28th Feb '17

Lecture - 29.

* LR parsing algorithm :

Input :

An input string w and an LR parsing table with function ACTION & GOTO for a grammar A.

Output :

If w is in $L(G)$, the reduction steps for ~~parse~~ :

Method :

Initially s_0 on its stack & $w\$$ in the input buffer.

Let, 'a' be the first symbol of $w\$$

```
while(1)
{
  let s be the state on top of the stack;
  if (ACTION[s, a] = shift t)
  {
    push t onto the stack;
    let a be the next input symbol;
  }
  else if (ACTION[s, a] = reduce A → β)
  {
    pop |β| symbols off the stack;
```

Let state t now be the on top of stack;
push GOTO[t, A] onto the stack;
output the production A → β;
} elseif (ACTION [s, a] = accept)
break;
else error;
}

Q. Consider an augmented grammar:

E' → E
E

1st March'17.
Lecture-30

(ii) SLR(1):

* Construction of SLR(1) or SLR parsing table:

Input : Augmented Grammar G'

Output : The SLR(0) parsing table functions ACTION & GOTO for G'

Method :

Step-1 : Construct $C = \{I_0, I_1, \cdots, I_n\}$, the collection of sets of SLR(0) items for G'.

Step-2:

Let i is constructed from I_i The parsing actions for state i are determined as follows:

a.) If $A \rightarrow \alpha .$ is in I_i then set ~~production~~ ACTION [i, a] to reduce $A \rightarrow \alpha$ for all a in ~~for~~ FOLLOW (A).

b.)

c.)

Now, the SLR(1) parsing table for previous LR(0) parsing table.

-	ACTION						GOTO		
I	id	+	*	(	)	$	E	T	F
0	S5			S4			1	2	3
1		S6				accept			
2		r2	S7		r2	r2			
3	~~r4~~	r4	r4		r4	r4			
4	S5			S4			8	2	3
5		r6	r6		r6	r6			
6	S5			S4				9	3
7	S5			S4					10
8		S6			S11				
9		r1	S7		r1	r1			
10		r3	r3		r3	r3			
11		r5	r5		r5	r5			

for I_2; $r_2 \Rightarrow E \to T$

FOLLOW(E) = { $, +,) }

for I_3; $r_4 \Rightarrow T \to F$

FOLLOW(T) = { $, +,), * }

for I_5; r_6 : $F \to id$

FOLLOW(F) = { FOLLOW(T) = { $, +,), * }

for I_9; r_1 : $E \to E + T$

FOLLOW(E) = { $, +,) }

for I_{10}; r_3 : FOLLOW(T) = { $, +,), * }

for I11 ; I5 :

1. Parse the string :

id * id + id $

from the SLR(1) parsing table.

STACK	I/P	ACTION
0	id * id + id $	Shift
0 id 5	* id + id $	Reduce F → id
0 F 3	* id + id $	Reduce T → F
0 T 2	* id + id $	Shift
0 T 2 * 7	id + id $	Shift
0 T 2 * 7 id 5	+ id $	Reduce F → id
0 T 2 * 7 F 10	+ id $	Reduce T → T * F
0 T 2	+ id $	Reduce E → T
0 E 1	+ id $	Shift
0 E 1 + 6	id $	Shift
0 E 1 + 6 id 5	$	Reduce F → id
0 E 1 + 6 F		

Stack	i/p	Action
0	id * id + id $	Shift
0 id 5	* id + id $	Reduce F → id (Pop 2)
0 F 3	* id + id $	Reduce T → F (Pop 2)
0 T 2	* id + id $	Shift
0 T 2 * 7	id + id $	Shift
0 T 2 * 7 id 5	+ id $	Reduce F → id (Pop 2)
0 T 2 * 7 F 10	+ id $	Reduce T → T * F (Pop 6)
0 T 2	+ id $	Reduce E → T (Pop 2)
0 E 1	+ id $	Shift
0 E 1 + 6	id $	Shift
0 E 1 + 6 id 5	$	Reduce F → id (Pop 1)
0 E 1 + 6 F 3	$	Reduce T → F (Pop 1)
0 E 1 + 6 T 9	$	Reduce E → E + T (Pop 6)
0 E 1	$	Accept

