Top - down Parser

Recursive Descent Parser (with Back-tracking)

LL(1) (without Back-Tracking)

(i) Recursive Descent Parser :-

```
S()
{
   select any production of S
       let( S → x1 x2 x3 .... xk)

   for( i=1 ; i≤k; i++)
   {
      if( xi is variable)
         xi();

      else
        if( xi == Look Ahead symbol)
                 incr i/p pointer;

      else
         error (try other possibilities)
                                Next production
   }
}
```

Eg:- S → ABC | DEF | GHI

A → ab | gh | mn

B → cd | ij | op

C → ef | kl | qr

D → d

E → e

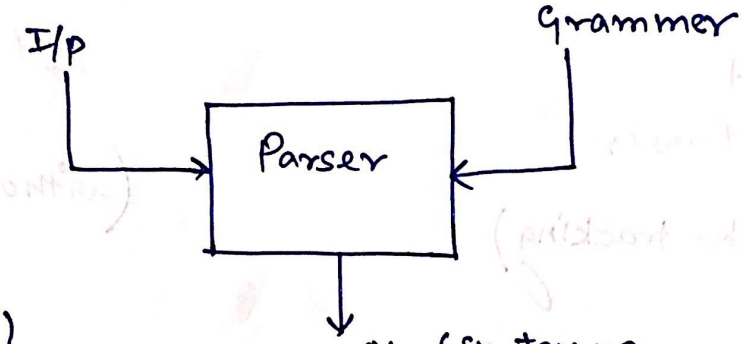
F → f

G → g

H → h

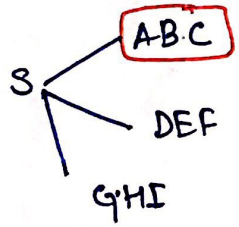
I → i

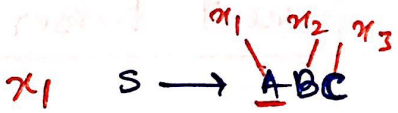
I/p:- abcdqr $



PUSH , S()

S():



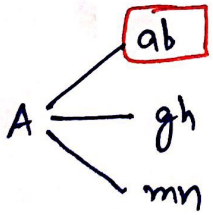


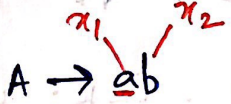
A is non-terminal

so A();

PUSH ,,-A()

A():





x_1: a is terminal,

if(a == LAS)

a b c d q r $

↑

LAS

so inc LAS;

x_2: b is terminal

& (b == LAS)

a b c d q r $

↑

LAS

~~a~~ ~~b~~ c d q r $ INC LAS;
↑

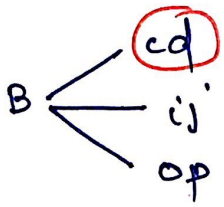
POP A();

- S() : r_2 : B is non-terminal

so, B();

PUSH B()

B() :

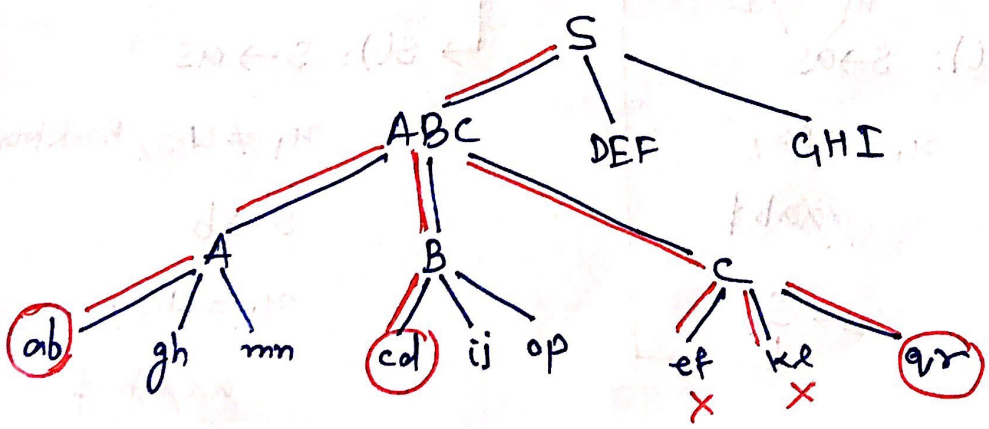


r_2 : B → c d (r_1, r_2)

