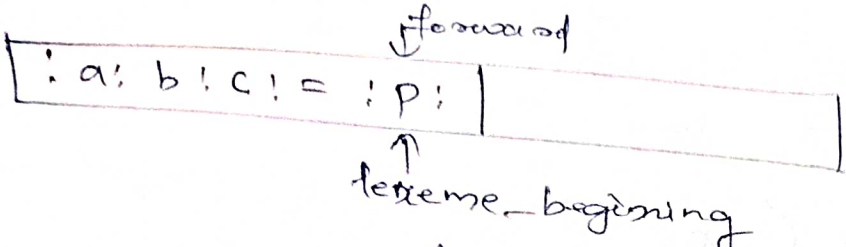


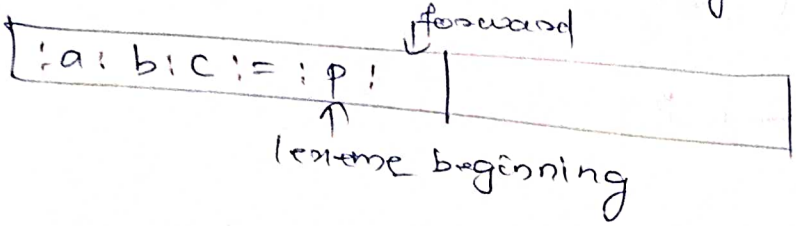


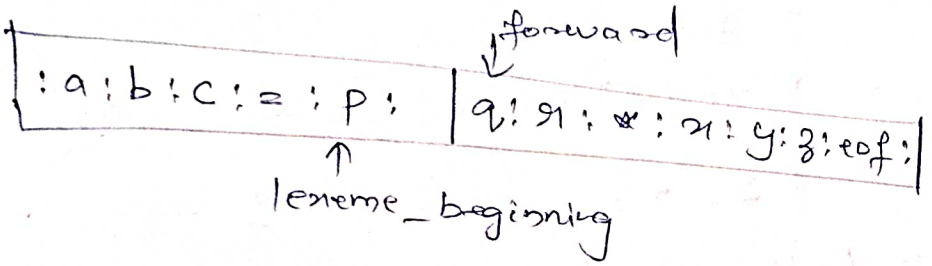
abc => Identifier



= => assignment operator

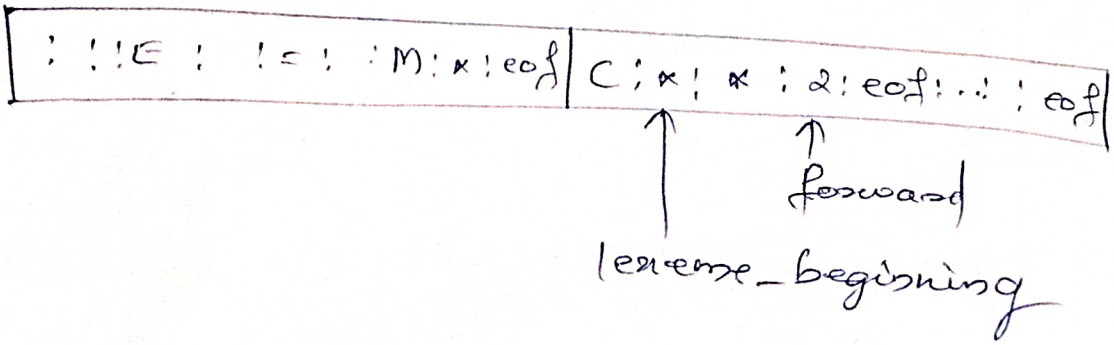






Reload 2nd half

- Except at the end of the buffer halves, the above code requires 2 tests for each advance of the forward pointer.
- We can reduce the two tests to one if we extend each buffer half to hold a sentinel character at the end.
- Sentinel is a special character that cannot be part of the source program (eof).



8) (a) $S \to (L) \mid a$

$L \to SL'$

$L' \to \varepsilon \mid ,SL'$

(i) First

First(S) = { (, a }

First(L) = { (, a }

First(L') = { ε, , }

Follow

Follow(S) = { $ }

= First(L') $L \to SL'$

= { ε, , }

$L \to S$

Follow(S) = Follow(L)

Follow(L) = First( ))

= {) }

Follow(S) = { $, , ,) }

Follow(L') = Follow(L) $L \to SL'$

= {) }

Follow(L') = {) }

	(	)	a	,	$
S	S→(L)		S→a		
L	L→SL'		L→SL'		
L'		L'→ε		L'→,SL'	

(ii) The given grammar is an LL(1) parser.

8(b)

After removing left recursion

$S \to aAC \mid bB$

$A \to gA'$

$A' \to bcA' \mid bdA' \mid \epsilon$

$B \to f \mid g$

$C \to h \mid i$

$A' \to bcA' \mid bdA' \mid \epsilon$

$A' \to bA'A$

$A \to c \mid d$

$S \to aAC \mid bB$

$A \to gA' \mid c \mid d$

$A' \to bA'A$

$B \to f \mid g$

$C \to h \mid i$

$A \to \beta A'$

$A' \to \alpha A' \mid \epsilon$

$A \to Abc$

$A \to gA'$

$A' \to bcA' \mid \epsilon$

$A \to Abd \mid g$

$A = A, \alpha = bd$

$\beta = g$

$A \to gA'$

$A' \to bdA'$

First(S) = {a, b}

First(A) = {g, c, d}

First(A') = {b}

First(B) = {f, g}

First(C) = {h, i}

Follow(S) = {$}

Follow(A) = First{C} = {h, i}

Follow(A') = Follow(A) = {h, i}

Follow(B) = Follow(S) = {$}

Follow(C) = Follow(S) = {$}

LL1 Parsing Table

	a	b	c	d	f	g	h	i	$
S	S→aA	S→bB							
A			A→c	A→ε		A→gA'			
A'		A'→εA'							
B					B→f	B→g			
C						C→h	C→i		