30th January, 2025

Unit - 2 Bottom - Up Parsers

$E \xrightarrow{RMD} T$

$T \longrightarrow T * F$

$\longrightarrow T * id$

$\longrightarrow F * id$

$\longrightarrow id * id$

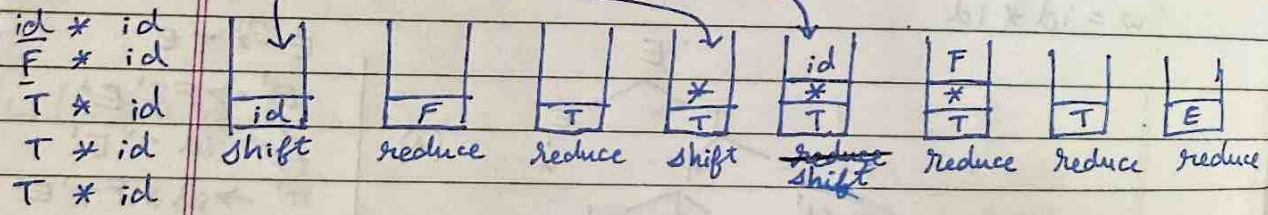
↑ Or: reverse of RMD

* match the stack with exactly from the right hand side of grammar, then replace it with left hand symbol

→ Working of Bottom Up Parser

* Shift - shift IP symbol from IP Buffer to stack
* Reduce
* handle - right hand side ~~symbol~~ production can be exactly matched ; eg replaced with left hand side symbol
* viable prefix - ~~top~~ content of stack at any time
* handle pruning - continuous process from IP string to start symbol

eg: id * id



grammar

① $E \rightarrow E+T$
② $E \rightarrow T$
③ $T \rightarrow T*F$
④ $T \rightarrow F$
⑤ $F \rightarrow id$

Date
Page

LR parsing - will expand right most - Bottom up

Stack	IP	Action (Shift/Reduce/Error/Accept)
$	id * id $	shift id
$ id	* id $	reduce using ⑤
$ F	* id $	reduce using ④
$ T	* id $	shift *
$ T*	id $	shift id
$ T*id	$	reduce using ⑤
$ T*F	$	reduce using ③
$ T	$	reduce using ②
$ E	$	accept

Shift-reduce conflict

→ Problem with Bottom-up Parser

1. Shift-reduce conflict (s/r)
2. reduce-reduce conflict (r/r)

→ Types of LR parsers

1. LR (0)
2. SLR(1) / SLR / Simple LR
3. LALR (1) / LALR / Look Ahead LR
4. CLR (1) / LR(1) / LR / Canonical LR

powerful ↓

* power of parsers : how quickly it can catch errors

* no. of states

$$(n_{LR(0)} = n_{SLR} = n_{(LALR)}) \leq n_{CLR}$$

H/W (q1) $S \to aAa \mid \lambda$

$A \to abS \mid \lambda$

First :	S	A
	a	a
	λ	λ

Follow :	S	A
	$	a
	follow(A) → a	

LL(1)

	a	b	$
S	$S \to aAa$, $S \to \lambda$		
A	$A \to abS$, $A \to \lambda$		

∴ Not $\in$ LL(1)

(q2) $S \to AA$

$A \to aA \mid b$

First:	S	A
	a	a
	b	b

Follow :	S	A
	$	first(A) = a, b
		follow(A)
		follow(S) = $

LL(1)

	a	b	$
S	$S \to AA$	$S \to AA$	
A	$A \to aA$	$A \to b$	

$\in$ LL(1)

3rd February, 2025

Introduction to Table Driven LR Parsers

* Class of grammar for which we can build shift-reduce parser (BUPs) is called LR grammars
  - non-backtracking
  - shift-reduce parsing

LL(1)-Grammar ⊂ LR(1)-Grammar

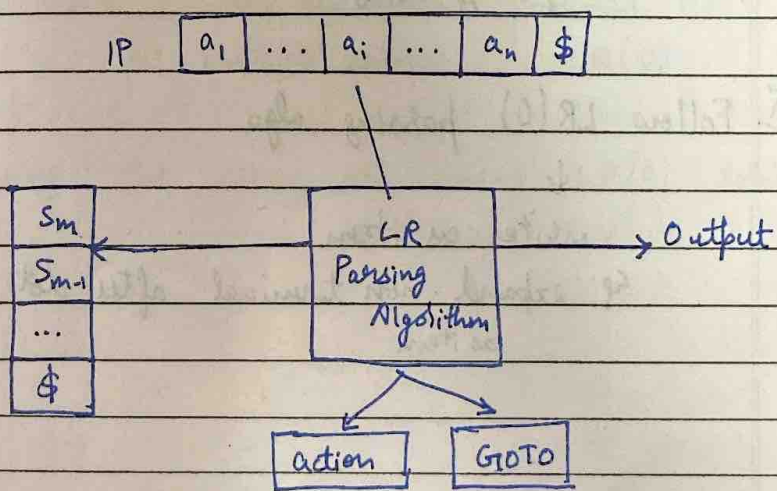
→ LR Automata

* All 4 types of LR parsers use DFA
* DFA of LR(0) & SLR(1) : LR(0) automata
  LR(0) items
* DFA of LALR & CLR
  - LR(1) automata
    LR(1) items

items
- initial item: $S \to \cdot ab$: parser has not seen anything
- $S \to a \cdot b$: parser has seen a, yet to see b
- $S \to ab \cdot$: parser has seen both & ready to reduce
  ↓
  final item / Kernel item

item: production with dot

Model:



→ LR(0) Parser - Construction

Steps for constructing the LR automation.

1. Augment the grammar
2. Number your productions in the original grammar
3. Follow LR(0) Parsing Algorithm
4. Create the parsing table

eg1. $S \rightarrow AA$
$A \rightarrow aA \mid b$

I Augment the grammar : Introduce new start symbol

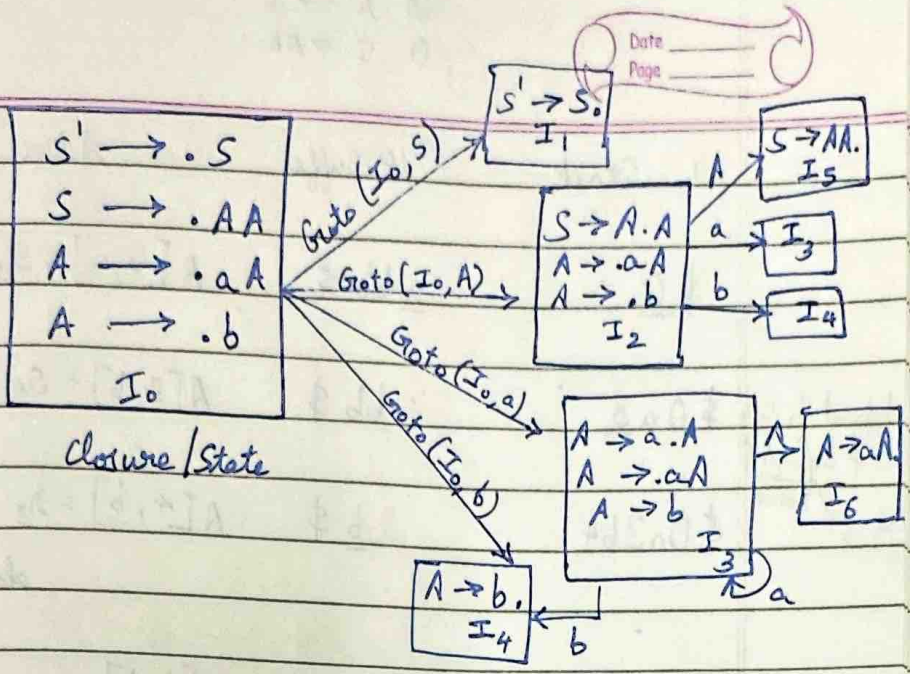
$S' \rightarrow S$
$S \rightarrow AA$
$A \rightarrow aA \mid b$

