CS & IT ENGINEERING

Compiler Design

Lexical Analysis & Syntax Analysis

Lecture No. 6

By- DEVA Sir

01 First & Follow Sets

02 LL(1) Parser

03 LL(1) CFG

04 LL(1) Table

05 LL(1) Algorithm

④ $S \to daT \mid Rf$

$T \to aS \mid baT \mid \varepsilon$

$R \to caTR \mid \varepsilon$

$Fi(S) = \{d, c, f\}$

$Fi(T) = \{a, b, \varepsilon\}$

$Fi(R) = \{c, \varepsilon\}$

$Fo(T) = \{\$, c, f\}$

$Fo(R) = \{f\}$

	a	b	c	d	f	$
S			S→Rf	S→daT	S→Rf	
T	T→aS	T→baT	T→ε		T→ε	T→ε
R			R→caTR		R→ε	

(5)

$S \rightarrow Aa$

$A \rightarrow Bd$

$B \rightarrow b \mid \varepsilon$

$D \rightarrow d \mid \varepsilon$

H.W.

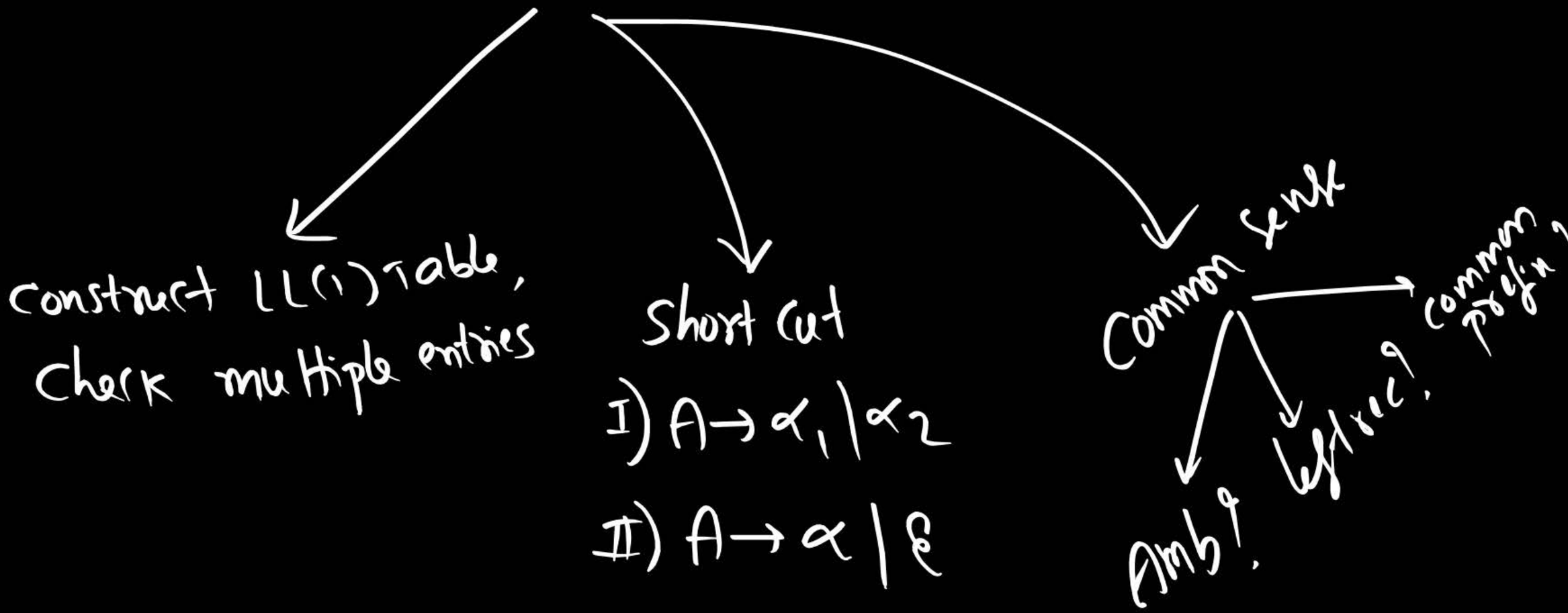
⑥ $S \to aAbB \mid bAaB \mid \varepsilon$

$A \to S$

$B \to S$

H.W.

How to check given CFG is LL(1) ?



① $S \to a$

LL(1) CFG ✓

Single production is always in LL(1)

② $S \to a \mid b$

$\{a\} \cap \{b\} = \phi$

$\Rightarrow$ LL(1) ✓

③ $S \to a \mid ab$ $\Rightarrow$ not LL(1)

$\{a\} \cap \{a\} \neq \phi$

$= \{a\}$

	a
S	$S \to a$ $S \to ab$

④ $S \rightarrow aSa \mid \varepsilon$ ***

$A \rightarrow \alpha \mid \varepsilon$

$\{a\} \cap Fo(S)$

$\{a\} \cap \{\$, a\} \neq \phi$

$Fi(\alpha) \cap Fo(A)$!

not LL(1)

⑤ $S \rightarrow aSa \mid b$ It is LL(1)

⑥ $S \rightarrow Sa \mid b$

$\{b\} \cap \{b\} \neq \phi$

Not LL(1)

	b	a	$
S	$S \rightarrow Sa$ $S \rightarrow b$		

(7) $S \to AB \mid ab$

data ∩ follow ≠ ϕ

$A \to a$ ✓

$B \to b$ ✓

Not LL(1)

	a
S	$S \to AB$ $S \to ab$

8) $S \rightarrow aSb \mid \varepsilon$

It is LL(1)

$\{a\} \cap \{\$, b\} = \phi$

9) $S \rightarrow a \mid bc \mid acd \mid \varepsilon$

$\phi \quad \phi \quad \neq \phi$

not LL(1)

	a
S	S→a S→acd

10) If CFG is LL(1) then

A) It is Unambiguous
B) It is non Left recursive
C) It is Left factored
D) None

11) If CFG is Unambiguous, non left recursive, and Left factored then CFG is ____