r_1 : c is terminal

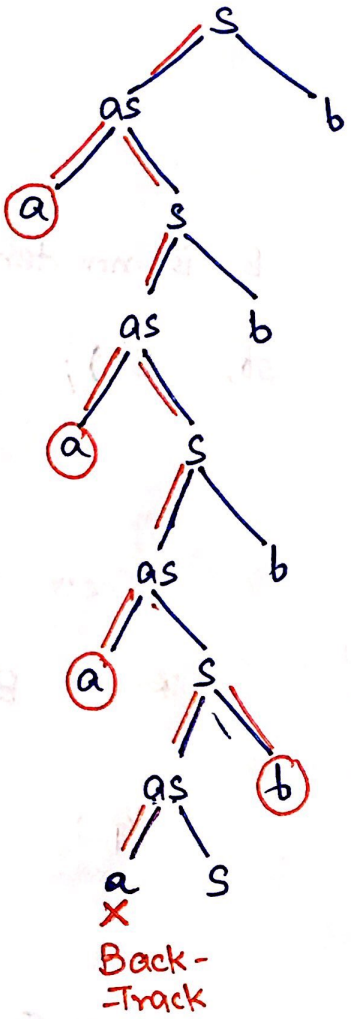
~~a~~ ~~b~~ c d q r $ c == LAS
↑
LAS

~~a~~ ~~b~~ ~~c~~ d q r $ INC LAS;
↑
LAS



X: Error (Backtrack)

eg:- $S \rightarrow aS \mid b$ I/P:- aaab$



Back-Track

aaab$

S(): $S \rightarrow aS$ (α_1 / α_2)

α_1 = LAS, aaab$

α_2 = S

S(): $S \rightarrow aS$ (α_1 / α_2)

α_1 = LAS, aab$

α_2 = S

S(): $S \rightarrow aS$

α_1 = LAS, ab$

α_2 = S

S(): $S \rightarrow aS$

$\alpha_1 \neq$ LAS, Backtrack

$S \rightarrow b$

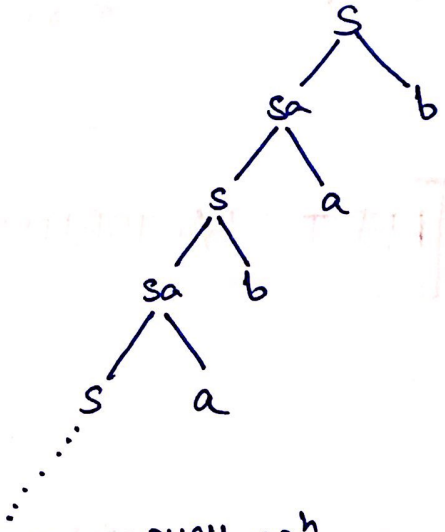
α_1 = LAS, aaab$

ACCEPTED

eg: $S \rightarrow Sa \mid b$

I/p: baaa$

[LEFT RECURSION]



As we can see in

$S \rightarrow Sa$

first symbol in production is non-terminal which makes parser to call S() infinitely until STACK overflow.

And as we know $S \rightarrow Sa$ is Left recursion

so if Left recursion is present in then Recursive descent parser goes into a ∞ loop.

NOTE:-

In parser STACK is used and in LA (lexical analyser) DFA is used for tokenization so,

(DFA + STACK = PDA)

so Parsing use PDA

Q: S → ABC | DEF | GHI

A → ab | abc | abcd

B → ef | gh | ij

C → kl | mn | op

D → d

E → e

F → f

G → g

H → h

I → i

[LEFT FACTORING]

I/P:- abcdghop $

S():

S → ABC S → DEF S → GHI

A():

A → ~~ab~~

A → abc

A → abcd

if this production is selected then no problem

B():

B → ef ✗

Backtrack

B → gh ✗

Backtrack

B → ij ✗

Backtrack

As A → ab

Doesn't show any error because it has same symbol as A → abcd

But 'cd' can only be drive from A only so A done its halt work, due to which all other showing error.