II Number your productions : only old ones

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

III Follow LR(0) parsing algo
↓
write as item
↓ expand non terminal after dot as item

IV



IV LR(0) Table Construction

non terminal ⇒ GoTo
grammar reduced on state ⇒ fill in all of action
(S → AA•)

State	Action			GoTo	
	a	b	$	S	A
0	S_3	S_4		1	2
1			ACCEPT		
2	S_3	S_4			5
3	S_3	S_4			6
4	r_3	r_3	r_3		
5	r_1	r_1	r_1		
6	r_2	r_2	r_2		

on state 5, we can reduce grammar ①
No multiple entries ⇒ LR(0)

V Parsing the string abb using LR(0) table

start with state 0

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

Stack	IP Buffer	Action
$0	abb$	A[0,a] = S_3 push a on stack, push 3 on stack
$0a3	bb$	A[3,b] = S_4 push b & 4
$0a3b4	b$	A[4,b] = r_3 pop b & 4, push A also Goto [3,A] = 6
$0a3A6	b$	A[6,b] = r_2 pop a3A6 push A Goto [0,A] = 2
$0A2	b$	A[2,b] = S_4 push b & 4
$0A2b4	$	A[4,$] = r_3 pop b & 4 push A Goto [2,A] = 5
$0A2A5	$	A[5,$] = r_1 pop 5 A 2 A push S, Goto [0,S] = 1
$0S1	$	A[1,$] = ACCEPT

pop b & 4
$0a3A
push A
6

eg 2

I

II N

III

S
S
S

eg2.

$S \to dA \mid aB$

$A \to bA \mid c$

$B \to bB \mid c$

I Augment

$S' \to S$

$S \to dA \mid aB$

$A \to bA \mid c$

$B \to bB \mid c$

II Number

$S' \to S$

① $S \to dA$ ~~$\mid aB$~~

② ~~S~~ $S \to aB$

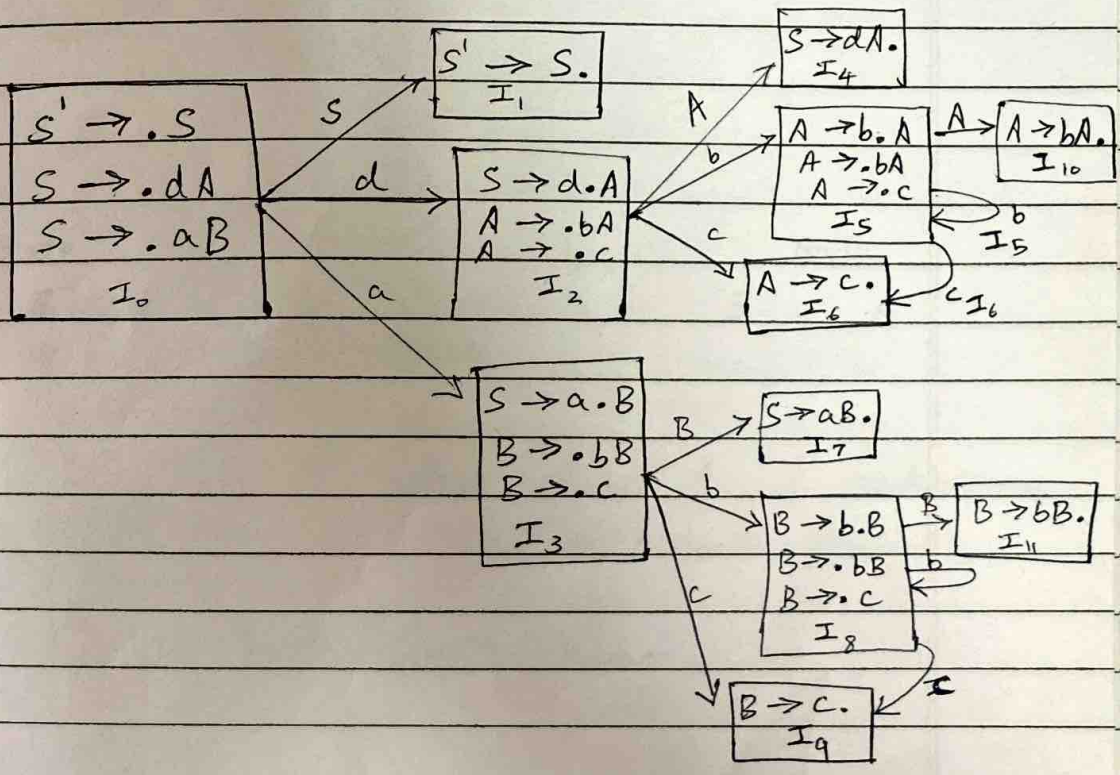
③ $A \to bA$

④ $A \to c$

⑤ $B \to bB$

⑥ $B \to c$

III



IV

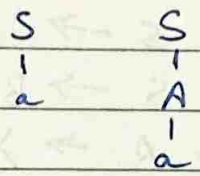
State	Action					GOTO	
	a	b	c	d	$	S	A	B
0	S_3			S_2		1		
1					ACCEPT			
2		S_5		S_6			4	
3		S_8		S_9				7
4	R_1	R_1		R_1	R_1			
5		S_5		S_6			10	
6	R_4	R_4		R_4	R_4			
7	R_2	R_2		R_2	R_2			
8		S_8		S_9				11
9	R_6	R_6		R_6	R_6			
10	R_3	R_3		R_3	R_3			
11	R_5	R_5		R_5	R_5			

* There should be unique LMD / RMD & the should be similar, if there are more the 1 tree possible ⇒ conflict / ambiguity

eg:

$S \to A \mid a$

$A \to a$

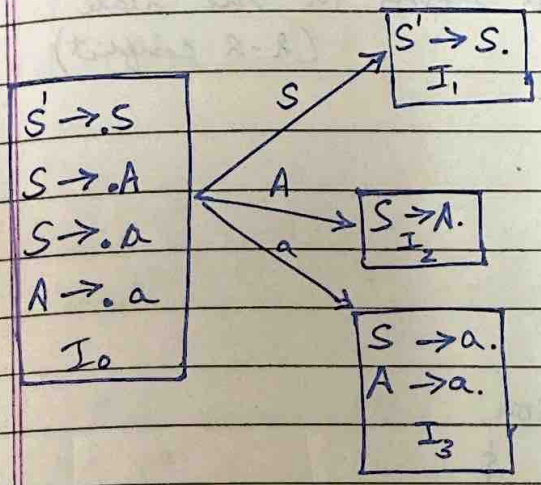


$S' \to S$

① $S \to A$

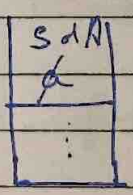
② $S \to a$

③ $A \to a$



State	Action		GoTo	
	a	$	S	A
3	r_2 / r_3	r_2 / r_3		

in same state if 2 reductions ⇒ reduce-reduce conflict



parser doesn't know

q. $S \to AaAb \mid BbBa$ $\{ab, ba\}$

$A \to \lambda$

$B \to \lambda$

$S' \to S$

1. $S \to AaAb$
2. $S \to BbBb$
3. $A \to \lambda$
4. $B \to \lambda$

$S' \to .S$

$S \to .AaAb$

$S \to .BbBb$

$A \to .\lambda \quad B \to .\lambda$

I_0

2 final items in the state (r-r conflict)