$S \rightarrow aSa \mid \varepsilon$

$\{a\} \cap \{\$, a\}$

A) LL(1)
B) Not LL(1)
C) May or may not LL(1)

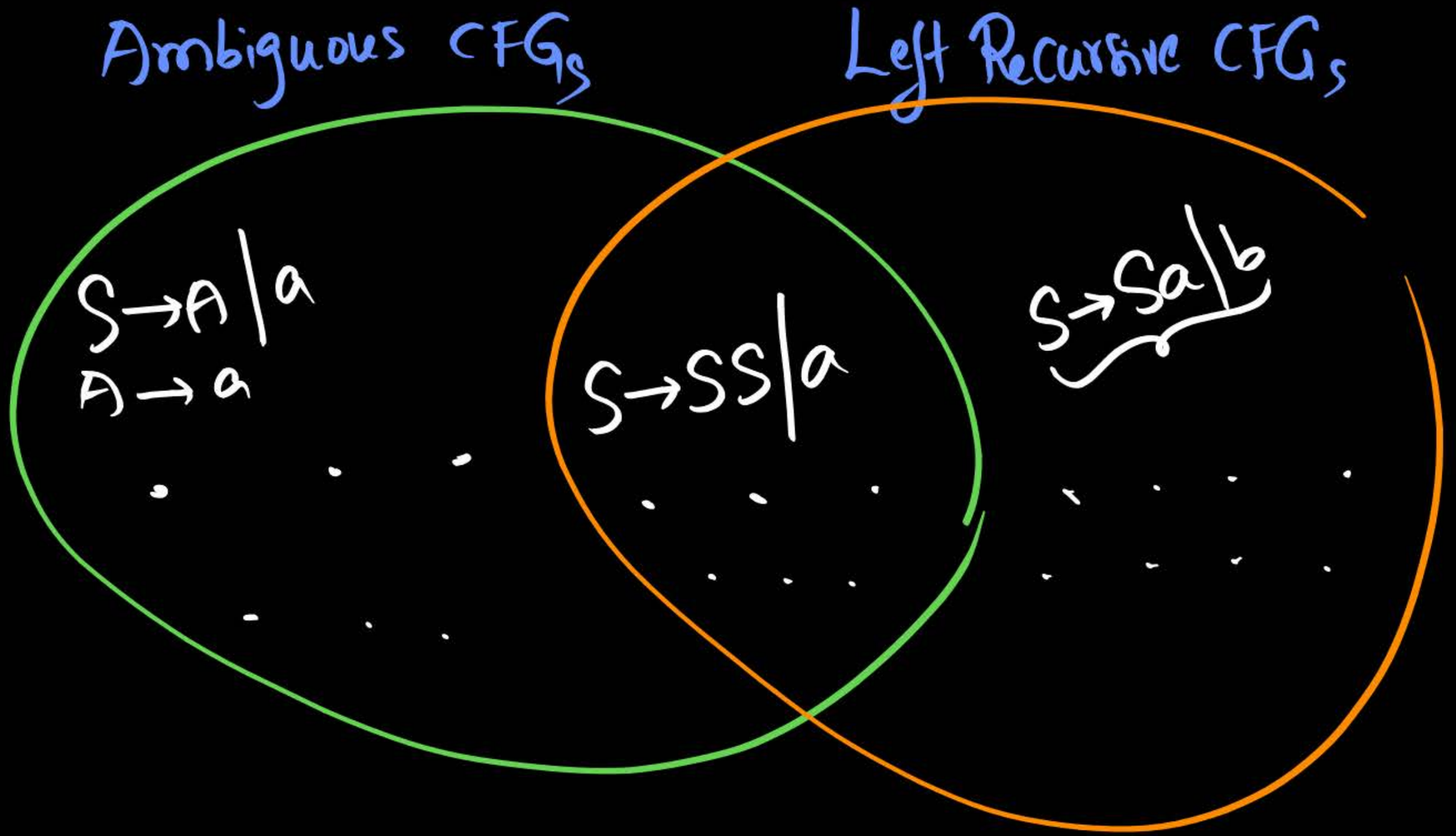
12) If CFG is Unambiguous, non left recursive, Left factored, and free from null rules then CFG is _____

A) LL(1)

B) Not LL(1)

C) Need not be LL(1)

Note : No relation b/w Ambiguous & Left recursion



LL(1) parser [Predictive parser] [TOP-down parser]

Recursive-descent parser

Every production has procedure

$S \rightarrow a \mid b$

```
S( )
{
  if (l == a)
    match(a);
}
```

```
S( )
{
  if (l == b)
    match(b);
}
```

Non-Recursive descent parser (Table Approach)

	a	b	$
S	S→a	S→b	

13) LL(1) parser is ——

A) Top-down parser

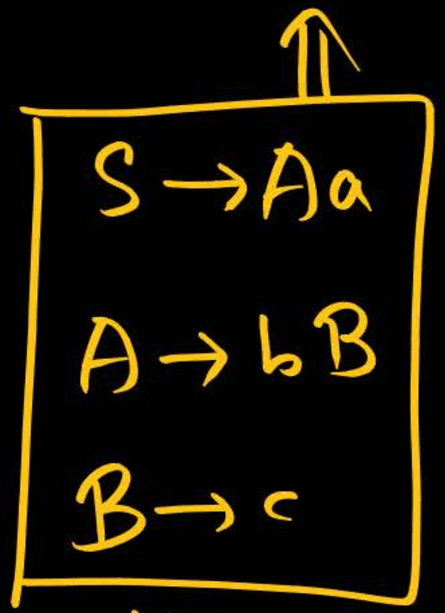
B) Uses LMD

C) predictive parser

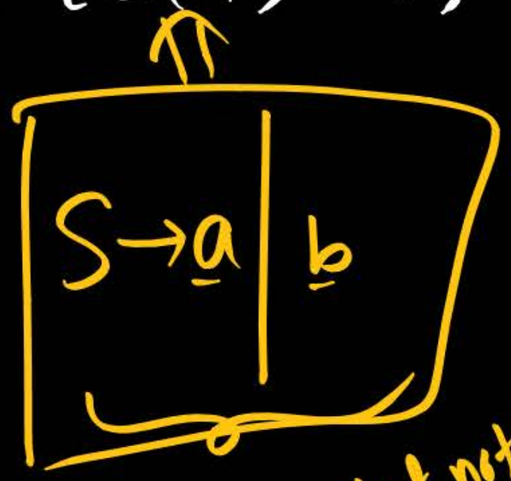
D) Recursive-descent parser

E) Non-recursive descent parser

14) LL(0) Vs LL(1) Vs LL(2) Vs .....