NOTE:-

- if Grammer contain Left recursion → Recursive - Descent parser goes into ∞ loop (can't give answer)

- if Grammer contains Left factoring → Recursive Descent parser may failed as it can show false result (string is diriviable from grammar but parser can't)

(ii) LL(1) Parser / Predictive parser :-

- LL(1) is without backtracking because we are using LL(1) parsing table.
- To construct LL(1) parsing table, we need two functions :
  - ↳ FIRST()
  - ↳ FOLLOW()

FIRST(): contains set of all terminals present at 1st position of every string derived by A.

eg: $S \rightarrow abc \mid def \mid ghi \mid \epsilon$

FIRST(S) = Fi(abc) , Fi(def) , Fi(ghi) , Fi(ϵ)

(S: variable)

= Fi(a) , Fi(d) , Fi(g) , ϵ

= {a, d, g, ϵ}

First(terminal) ⟶ terminal

First(null) ⟶ null (ϵ)

eg:

S → ABC

A → a | d | ϵ

B → b | g | ϵ

C → c | j | ϵ

First(S) = Fi(ABC)

= Fi(A)

= Fi(a) , Fi(d) , Fi(ϵ) — next of A if exist

= " , " , Fi(B)

= Fi(b) , Fi(g) , Fi(ϵ) — next of B if exist

= _____ , Fi(C)

= Fi(c) , Fi(j) , Fi(ϵ) — not next of C so include

= Fi(a), Fi(d), Fi(b), Fi(g), Fi(c), Fi(j), Fi(ϵ)

= {a, d, b, g, c, j, ϵ}

eg:-

$S \rightarrow aBDh$

$B \rightarrow bC \mid \epsilon$

$C \rightarrow cC \mid \epsilon$

$D \rightarrow FE$

$F \rightarrow g \mid \epsilon$

$E \rightarrow f \mid \epsilon$

$\mathcal{U}$	First($\mathcal{U}$)
S	a
B	b, ϵ
C	c, ϵ
D	g, f, ϵ
F	g, ϵ
E	f, ϵ

First(S) = First(aBDh)

= Fi(a)

= {a}

First(B) = fi(bC) , fi(ϵ)

= Fi(b), Fi(ϵ)

= {b, ϵ}

First(C) = Fi(cC) , Fi(ϵ)

= Fi(c) , Fi(ϵ)

= {c, ϵ}

First(D) = Fi(FE)

= Fi(F)

= Fi(g) , Fi(ϵ)

=

Fi(E)

Fi(f) , Fi(ϵ)

= {g, f, ϵ}

First(F) = {g, ϵ}

First(E) = {f, ϵ}

eg:- $S \rightarrow aSbS \mid bSaS \mid \epsilon$

$First(S) = Fi(aSbS) , Fi(bSaS) , Fi(\epsilon)$

$= \{a, b, \epsilon\}$

eg:- $S \rightarrow AaAb \mid BbBa$

$A \rightarrow \epsilon$

$B \rightarrow \epsilon$

$First(S) = Fi(AaAb) , Fi(BbBa)$

$= Fi(A) , Fi(B)$

$Fi(A)$ → $Fi(\epsilon)$ next → $Fi(aAb)$ → $Fi(a)$

$Fi(B)$ → $Fi(\epsilon)$ next → $Fi(bBa)$ → $Fi(b)$

$= \{a, b\}$

x	$First(x)$
S	a, b
A	ϵ
B	ϵ

FOLLOW(:): contain set of all terminals present at Ist place of every string immediately right of that variable.

eg:-

$S \rightarrow ABC$

$A \rightarrow DEF$

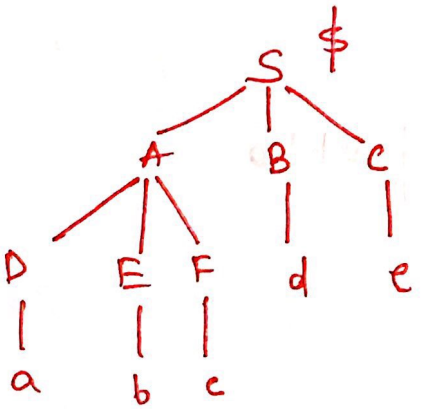
$D \rightarrow a$

$E \rightarrow b$

$B \rightarrow d$

$C \rightarrow e$

$F \rightarrow c$



Follow(D) = First(EF)