State	Action		
	a	b	$
0	r_3/r_4	r_3/r_4	r_3/r_4

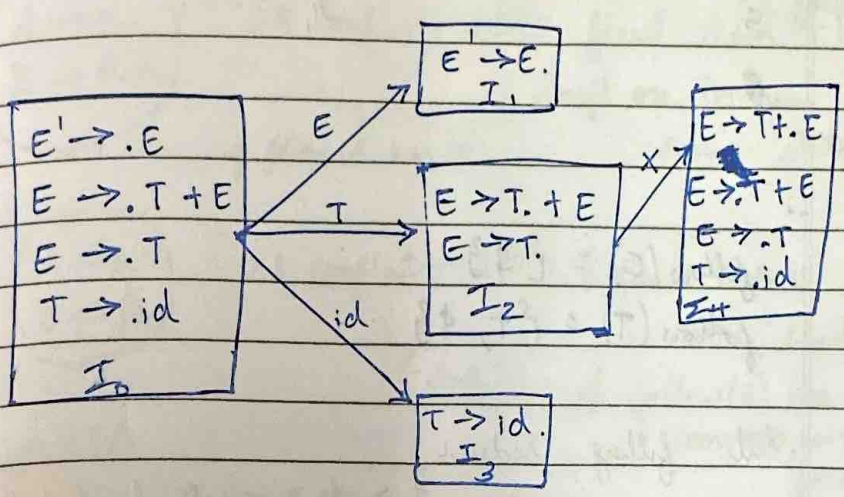
2 diff productions reaching final state in same state ⇒ ambiguous. conflict

q. $E \rightarrow T + E \mid T$

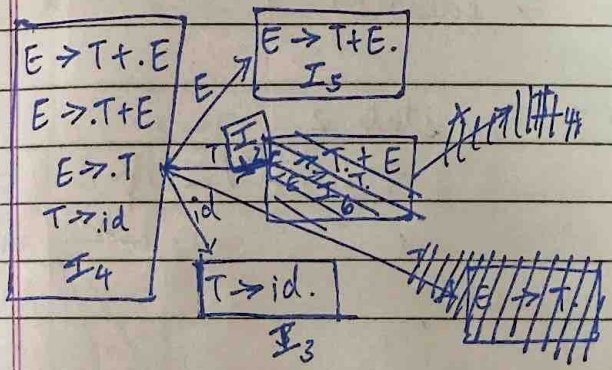
$T \rightarrow id$

$E' \rightarrow E$

1. $E \rightarrow T + E$
2. $E \rightarrow T$
3. $T \rightarrow id$



state	Action		
	+ :	id	$
2	r_2 / s_4	r_2	r_2



* here final item & there is a transition
  ↓ conflict : shift - reduce

* same state, 2 final items
  ↓ conflict : reduce - reduce

6th February 2025

SLR(1) table

state	Action			GoTo	
	+	id	$	E	T
0		S_3		1	2
1			ACCEPT		
2	S_4		r_2		
3					
4		S_3		5	2
5			r_1		

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

while filling reduce

eg: $T \to id.$ (I_3) instead of putting r_3 in all of state 3 put only in follow(T)

① $E \to T+E$ in state (5) reducing
instead find follow (left hand side non terminal)
⇒ follow(E) = $

② $E \to T$ state 2
⇒ follow(E) = $

LR(0)
(reduce: ov

A → α.
B → β.

A → α.
B → β

eg: SLR(1)

S' → S
① S → A
② A → aA
③ A → b

S' → .S
S → .AA
A → .aA
A → .b
I_0

S
A
a
b

LR(0) Automata

- LR(0) parsing table (reduce: on entire action)
- SLR(1) parsing table (reduce: only follow(LHS))

$A \to \alpha.$, $B \to \beta.a$ — a → SR conflict → LR(0): final item + shift on terminal in same state

SLR(1): if follow(A) has 'a'

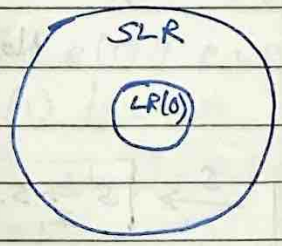
$A \to \alpha.$, $B \to \beta.$ — RR conflict → LR(0): 2 final items in same state

SLR(1): if follow(A) has smtg common with follow(B)

eg: SLR(1)

$S \to AA$

$A \to aA \mid b$

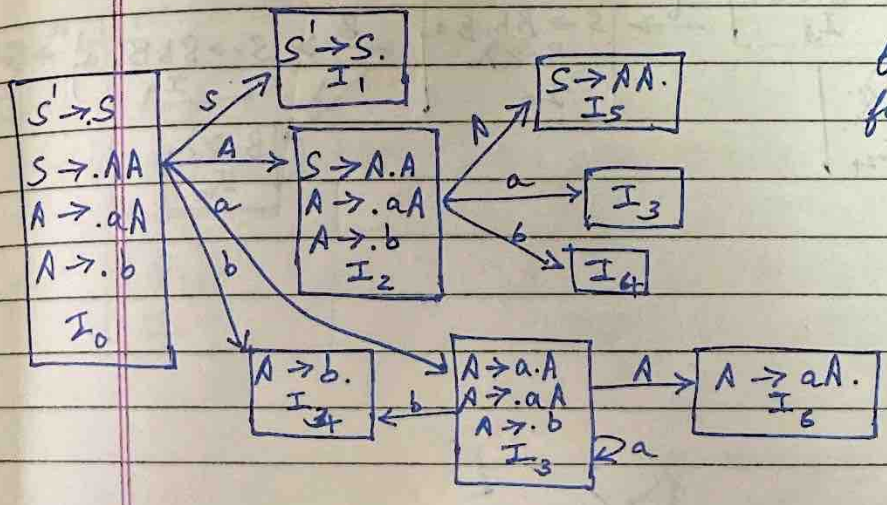


$S' \to S$

① $S \to AA$

② $A \to aA$

③ $A \to b$



follow(S) = {$}

follow(A) = {$, a, b}

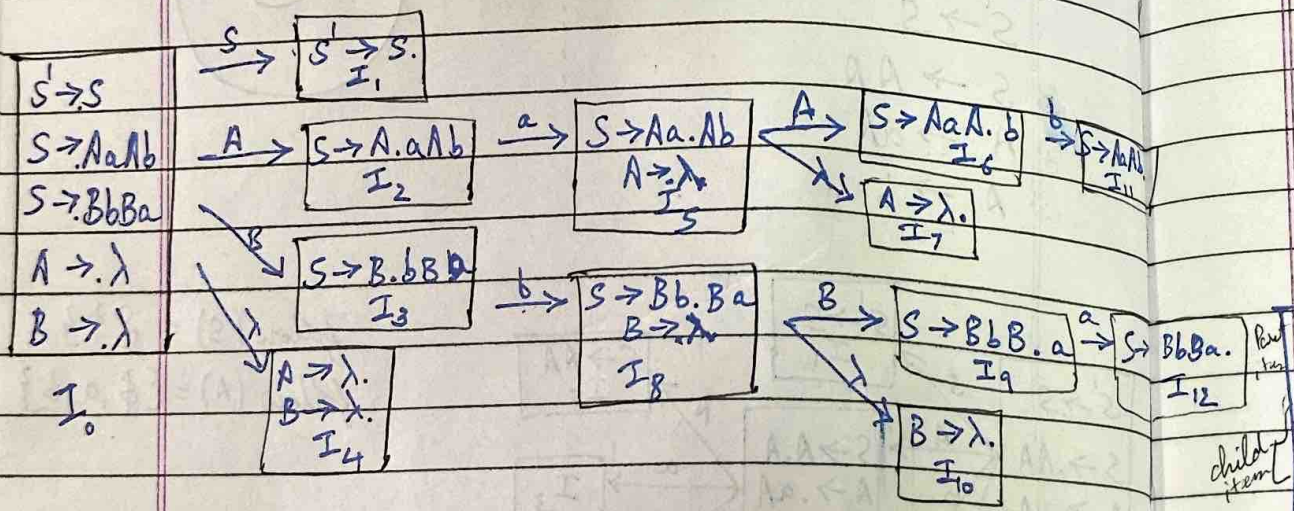
SLR(1)

State	Action			Goto	
	a	b	$	S	A
0	S_3	S_4		1	2
1			ACCEPT		
2	S_3	S_4			5
3	S_3	S_4			6
4	R_3	R_3	R_3		
5			R_1		
6	R_2	R_2	R_2		

$S' \to S$

eg 3. ① $S \to AaAb$
② $S \to BbBa$
③ $A \to \lambda$
④ $B \to \lambda$

Not LR(0) check SLR(1)



* In follow set is common, then SLR ⇒ Reduce - Reduce

(A) → α.
(B) → β.