2nd March '17
Lecture 31

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

from LR(0) to SLR(1)

	ACTION			GOTO	
I	a	b	$	S	A
0	S3	S4		1	2
1			accept		
2	S3	S4			5
3	S3	S4			6
4	r_3	r_3	r_3		
5			r_1		
6	r_2	r_2	r_2		

for I_4; $r_3 = A \to b$

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

for I_5; $r_1 = S \to AA$

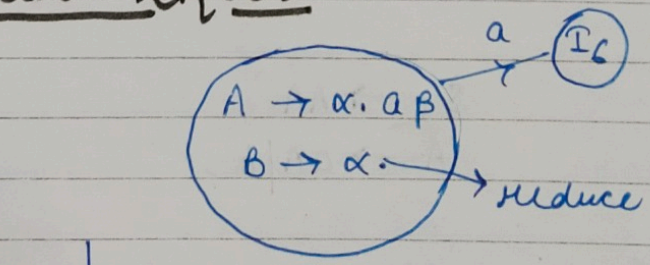
FOLLOW(S) = { $ }

for I_6; $r_2 = A \to aA$

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

* Conflicts in LR(0) :

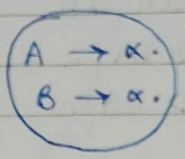
1.) SR conflict :



	a	b	$
5	S6/r_3	r_3	r_3

SR conflict

2. RR conflict :



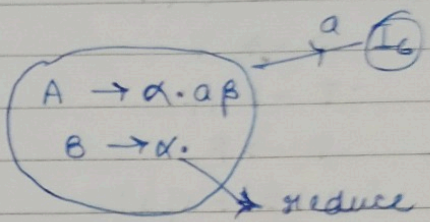
	a	b	$
5	r_1/r_3	r_1/r_3	r_1/r_3

2 different actions on a terminal.

conflicts in SLR(1) :

1. SR conflict :

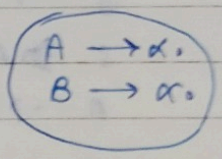
	a	b	$
5	S_6/r_3	r_3	r_3



FOLLOW (B) = {a}

on finding FOLLOW (B) & solving by SLR(1) method we still get shift & reduce both operations on terminal 'a'. Hence, SR conflict still present.

2. RR conflict :



if FOLLOW (A) ∩ FOLLOW (B) ≠ ϕ
then RR conflict.

Q. S → A | a
A → a

LL(1)
LR(0)
SLR(1)

S → A | a FIRST(A) = {a}
A → a FIRST(a) = {a}
So, not LL(1) grammar

$$
$ $
$ $

2. $S \to aA \mid aB$
$A \to bA \mid c$
$B \to bB \mid c$

3. $S \to (L) \mid a$
$L \to L,S \mid S$

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

5. $E \to T+E \mid T$
$T \to i$

6. $E \to E+T$
$E \to T$
$T \to TF$
$T \to F$
$F \to F* \mid a \mid b$

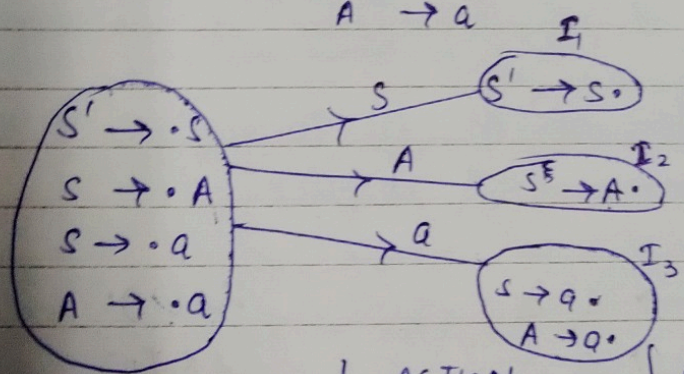
---

1. LL(1)?

		FIRST	FOLLOW
$S \to A \mid a$		{a}	{$}
$A \to a$		{a}	{$}

$\Rightarrow$ LL(1) not poss.