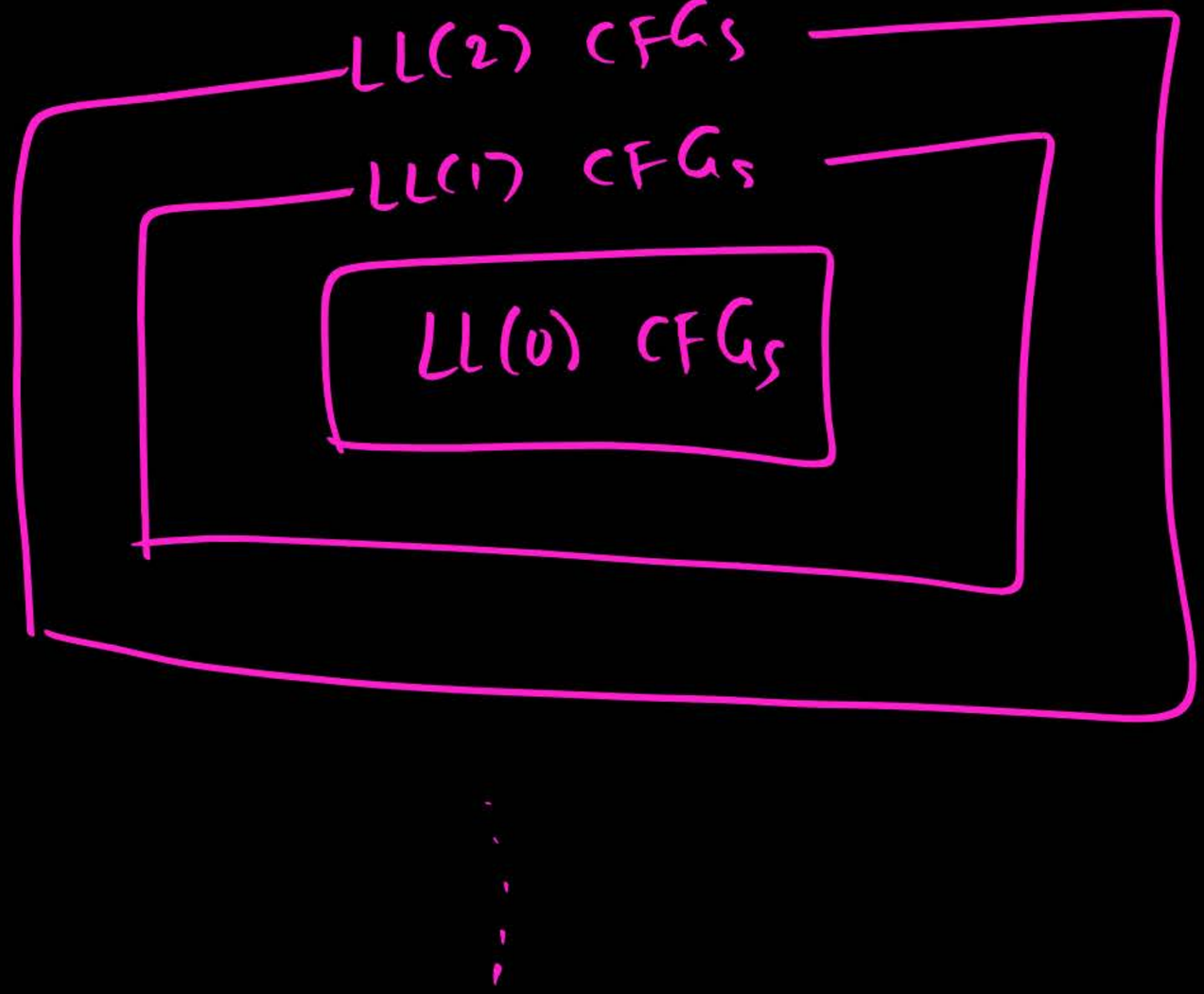
LL(0)
LL(1)
LL(2)
LL(3)
.
.
.



LL(1) but not LL(0)
LL(2)
LL(3)
.
.
.

S → ab | acd

LL(2) but not LL(1)


LL(2) CFGs
LL(1) CFGs
LL(0) CFGs

15) $S \to \varepsilon$ LL(0) ✓ [LL(1), LL(2), ...]

16) $S \to a \mid b$ LL(1) but not LL(0)

17) $S \to a \mid ab$ LL(2) but not LL(1)

18) $S \to a \mid ab \mid abc$ LL(3) but not LL(2)

19) $S \to a \mid ab \mid abc \mid abcd$ LL(4) but not LL(3)

20) $S \to a \mid ab \mid abc \mid abcd \mid abcde$ LL(5) but not LL(4)

$k \geq 0$

Note: Any LL(k) CFG is convertible to LL(1) CFG.