CLR & LALR

uses LR(1) automata; it uses LR(1) item

↓

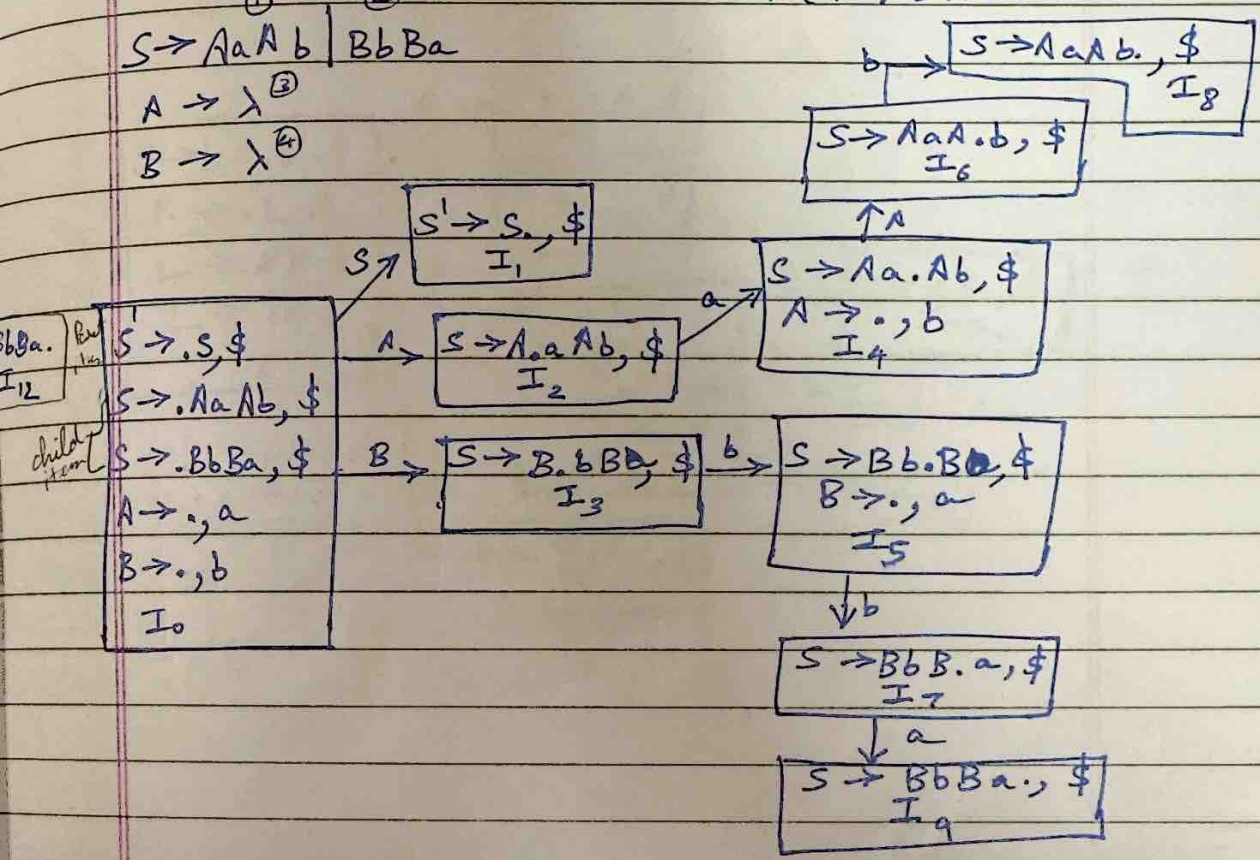
LR(1) item = LR(0) + Look ahead

In LR(0), we start with $S' \rightarrow .S$

but in LR(1) automata we start with $S' \rightarrow .S, \$$ (lookahead)