LR(0) : $S' \to S$
$S \to A \mid a$
$A \to a$



NOT LR(0)

I	ACTION a	ACTION $	GOTO S	GOTO A
0	S3		1	2
1		accpt		
2	r4	r1		
3	r0/r3	r1/r3		

SLR(1):

I	ACTION a	ACTION $	GOTO S	GOTO A
0	S3		1	2
1		accept		
2		H_4		
3		H_2/H_3		

not SLR(1)

2

$S \rightarrow dA \mid aB$

$A \rightarrow bA \mid c$

$B \rightarrow bB \mid c$

LL(1):

	FIRST	FOLLOW
$S \rightarrow dA \mid aB$	{d, a}	{$}
$A \rightarrow bA \mid c$	{b, c}	{$, b, c}
$B \rightarrow bB \mid c$	{b, c}	{$, b, c}

not LL(1)

LR(0):

$S' \rightarrow S$

$S \rightarrow dA$

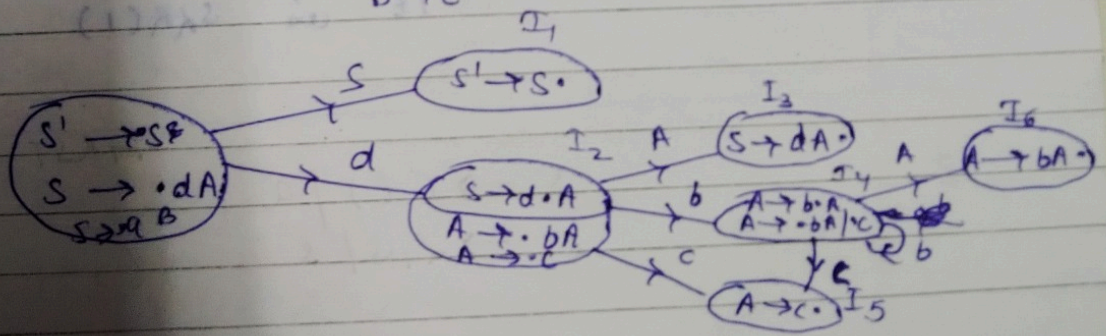
$S \rightarrow aB$

$A \rightarrow bA$

$A \rightarrow c$

$B \rightarrow bB$

$B \rightarrow c$



	ACTION					GOTO	
I	a	b	c	d	$	S	A
0				s2		1	
1					accpt		
2			s4	s5			3
3	r_1	r_1	r_1	r_1	r_1		
4			s4	s5			6
5	r_4	r_4	r_4	r_4	r_4		
6	r_3	r_3	r_3	r_3	r_3		

It is LR(0)

SLR(1) :

	ACTION					GOTO	
I	a	b	c	d	$	S	A
0				s2		1	
1					accpt		
2			s4	s5			3
3					r_1		
4			s4	s5			6
5	r_4	r_4	r_4				
6	r_3	r_3	r_3				

It is SLR(1).

3.

	FIRST	FOLLOW
$S \rightarrow (L) \mid a$	{(, a}	{$,), ,}
$L \rightarrow L, S \mid S$	{(, a}	{), ,}

not LL(1) : left recursive

LR(0) :