$A \rightarrow DEF$

= Fi(E)

= b

Follow(F) = Follow(A)

$A \rightarrow DEF$

it var is last in production

= Fi(BC)

= Fi(B)

= d

Follow(C) = Follow(S)

$S \rightarrow ABC$

C is last var in production

= First($)

= $

eg:

$S \rightarrow ABC$

$A \rightarrow DEF$

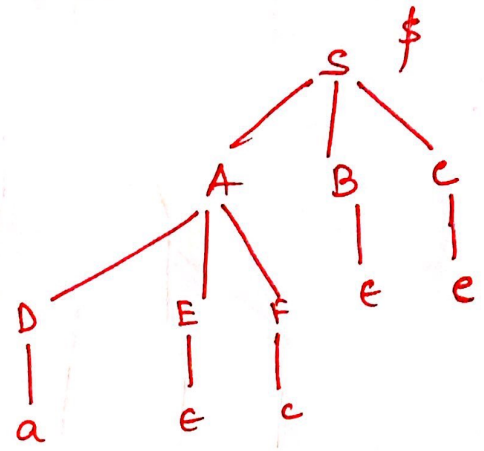
$D \rightarrow a$

$B \rightarrow \epsilon$

$E \rightarrow \epsilon$

$C \rightarrow e$

$F \rightarrow c$



Follow (D) = First (EF)

= First(E)

↓

Fi(ε) next Fi(F)

↓

Fi(c) = c

Follow (F) = Follow (A)

= First (BC)

= Fi(B)

↓

Fi(ε) next Fi(C)

↓

Fi(e) = e

Follow (C) = Follow(S)

= First($)

= $

eg:-

$S \rightarrow ABC$

$A \rightarrow DEF$

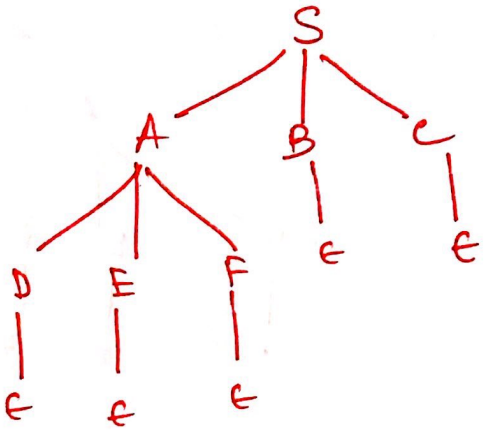
$B \rightarrow \epsilon$

$C \rightarrow \epsilon$

$D \rightarrow \epsilon$

$E \rightarrow \epsilon$

$F \rightarrow \epsilon$



Follow (B) = First(C)

↓

Fi(ϵ) next Follow(S)

↓

Fi($) = ($)

Follow (D) = First (EF)

= Fi(E)

↓

Fi(ϵ) next Fi(F)

↓

Fi(ϵ) next Follow(A)

↓

Fi(BC)

↓

Fi(B)

↓

Fi(ϵ) next Fi(C)

↓

Fi(ϵ)

↓

Follow(S)

=($)

NOTE:- " Follow can't contain ϵ "

Rules to find follow:-

Assume A is variable

i) If A is start symbol Follow(A) = "$"

ii) $B \rightarrow \epsilon A(D\gamma)$

$D \rightarrow b$

Follow(A) = Fi(Dγ)

= Fi(D) = Fi(b) = b

iii) $B \rightarrow cA$ or $B \rightarrow cAD$

$D \rightarrow \epsilon$

Follow(A) = Follow(B)

Q. $E \rightarrow TE'$

$E' \rightarrow \epsilon \mid +TE'$

$T \rightarrow FT'$

$T' \rightarrow \epsilon \mid *FT'$

$F \rightarrow id \mid (E)$

x	First(x)	Follow(x)
E	id, (	$,)
E'	ϵ, +	$,)
T	id, (	$, +,)
T'	ϵ, *	$, +,)
F	id, (	$, *, +,)

First(E) = First(TE')

= Fi(T)