eg1. Is the Grammar CLR / LR / LR(1) / CLR(1) / LALR(1) / LALR

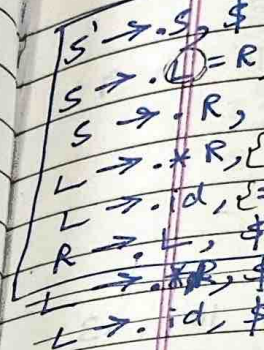
① ②
S → AaAb | BbBa
A → λ ③
B → λ ④

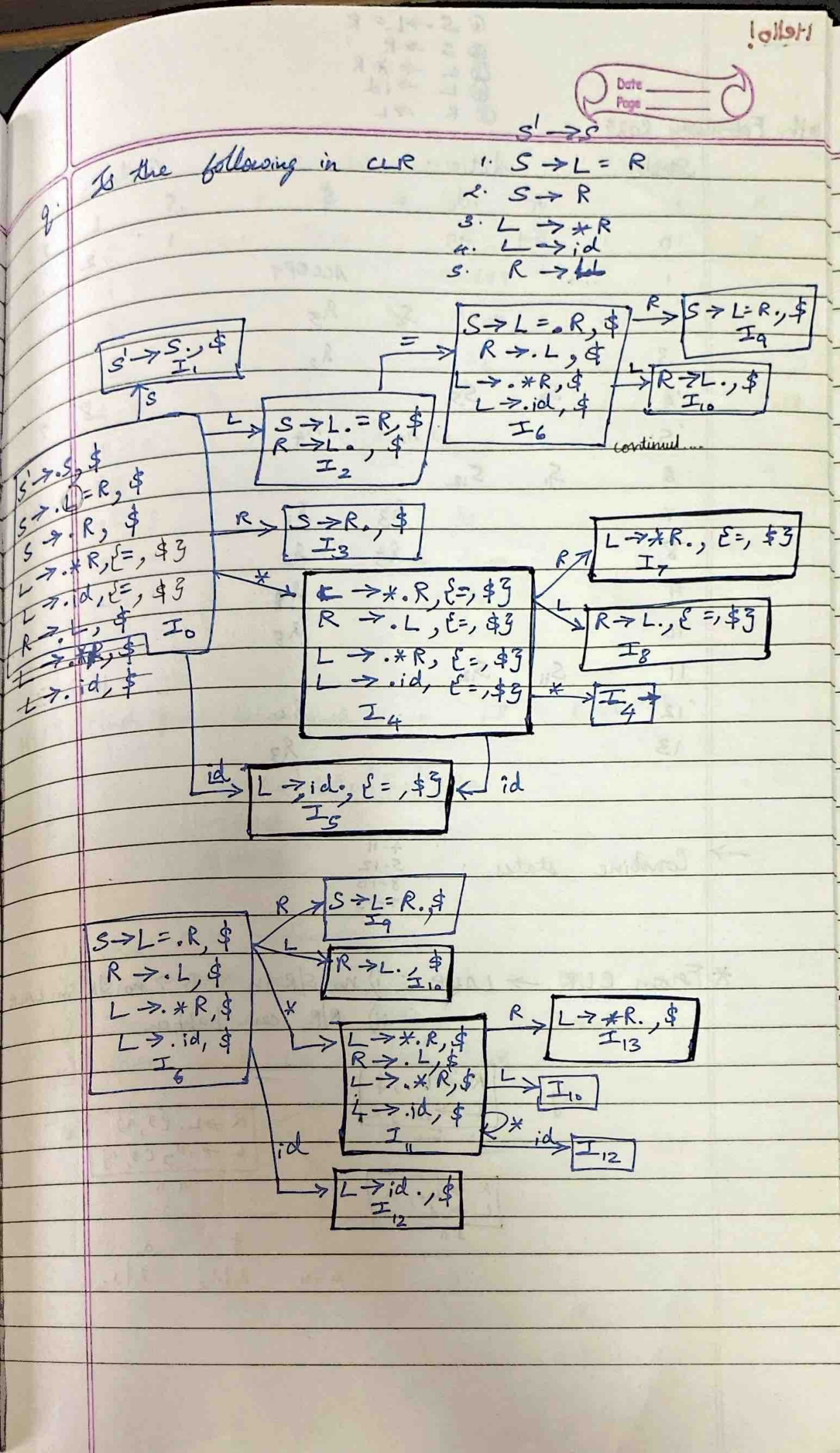


* Calculate lookahead for child item L by using Parent item

lookahead of child items = follow (NT) in parent item

* place reduce move only in lookahead




Date
Page
S' → S
Q: To the following in CLR
1. S → L = R
2. S → R
3. L → *R
4. L → id
5. R → L
S' → S., $
I1
S
S' → .S, $
S → .L = R, $
S → .R, $
L → .*R, =, $
L → .id, =, $
R → .L, $
L → .*R, $
L → .id, $
I0
L
S → L.= R, $
R → L., $
I2
=
S → L = .R, $
R → .L, $
L → .*R, $
L → .id, $
I6
R
S → L = R., $
I9
L
R → L., $
I10
continued...
R
S → R., $
I3
*
L → *.R, =, $
R → .L, =, $
L → .*R, =, $
L → .id, =, $
I4
R
L → *R., =, $
I7
L
R → L., =, $
I8
*
I4
id
L → id., =, $
I5
id
S → L = .R, $
R → .L, $
L → .*R, $
L → .id, $
I6
R
S → L = R., $
I9
L
R → L., $
I10
*
L → *.R, $
R → .L, $
L → .*R, $
L → .id, $
I11
R
L → *R., $
I13
L
I10
*
id
I12
id
L → id., $
I12

Hello!

10th February, 2025

① $S \to L = R$
② $S \to R$
③ $L \to *R$
④ $L \to id$
⑤ $R \to L$

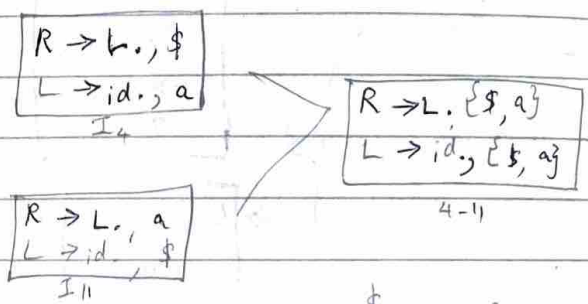
State	Action				Goto		
	*	id	=	$	S	L	R
0	S_4	S_5			1	2	3
1				ACCEPT			
2			S_6	R_5			
3				R_2			
4	S_4	S_5				8	7
5			R_4	R_4			
6	S_{11}	S_{12}				10	9
7			R_3	R_3			
8			R_5	R_5			
9				R_1			
10				R_5			
11	S_{11}	S_{12}				10	13
12				R_4			
13				R_3			

HW 9:

→ Combine states : 4-11
5-12
8-10
7-13

* From CLR → LALR : i) no S/R in CLR ⇒ no S/R in LALR
ii) R/R can happen

eg:



	$	a
4-11	$R_1 \mid R_2$	$R_1 \mid R_2$

i)
ii)
iii)
iv)

State	Action				Goto		
	*	id	=	$	S	L	R
0	S_{4-11}	S_{5-12}			1	2	3
1				ACCEPT			
2			S_6	r_5			
3				r_2			
4-11	S_{4-11}	S_{5-12}				8-10	7-13
5-12			r_4	r_4			
6	S_{4-11}	S_{5-12}				8-10	9
7-13			r_3	r_3			
8-10			r_5	r_5			
9				r_1			

HW 9: Find given Grammar is in LR & LALR ?