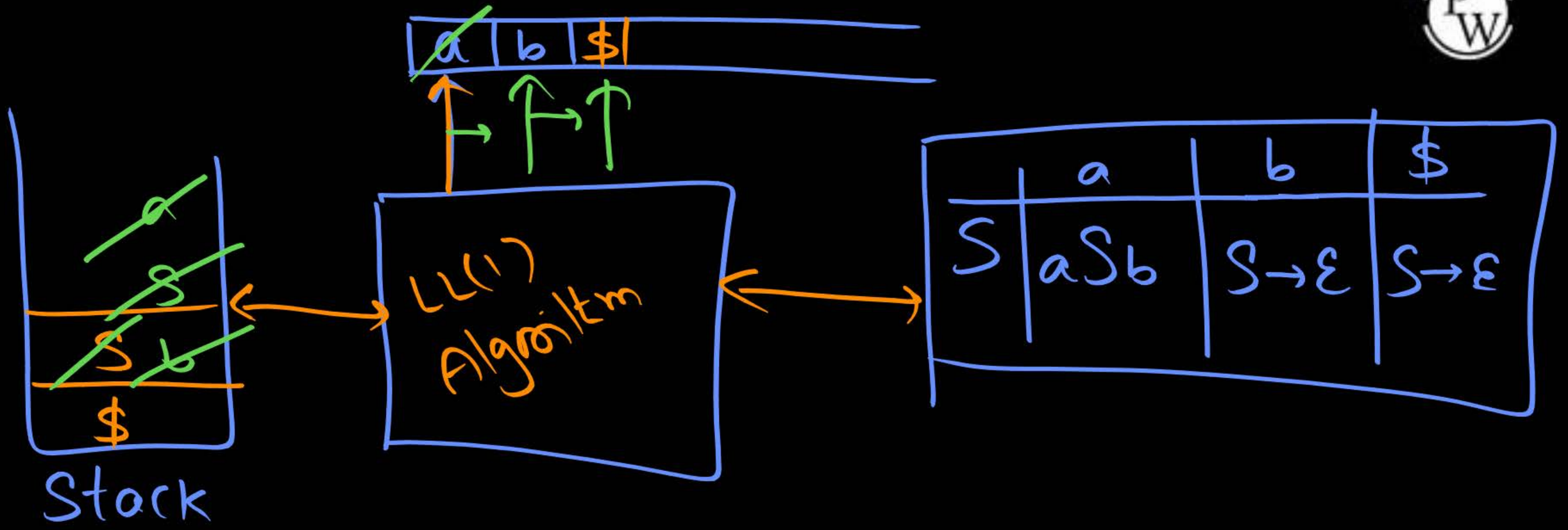
Represents LL(1) languages

L_1 = Set of all languages generated by LL(1) CFGs

L_2 = " " " by LL(2) CFGs

L_3 = " " " by LL(3) CFGs

$$L_1 = L_2 = L_3$$


a
b
$
a
b
$
S
S
a
b
$
Stack
LL(1)
Algorithm
S
aSb
S→ε
S→ε

$S \rightarrow aSb \mid \varepsilon$

$w = ab\$$

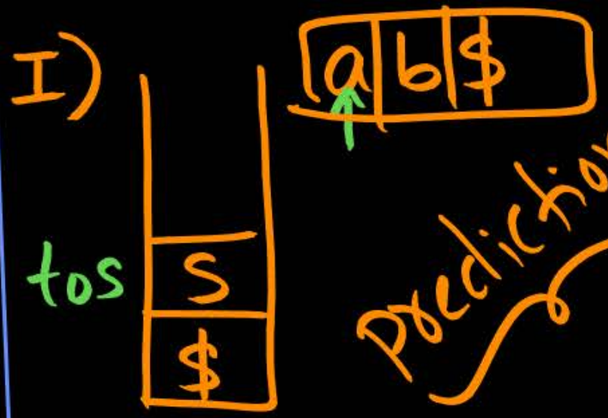
$S \Rightarrow aSb \Rightarrow ab$ (LMD)

Table:

M	a	b	$
S	S→aSb	S→ε	S→ε

Input: ab

I)

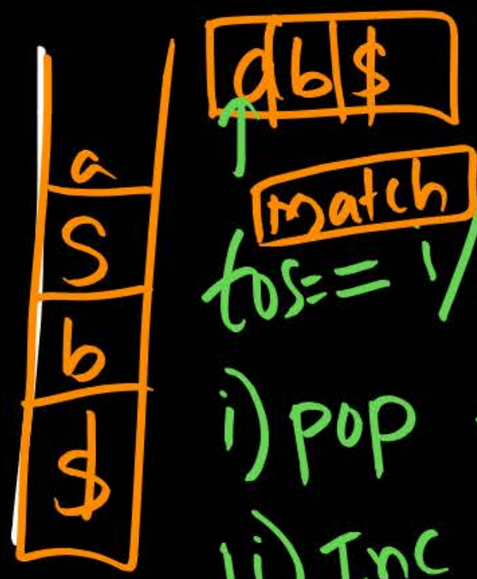


M[S, a] : S→aSb

i) Pop S

ii) Push aSb in reverse

II)