= Fi(FT')

= Fi(F)

= Fi(id), Fi((E))

= {id, (}

Q. $S \rightarrow asbs \mid bsas \mid \epsilon$

Find First and Follow

$First(S) = Fi(asbs), Fi(bsas), Fi(\epsilon)$

$= \{a, b, \epsilon\}$

$Follow(S) = \{\$, b, a\}$

S is starting symbol $= \{\$, \}$

$S \rightarrow asbS = \{\$, b, \}$

$S \rightarrow asbS$ no use

$S \rightarrow bSaS = \{\$, b, a\}$

$S \rightarrow bsaS$ no use

Q. $S \rightarrow AbAb \mid BaBa$

$A \rightarrow \epsilon$

$B \rightarrow \epsilon$

$First(S) = Fi(AbAb), Fi(BaBa)$

$= \{b, a\}$

Follow(A) = {b} $S \rightarrow \underline{A}b\underline{A}b$

Follow(B) = {a} $S \rightarrow \underline{B}a\underline{B}a$

x	First(x)	Follow(x)
S	{a, b}	{$}
A	ϵ	{b}
B	ϵ	{a}

Q. $S \rightarrow (L) \mid a$

$L \rightarrow SL'$

$L' \rightarrow \epsilon \mid ;SL'$

x	First	Follow
S	(, a	$, ;,)
L	(, a	)
L'	ϵ, ;	)

First(S) = Fi((L)), Fi(a)

= {(, a}

First(L) = First(SL') = Fi(S)

First(L') = Fi(ϵ), Fi(;SL')

= {ϵ, ;}

Follow(L) =)

Follow(L') =)

Q. $S \rightarrow aBDh$

$B \rightarrow bB \mid \epsilon$

$C \rightarrow cC \mid \epsilon$

$D \rightarrow FE$

$E \rightarrow f \mid \epsilon$

$F \rightarrow g \mid \epsilon$

x	First	Follow
S	a	$
B	b, ϵ	f, g, h
C	c, ϵ	—
D	f, g, ϵ	h
E	f, ϵ	h
F	g, ϵ	f, h

C is not reachable (useless symbol)

LL(1) - parsing table construction algo:-

For each production $A \rightarrow \alpha$

Repeat following steps :

(i) ADD $A \rightarrow \alpha$ under $M[A, b]$

$\forall b \in First(\alpha)$

(ii) if $First(\alpha)$ contain ϵ then add

$A \rightarrow \alpha$ under $M[A, c]$, $\forall c \in Follow(A)$

Parsing table

- no. of rows = no. of variable
- no. of columns = no. of terminal + 1 ($)

eg:-

$S \rightarrow (L) \mid a$

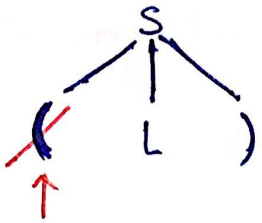
$L \rightarrow SL'$

$L' \rightarrow \epsilon \mid ,SL'$

LL(1)	,	a	(	)	$
S		$S \rightarrow a$	$S \rightarrow (L)$		
L		$L \rightarrow SL'$	$L \rightarrow SL'$		
L'	$L' \rightarrow ,SL'$			$L' \rightarrow \epsilon$	

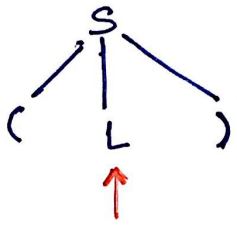
α	First (α)	Follow(α)
S	(, a	$,)
L	(, a	)
L'	ϵ, ,	)

I/p: (a, a, a) $

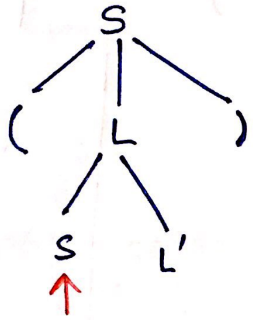


M[S, (]