$S \rightarrow Aa \mid bAc \mid Bc \mid bBa$

$A \rightarrow d$

$B \rightarrow d$

i) Draw LR(1) automata

ii) Find states to be merged in LALR

iii) Draw tables for both LR & LALR

iv) Also check LL, LR(0), SLR

$S' \to S$

① $S \to Aa$

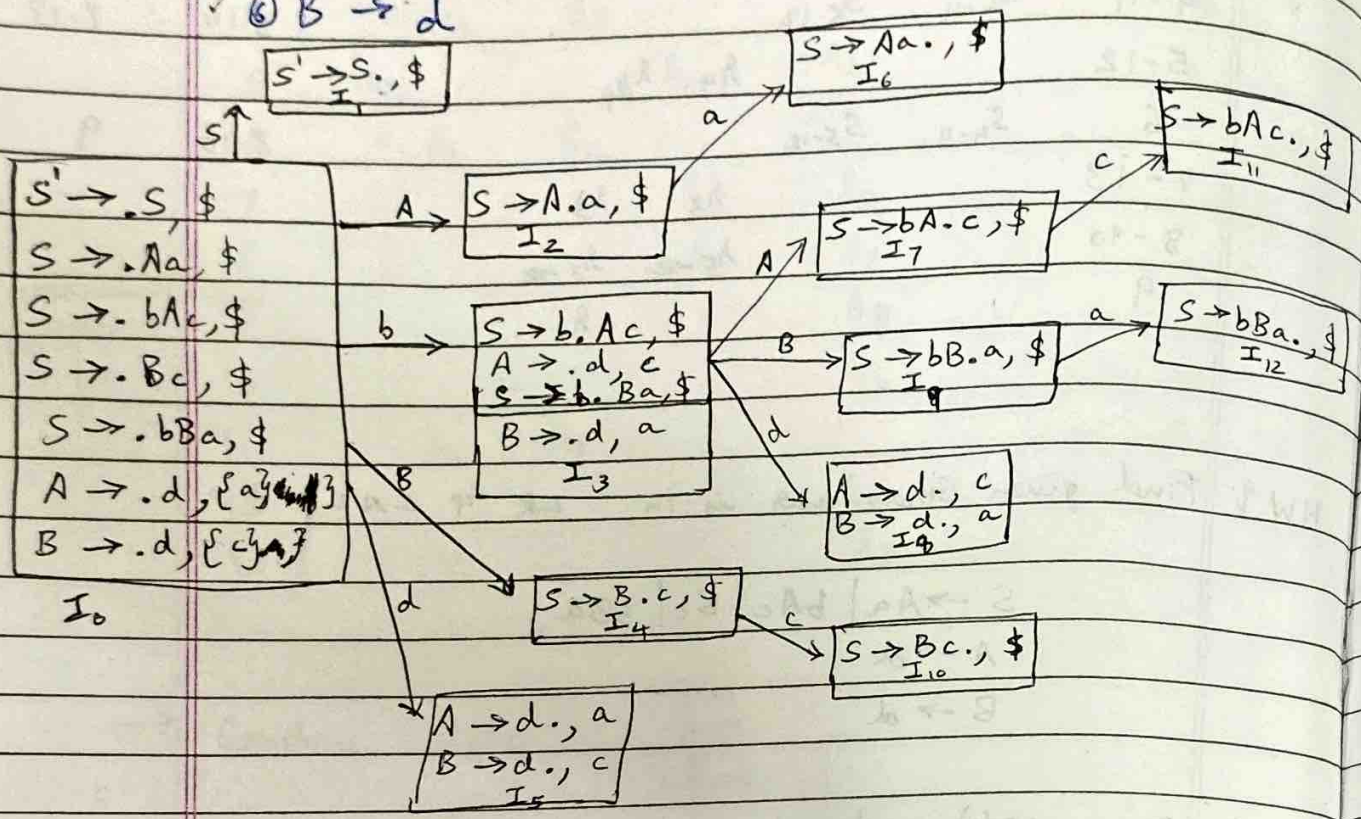
② $S \to bAc$

③ $S \to Bc$

④ $S \to bBa$

⑤ $A \to d$

⑥ $B \to d$



CLR

state	Action a	b	c	d	$	Goto S	A	B
0		S_3		S_5		1	2	4
1					ACCEPT			
2	S_6							
3				S_8			7	9
4			S_{10}					
5	r_5		r_6					
6					r_1			
7			S_{11}					
8	r_5		r_5					
9	r_2							
10					r_3			
11					r_2			
12					r_4			

5-8 states to be merged

state	Action a	b	c	d	$	Goto S	A	B
0		S_3		S_{5-8}		1	2	4
1					ACCEPT			
2	S_6							
3				S_{5-8}			7	9
4			S_{10}					
5-8	$r_5 \mid r_6$		$r_6 \mid r_5$					
6					r_1			
7			S_{11}					
9	S_{12}							
10					r_3			
11					r_2			
12					r_4			

Since 2 states ⇒ Not LALR

LL check

First

S	A	B
b	d	d
d		

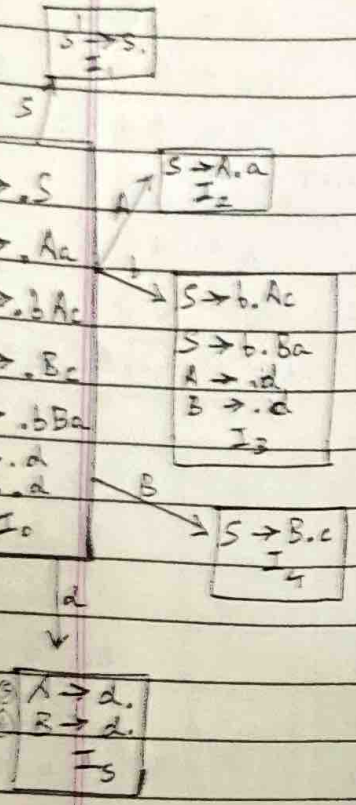
Follow

S	A	B
$	a	a
	c	c

	a	b	c	d	$
S		$S \to bAc$		$S \to Aa$ $S \to Bc$	
A					
B					

∴ Not LL

SLR parsing



follow(A) = follow(B)

SLR	state	a	b	d	c	$
	5	r_5/r_6		r_5/r_6		

LR(0)	5	r_5/r_6	r_5/r_6	r_5/r_6	r_5/r_6	r_5/r_6

∴ Not LR(0), SLR

February 2025

Conflicts in LR(1) / CLR / CLR(1) / LR

a) S/R conflict

$A \to \alpha . a\beta, \{c, d\}$
$B \to \gamma ., \{a, \$\}$

$\xrightarrow{a}$ shift on 'a' & there exists a final item for which the lookahead is 'a' (ie reduce move under 'a')

b) R/R conflict

$A \to \alpha ., \{a, b\}$
$B \to \beta ., \{a\}$

Two final items with common lookahead symbol

Conflicts in LALR

I_4:
1. $A \to \alpha ., a$
2. $B \to \beta ., b$

I_{11}:
$A \to \alpha ., b$
$B \to \beta ., a$

merged in LALR

I_{4-11}:
$A \to \alpha ., \{a, b\}$
$B \to \beta ., \{a, b\}$

R/R conflict

* LALR can have shift reduce conflict only if CLR has shift reduce conflict.

Summary

Parser	LR(0)	SLR(1) SLR Simple LR	LALR(1) LALR (Lookahead LR)	CLR(1) / CLR LR(1) / LR (Canonical LR)
DFA	LR(0) automata LR(0) item		LR(1) automata LR(1) item = LR(0) item + lookahead	
reduce move	entire row of state that contains final item	place reduce move only in Follow(LHS)	Place reduce node in the lookahead	Place reduce move in the lookahead
Power	Least	> LR(0)	> SLR	Most
no. of states	$n(LR(0)) = n(SLR) = n(LALR) <= n(CLR)$			

* No Grammar will work with ambiguous grammar

unambiguous grammars		ambiguous grammars
LL(k)	LR(k)	
LL(1)	LR(1)	
	LALR(1)	
	SLR(1)	
	LR(0)	
LL(0)		