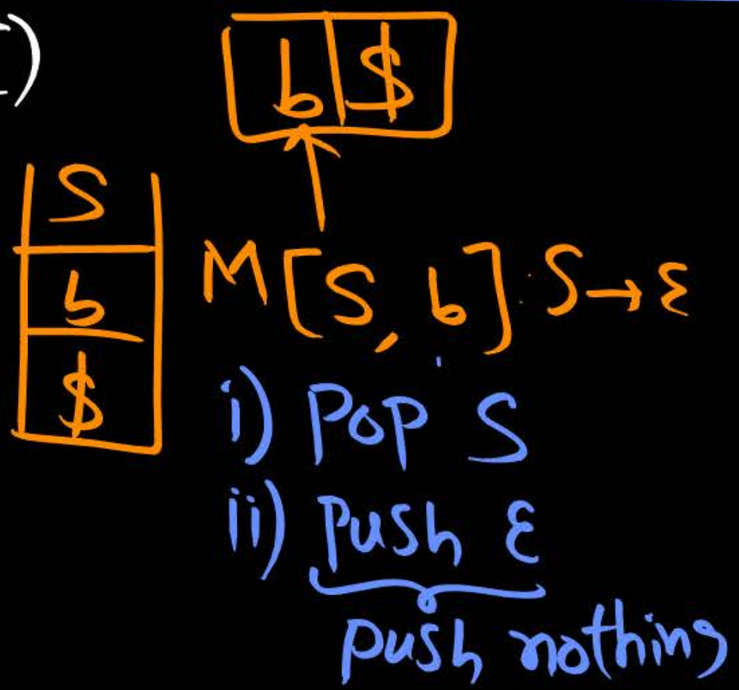


tos == i/p

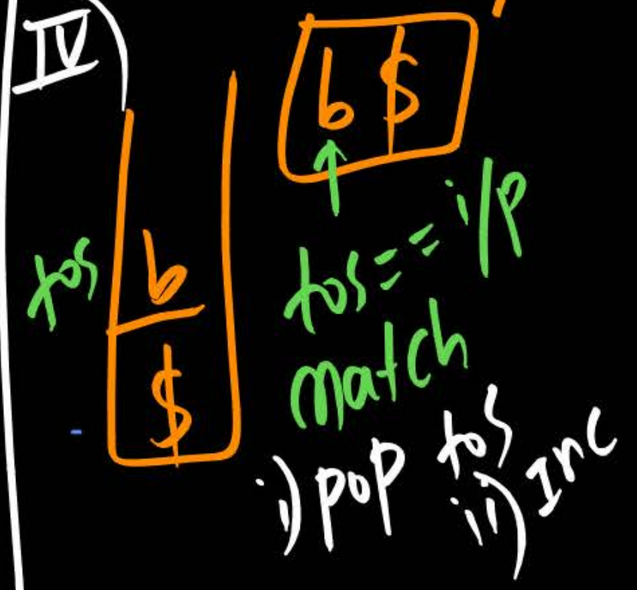
i) pop tos

ii) Inc pointer

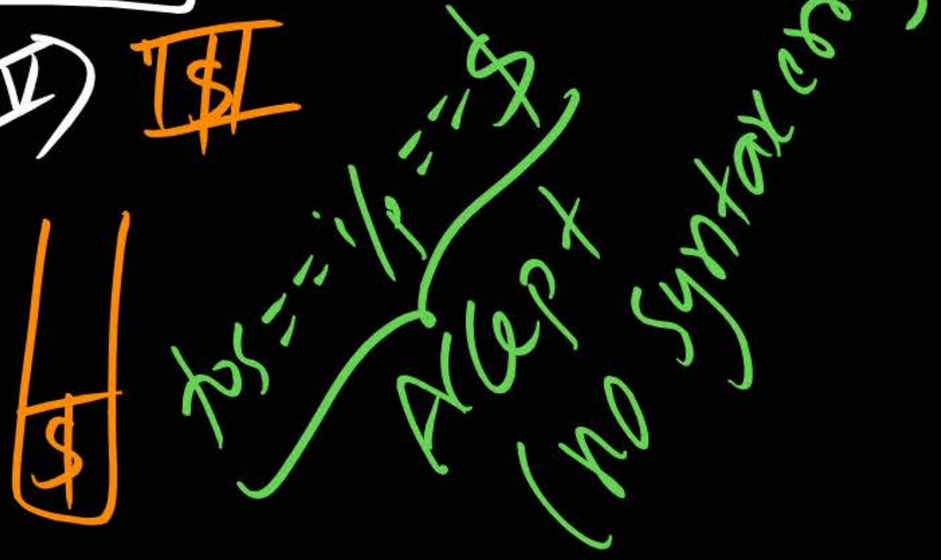
III)

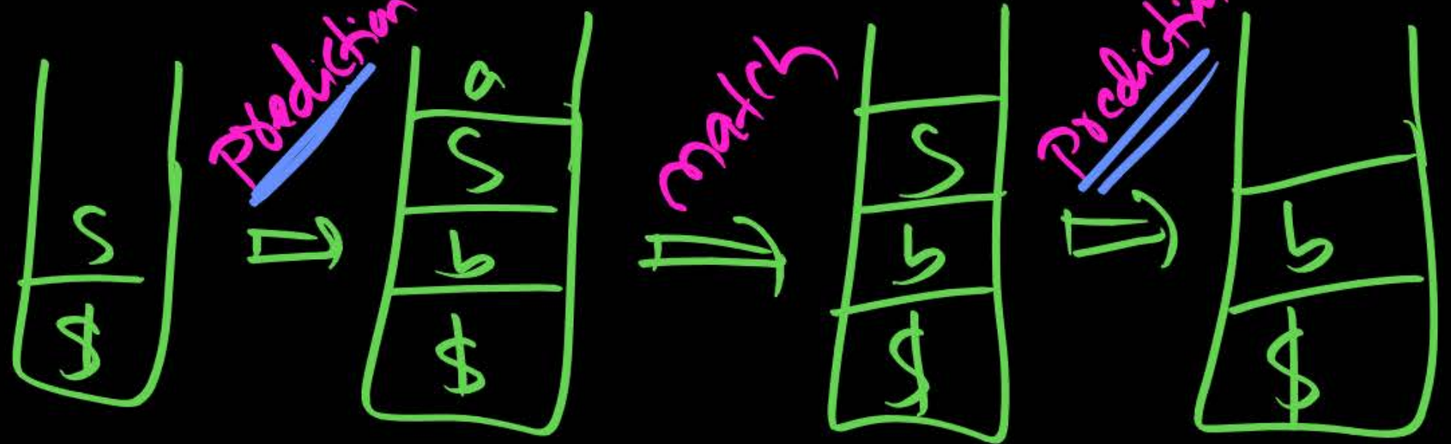


IV)



V)





2 4 3 2 1

$S \Rightarrow aSb \quad S \to aSb$

$aSb \Rightarrow ab \quad S \to \varepsilon$

ab

max stack size during LL(1) parsing to derive "ab" by assuming initially $ and S present on stack = 4

If tos is Non-terminal

⇓

check table

⇓

Production (Substitution)

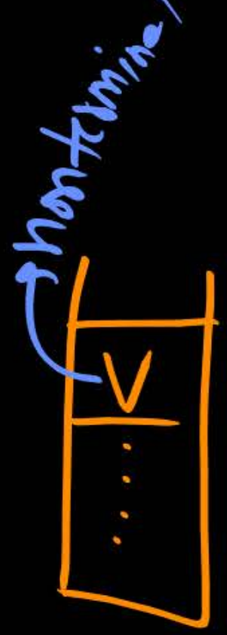
If tos is terminal

⇓

match with i/p

LL(1) parsing

↳ What are the problems can be faced during parsing i/p ?