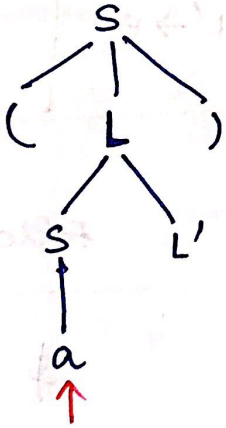
(a, a, a) $



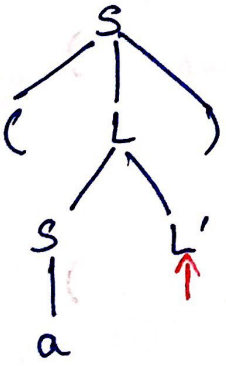
M[L, a]



M[S, a]

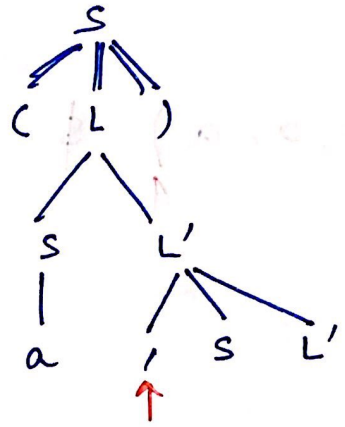


(a, a, a) $

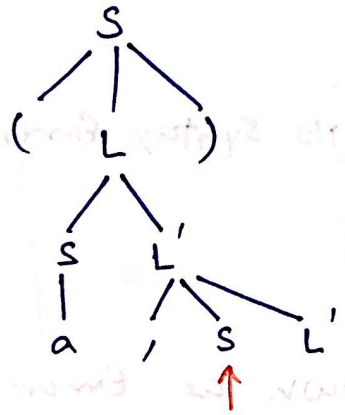


(a, a, a) $

M[L', ,]

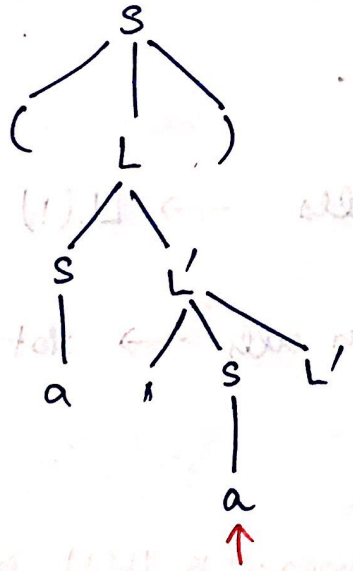


(a, a, a) $

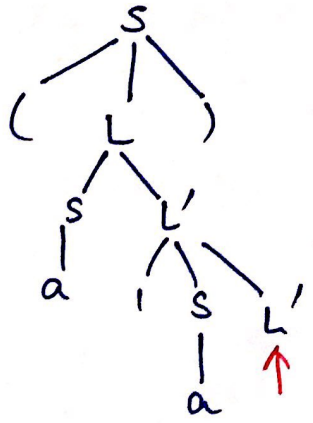


(a, a, a) $

M[S, a]

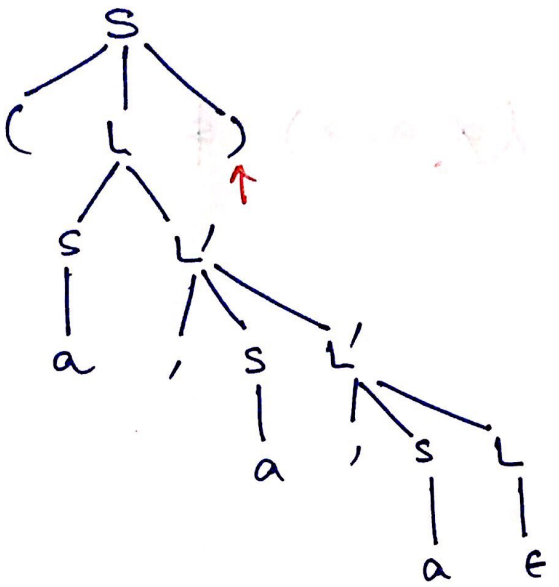


(a, a, a) $



(a, a, a) $

M[L', ,]



(a, a, a) $

↑

Hence string accepted [No syntax Error]

NOTE:- - Blank Entries are known as Error Entries

- no entry in LL(1) parsing table should contain more than one productions.