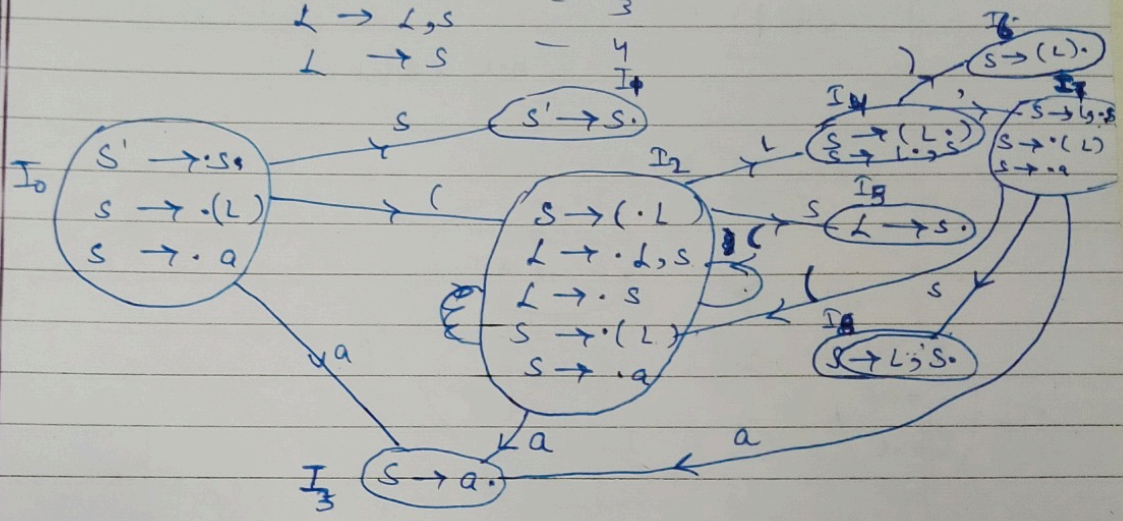
$S' \to S$

$S \to (L)$ — 1

$S \to a$ — 2

$L \to L,S$ — 3

$L \to S$ — 4



	ACTION					GOTO	
I	(	)	a	,	$	S	L
0	S2		S3			1	
1					accept		
2	S2		S3			5	4
3	r2	r2	r2	r2	r2		
4		S6		S7			
5	r4	r4	r4	r4	r4		
6	r4	r4	r4	r4	r4		
7	S2		S3			8	
8	r3	r3	r3	r3	r3		

It is LR(0).

SLR(1) : ∵ It is LR(0)

So, it is also SLR(1).

Date: / /
Page:

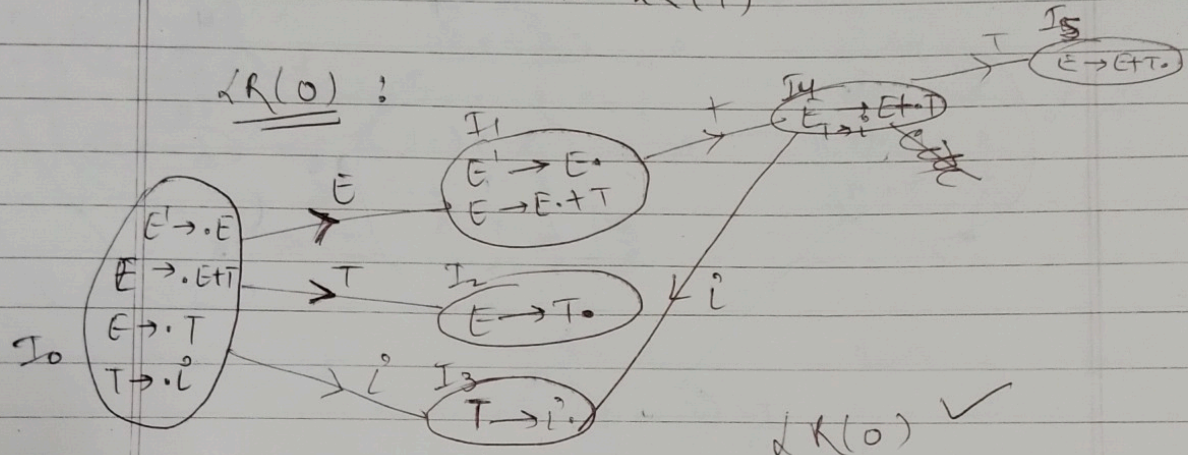
4 $E \to E+T \mid T$
$T \to i$

$\Rightarrow E' \to E$
$E \to E+T \quad \{i\} \quad \{\$, +\}$
$E \to T \quad \{i\} \quad \{\$, +\}$
$T \to i \quad \{i\} \quad \{\$, +\}$

LL(1) :

not LL(1)

LR(0) :