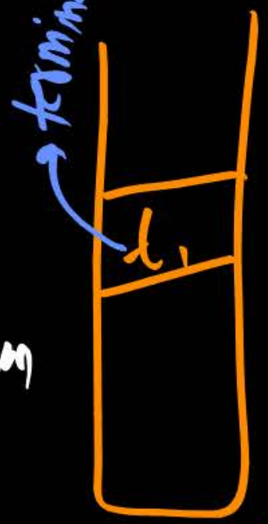


t

M[V, t] = Blank

If table has no production

I) N-conflict



t2

$t_1 \neq t_2$

II) T-conflict

If tos is not matching with i/p

I) N-conflict

If tos is Nonterminal

II) T-conflict

If tos is terminal

TDP

S

Using LMD

Productions (Substitutions)

String
ab

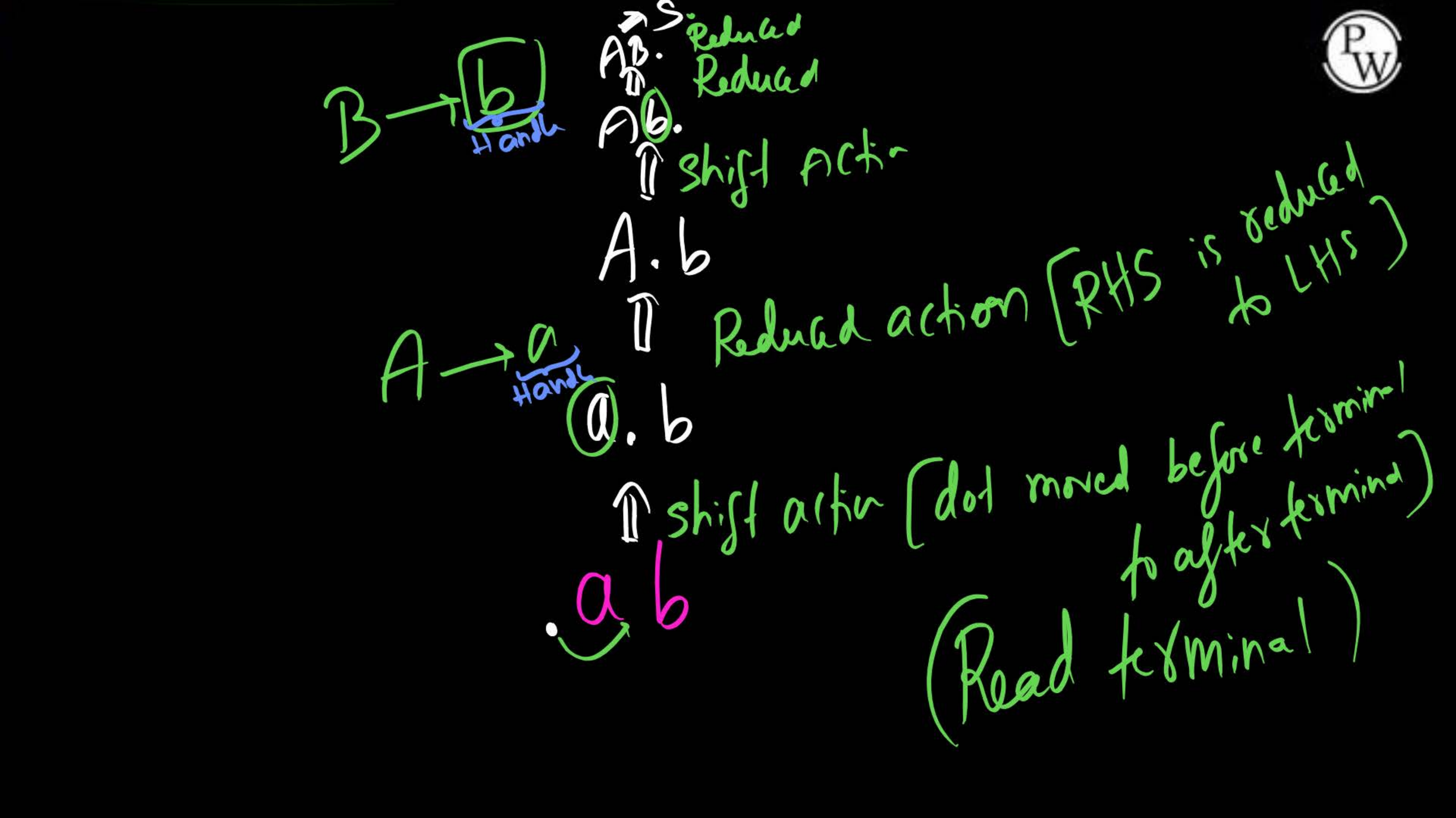
BUP [Shift-Reduced Parser]