- Single production in cells → LL(1) grammar
- Multiple production in cells → Not LL(1)

Q: Check the following grammer is LL(1) or not

S → AaAb | BbBa

A → ϵ

B → ϵ

M[ ]	a	b	$
S	S → AaAb	S → BbBa	
A	A → ∈	A → ∈	
B	B → ∈	B → ∈	

	First( )	Follow( )
S	a, b	$
A	∈	a, b
B	∈	a, b

(Grammes is LL(1))

NOTE:-

$S \rightarrow X_1 | X_2$ → it can sutter problem in which both $S \rightarrow X_1$ & $S \rightarrow X_2$ goes to same cell

$X_1 \rightarrow a$

$X_2 \rightarrow b$

they have only one option (choice) so they can't make any double entries

Q. Check LL(1) or not:

$S \rightarrow asbs \mid bsas \mid \epsilon$

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

	a	b	$
S	S→asbs S→ϵ	S→bsas S→ϵ	S→ϵ

Multiple Entries

So Not LL(1) grammer

Q. $E \rightarrow TE'$

$E' \rightarrow \epsilon \mid *TE'$

$T \rightarrow FT'$

$T' \rightarrow \epsilon \mid +FT'$

$F \rightarrow id \mid (E)$

M[]	id	+	*	$	(	)
E	E→TE'				E→TE'	
E'			E'→*TE'	E'→ϵ		E'→ϵ
T	T→FT'				T→FT'	
T'		T'→+FT'	T'→*	T'→ϵ		T'→ϵ
F	F→id				F→(E)	

	first()	Follow()
E	id, (	$,)
E'	∈, *	$,)
T	id, (	$,), *
T'	∈, +	$,), *
F	id, (	$,), *, +

Q. $S \rightarrow aBDh$

$B \rightarrow bC \mid \in$

$C \rightarrow cB \mid \in$

$D \rightarrow FE$

$E \rightarrow cB \mid \in$

$F \rightarrow bC \mid \in$

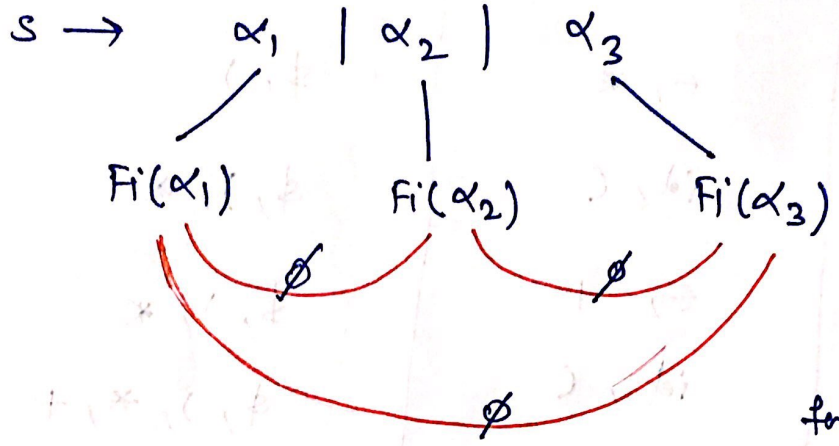
Check for LL(1)

(Not LL(1))

- check only for those variable which contains more than one choices. ie

M[]	a	b	c	h	$
B		B→bC B→∈	B→∈	B→∈	
C		C→∈	C→cB C→∈	C→∈	
E			E→cB	E→h	
F		F→bC	F→∈	F→∈	

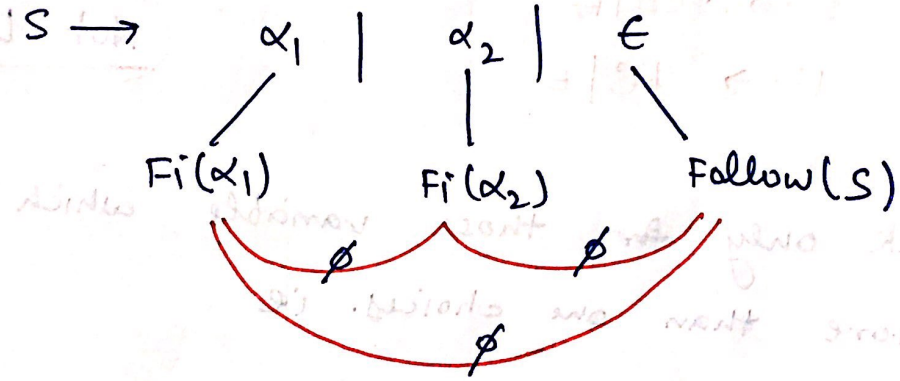
NOTE:- ↳ If productions have no null production —



for LL(1)