LR(0) ✓
no conflict

I	i	+	$	E	T
0	S3			1	2
1		S4	accpt		
2	r2	r2	r2		
3	r3	r3	r3		
4	S3			5	
5	r1	r1	r1		

SLR(0) :

✓

example

$S \to AaAb$ — ①
$\quad | BbBa$ — ②
$A \to \epsilon$ — ⑤
$B \to \epsilon$ — ④

LL(1) ✓
LR(0) ✗
SLR(1) ✗

I_0

$S' \to \cdot S$
$S \to \cdot AaAb$
$\quad | \cdot BbBa$
$A \to \cdot$
$B \to \cdot$

two reduce move in same state

SLR parsing table

	a	b	$
0	r3/r4	r3/r4	

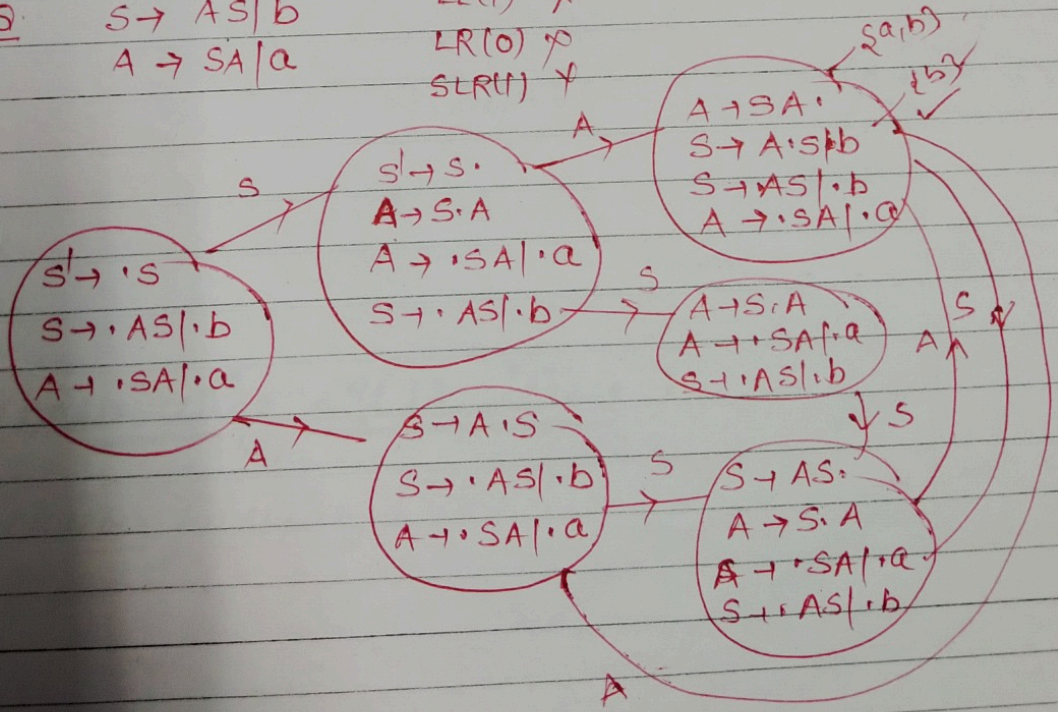
not SLR(1)

Ambiguous grammar

{a,b} {b}

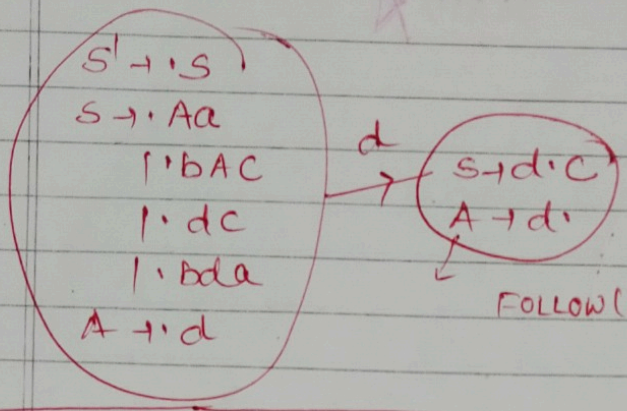
Q. $S \to AS \mid b$
$A \to SA \mid a$

LL(1) ✗
LR(0) ✗
SLR(1) ✓



$S \to Aa \quad \{d\}$
$\quad | bAc \quad \{b\}$
$\quad | dC \quad \{d\}$
$\quad | bda$
$A \to d$