S

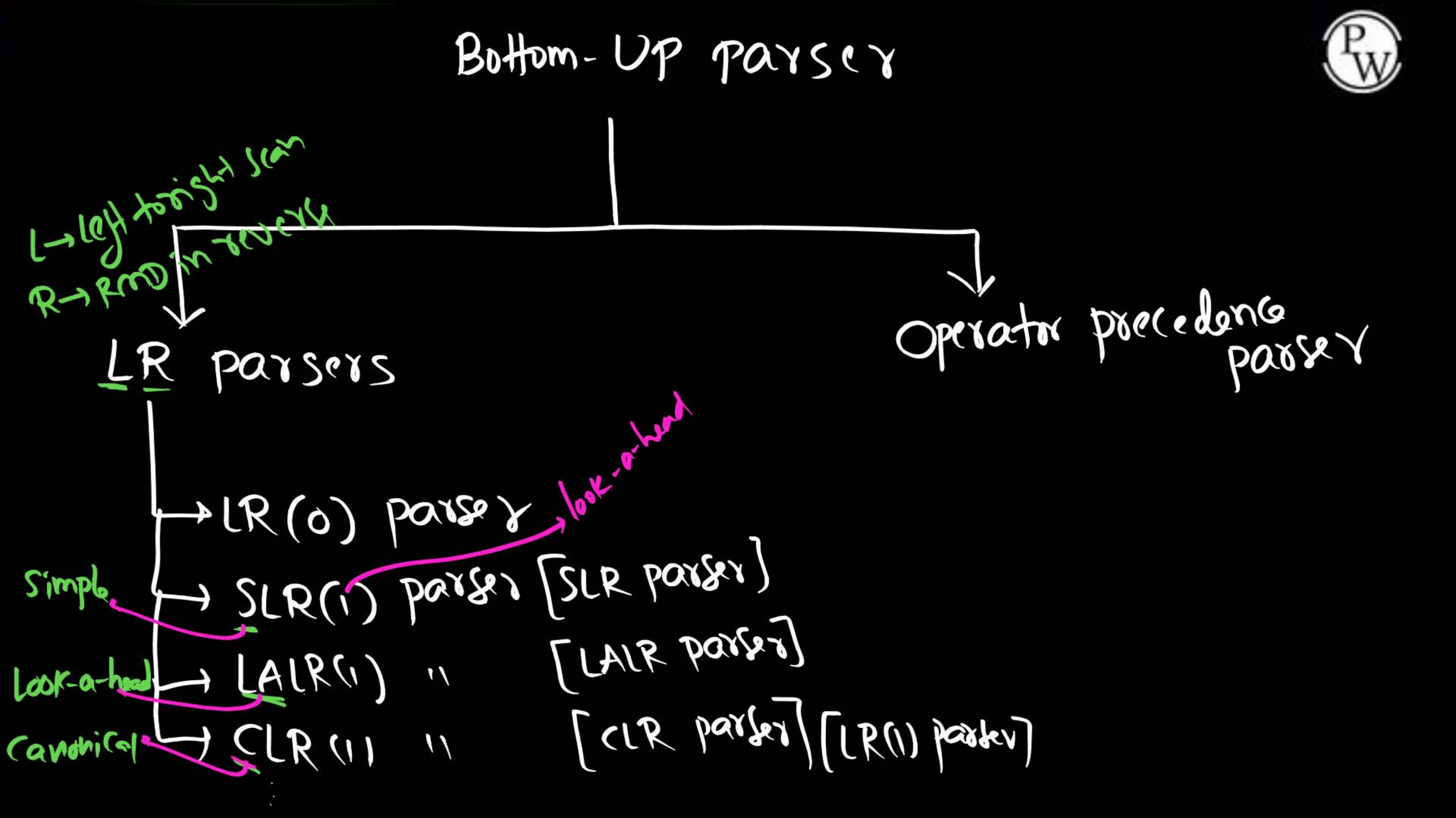
Using "RMD in reverse"
(Reverse of RMD)

[Shift action & Reduced action]

ab


B → b
Handle
S
A B.
A b.
Reduced
Reduced
shift Acti-
A.b
A → a
Handle
Reduced action [RHS is reduced to LHS]
a.b
shift actin [dot moved before terminal to after terminal]
.ab
(Read terminal)

Bottom-Up parser



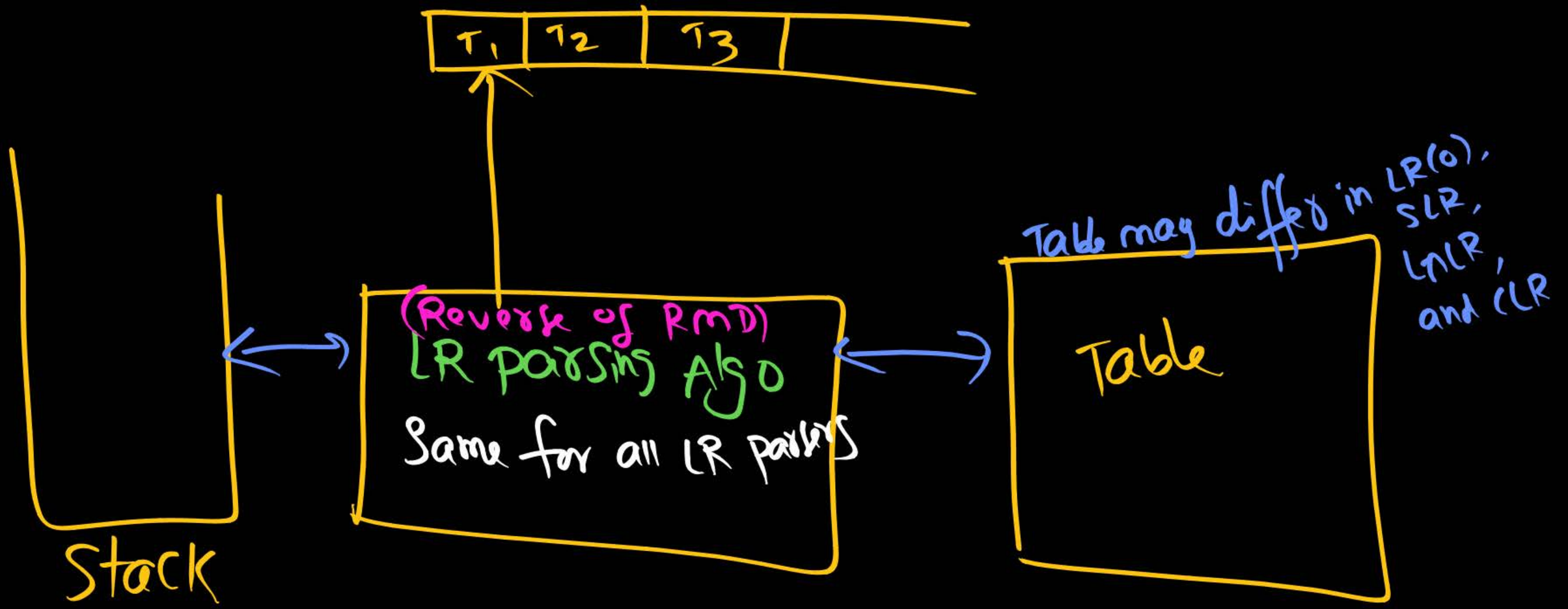
LR(0) Parser

- I) Augmented CFG
- II) Items
- III) closure( ) and goto( ) functions
- IV) Parsing DFA (Diagram) or Set of LR(0) items (Sets)

V) Parsing Table

***

LR Parsers (LR(0) / SLR / LALR / CLR)



TRUE ?