Every possible pair of Fi( ) is disjoint i.e they should not have common "first terminals"

↳ If ∈ production is present



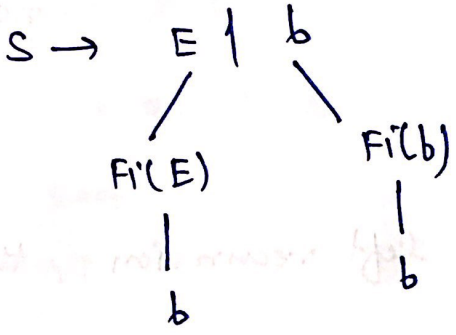
Every possible pair should be disjoint for LL(1)

Q. Check LL(1) or not:

$S \rightarrow E \mid b$

$E \rightarrow b$

Since S can create multiple entries so,



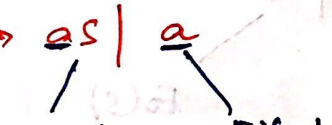
$b \wedge b \neq \emptyset$

not disjoint

so, (Grammer is not LL(1))

Q. Check LL(1) or not:

$S \rightarrow aS \mid a$

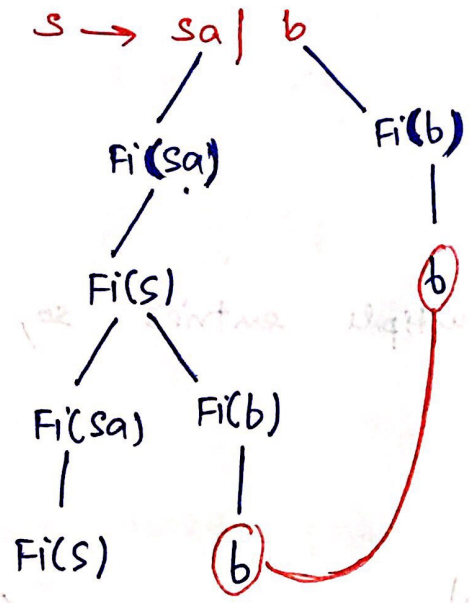


$a \wedge a \neq \emptyset$ Not disjoint

so, if grammer contains Left factoring then it cannot be LL(1)

(Left factoring → not LL(1))

Q. Check LL(1) or not:



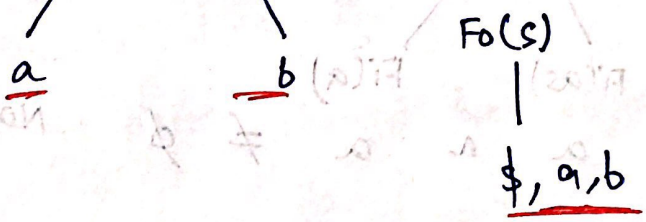
$b \cap b = b \neq \emptyset$

not disjoint

So, if grammer has left recursion, then grammer is not LL(1).

Left recursion ⟶ not LL(1)

Q. $S \rightarrow asbs \mid bsas \mid \epsilon$



not disjoint

Ambiguous grammer ⟶ not LL(1)

NOTE:- Every regular grammer need not be LL(1)

As regular grammer

- may contain Left factoring — $S \to as \mid a$
- may contain Left recursion — $S \to Sa \mid a$
- may be ambiguous — $S \to as \mid a \mid \epsilon$

Q. $S \to ab \mid ac \mid ad$

(a) LL(1) ✗

(b) LL(2) ✓

LL(1)

one symbol is scanned at a time.

$S \to ab \mid ac \mid ad$

Left factoring

not LL(1)

LL(2)

two symbal is scanned at a time

$S \to ab \mid ac \mid ad$

LL(2)

Power: LL(1) $\leq$ LL(2)