CLR state

0
1
2
3
4
5

4 $E \to T + E \mid T$

$T \to id$

Check CLR, LALR

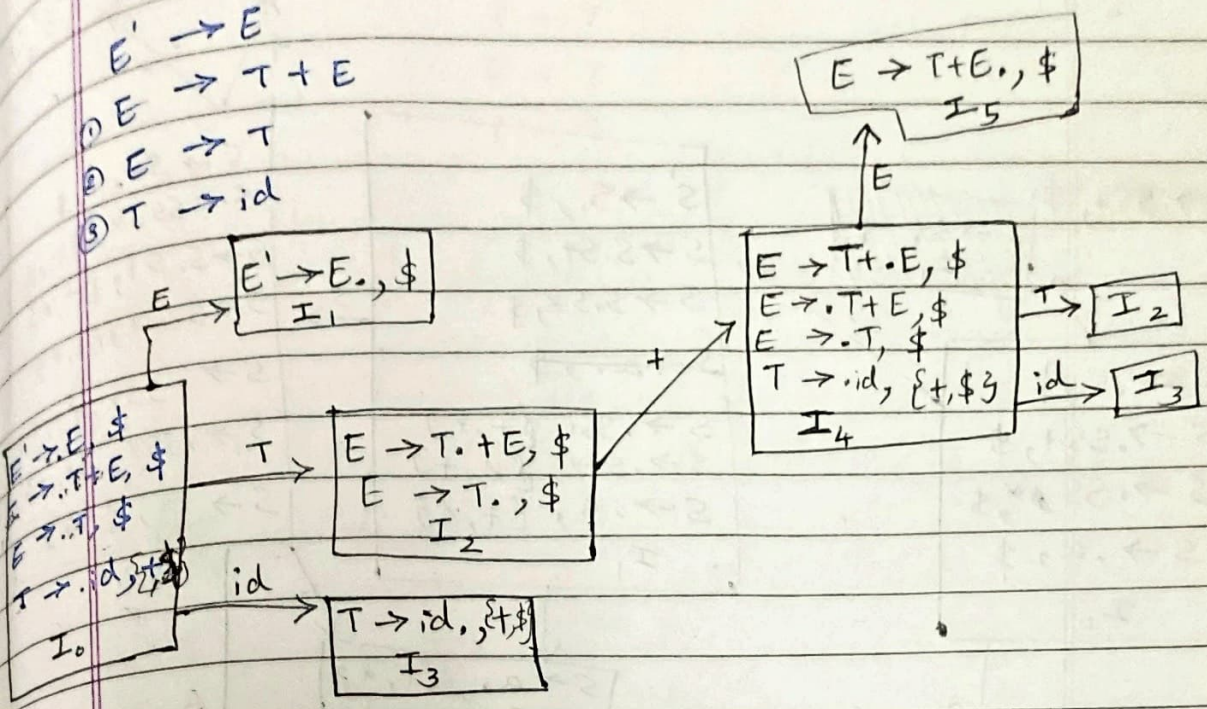
parse: id + id

$E' \to E$

① $E \to T + E$

② $E \to T$

③ $T \to id$



CLR parsing table

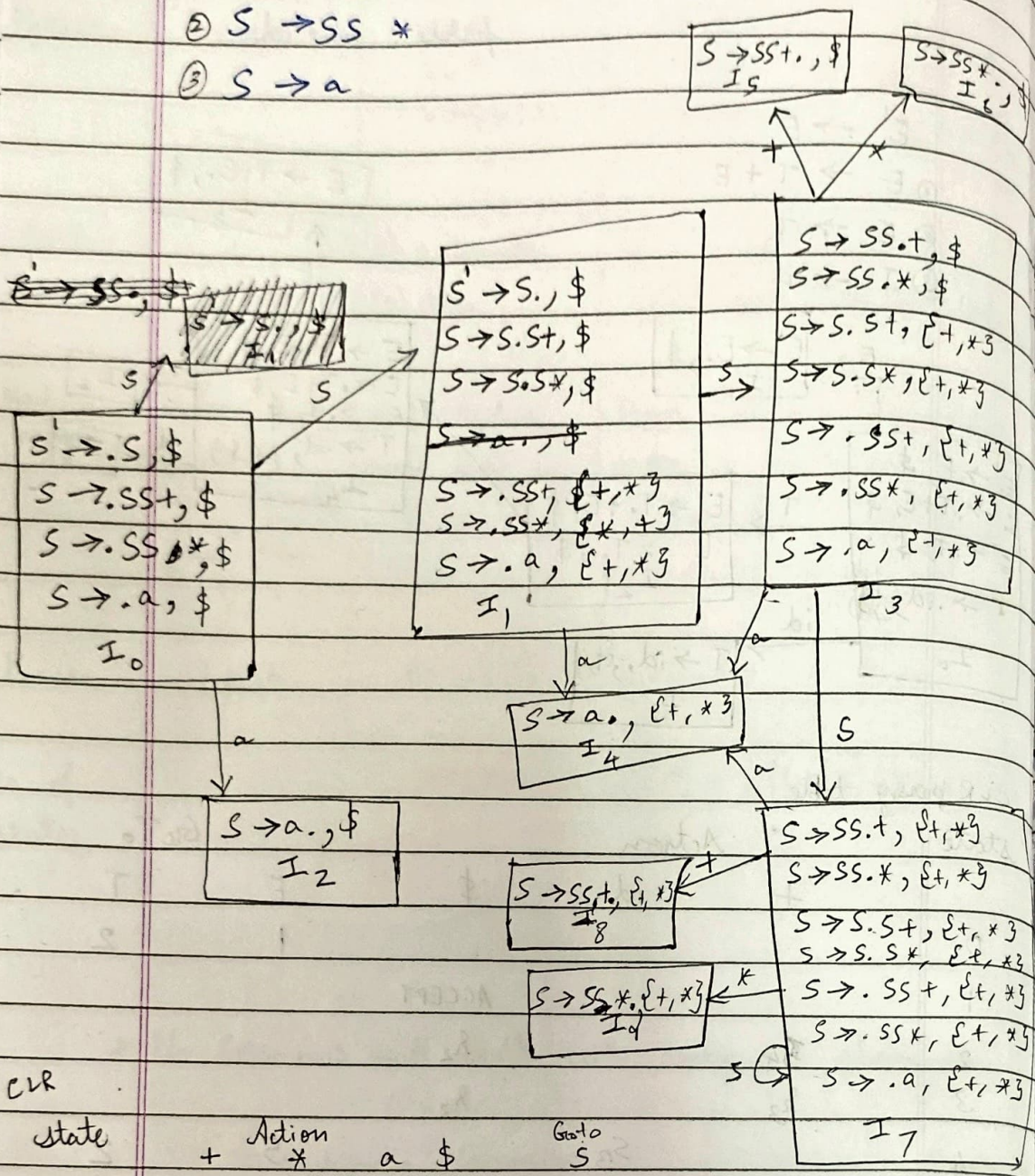
State	Action			GoTo	
	+	id	$	E	T
0				1	2
1			ACCEPT		
2	S_4		r_2		
3	r_3		r_3		
4		S_3		5	2
5			r_1		

Q $S' \to S$

① $S \to SS+$

② $S \to SS*$

③ $S \to a$



CLR

state	Action +	Action *	Action a	Action $	Goto S
0			S_2		1
1			S_4	ACCEPT	3
2				R_3	
3	S_5	S_6	S_4		7
4	R_3	R_3			
5				R_1	
6				R_2	
7	S_8	S_9			7
8	R_1	R_1			
9	R_2	R_2			

$S \to SS+ \mid SS* \mid a$ ∈LR?

Parse aa+aa*

To combine : 2 - 4
3 - 7
5 - 8
6 - 9

state	+	*	a	$	S Goto
0			S_2		1
1			S_4	accept	3
2-4	r_3	r_3		r_3	
3	S_5	S_6	S_4		7
5-8	r_1	r_1		r_1	
6-9	r_2	r_2		r_2	
7	S_8	S_9			7

Stack	Ip Buffer	Action
$0	aa+aa*	A[0,a] = S_2 ⇒ push a, 2
$0a2	a+aa*	A[2,a] =

Syntax Directed Definition
SDD
Grammar + Semantic Rules

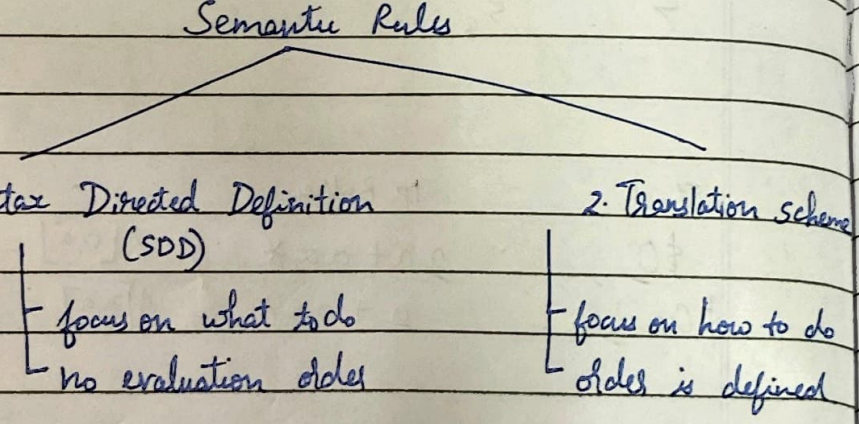
17th February, 2025

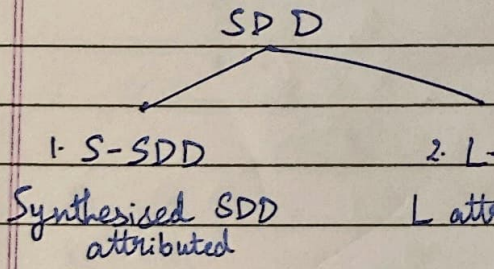
Semantic Analysis / Syntax Directed Translation

* computes additional info related to the meaning of the program once the semantic structure is known.