A) BUP uses RMD in reverse.

B) LR parser " " "

C) SR parser " " "

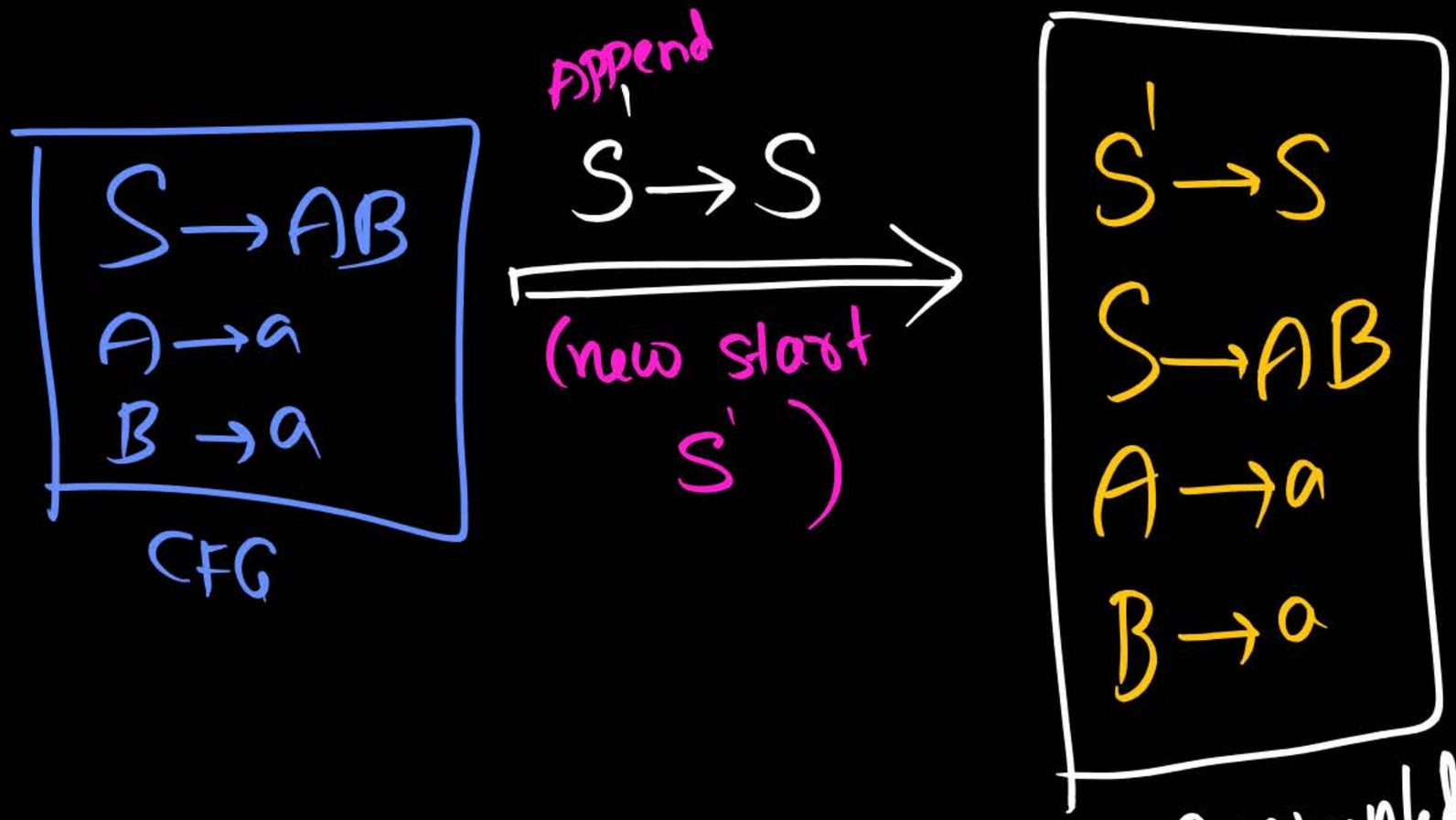
D) LR(0) " " "

E) SLR " " "

F) LALR " " "

G) CLR " " "

Q1) What is Augmented Grammar?



Q2) What is Item ? [It is a production along with "dot"]

$S \rightarrow aAbB$

production

$S \rightarrow .aAbB$

$S \rightarrow a.AbB$

$S \rightarrow aA.bB$

$S \rightarrow aAb.B$

$S \rightarrow aAbB.$

Items

Types of Items: [based on symbol immediately after Dot]

I) Shift Item [Terminal present just after dot]

II) State Item (goto Item) [Nonterminal present just after dot]

III) Reduced Item (Completed Item) [Dot is at the end]

$S \rightarrow .aAbB$ [shift Item]

$S \rightarrow a.AbB$ [state Item]

$S \rightarrow aA.bB$ [shift Item]

$S \rightarrow aAb.B$ [state Item]

$S \rightarrow aAbB.$ [Reduced Item (completed)]

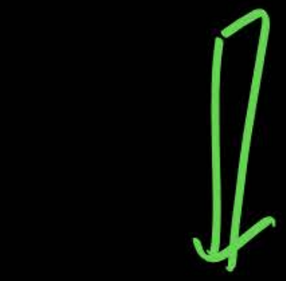
$S \rightarrow \underbrace{aB}_{\text{Handle}} \mid \underbrace{ad}_{\text{Handle}}$

$A \rightarrow \underbrace{a}_{\text{Handle}}$

$B \rightarrow \underbrace{b}_{\text{Handle}}$

Production is called as Handle

LL(1)



First & Follow

LR parser

Where is dot ?
Who is after dot immediately ?

Summary

- LL(1) CFG ✓
- LL(1) Algo ✓
- LR Concepts ✓

Next: Closure()
goto()


THANK YOU
GW
SOLDIERS !