* Syntax Directed Translation: meaning of an IP sentence is related to its syntactic structure i.e to its Parse-Tree.

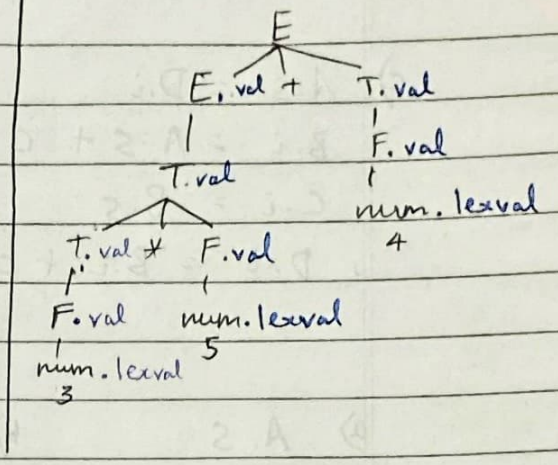
- We associate attributes to the grammar symbols
- Values for attributes are computed by semantic rules





eg1 $E \to E + T$
$E \to T$
$T \to T * F$
$T \to F$
$F \to num$
$F \to id$

$3 * 5 + 4$



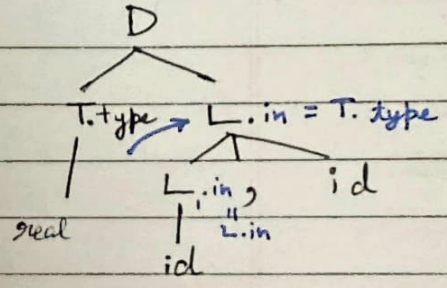
$E \to E_1 + T$	$E.val = E_1.val + T.val$
$E \to T$	$E.val = T.val$
$T \to T_1 * F$	$T.val = T_1.val * F.val$
$T \to F$	$T.val = F.val$
$F \to num$	$F.val = num.lexval$
$F \to id$	$F.val = id.lexval$

S-SDD ↓ uses only the synthesised attribute

* Synthesised attribute : if calculation of attribute always depends on its children.

eg2 $D \to T\ L$
$T \to int$
$T \to real$
$L \to L, id$
$L \to id$

real id1, id2 =>



$L_1.in$ depends on parent $L.in$
$L.in$ depends on left sibling T
} L-attributed

L - attribute : if any attribute calculation depends on its left sibling / parent

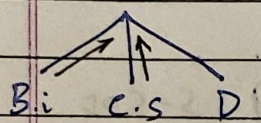
a) $A.S = B.i + C.S$

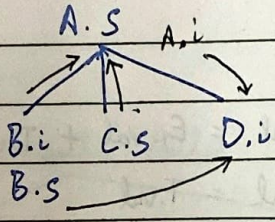
b) $A.S = B.i + C.S$
$D.i = A.i + B.S$

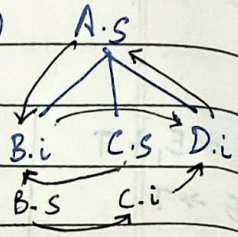
Productions
$A \rightarrow B\ C\ D$
i - inherited
S - synthesised
} attributes

c) $A.S = D.i$
$B.i = A.S + C.S$
$C.i = B.S$
$D.i = B.i + C.i$

B.i is getting value from C.S (right sib) NOT POSSIBLE

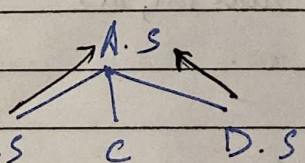
a)


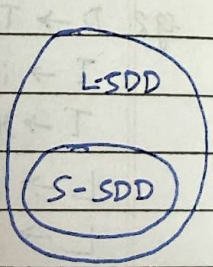
b)


c)


	S	L	Evaluation (check for cycle)
a	X	✓	✓
b	X	✓	✓
c	X	X	X
d	✓	✓	✓

d) $A.S = B.S + D.S$

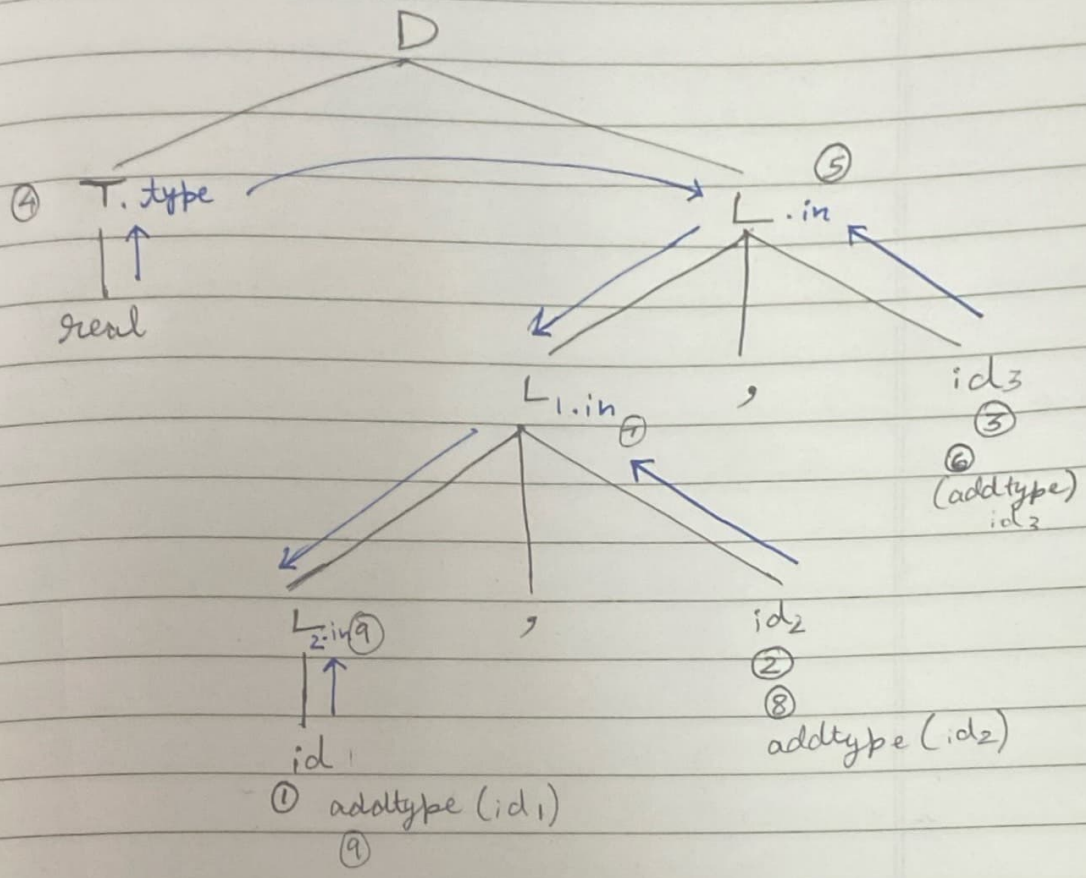




Eg 3

$D \rightarrow TL$
$T \rightarrow int$
$T \rightarrow real$
$L \rightarrow L, id$
$L \rightarrow id$

real id_1, id_2, id_3



Production	Semantic Rule
$D \rightarrow TL$	$L.in = T.type$
$T \rightarrow int$	$\{T.type = int\}$
$T \rightarrow real$	$\{T.type = real\}$
$L \rightarrow L_1, id$	$L_1.in = L.in$ addtype (id.entry, L.in)
$L \rightarrow id$	addType (id.entry, L.in)