LL(1) ✗
LR(0) ✗
SLR(1) ✗



FOLLOW(A) = {a, c}

	a	b	c	d	$
6			S/R		

LR(0) → any · on RHS

LALR(1) CLR(1)

LR(1) items = LR(0) items + lookahead

$S \to \cdot aA, a/b$

$S \to aA \cdot, c/d$

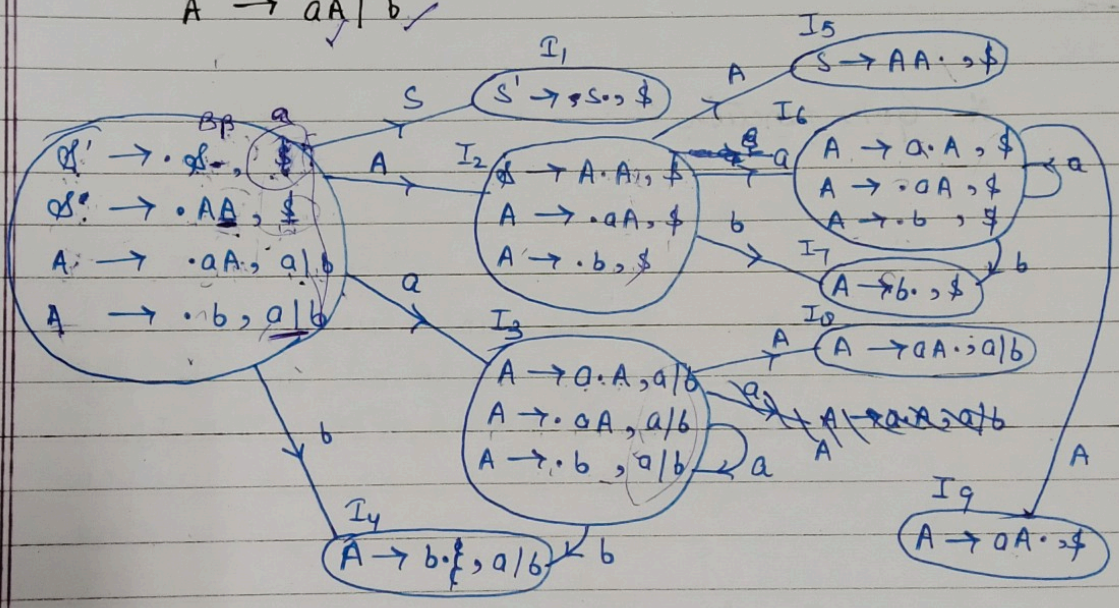
3rd March '17
Lecture - 32

Cannonical LR (CLR) :

Consider an augmented grammer :

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

compute LR(1) items.



4th March '17
Lecture - 33

Constructing LR(1) items :

* Closure function :

```
closure (I)
{
  repeat
    for (each item [A → α·Bβ, a] in I)
      for (each production B → γ in G')
        for (each terminal b in FIRST (βa))
          add [B → ·γ, b] to set I;
      until no more items are added to I;
}
```

* **GOTO function :**

```
GOTO (I, X)
{
  Initialize J to be the empty set;
  for (each item [A → α·Xβ, a] in I)
      add item [A → αX·β, a] to set J;
  return (closure (J));
}
```

* Construction of canonical LR(1) parsing table :

Input : Augmented Grammar G'

Output : The CLR(1) parsing table function ACTION & GOTO for G'

Method :

Step-1 : Construct $C = \{I_0, I_1, \cdots I_n\}$, the collection of sets of CLR(1) items for G'.

Step-2 :

State i is constructed from I_i. The parsing actions for state i are determined as follows:

a.) If $[A \to \alpha . a\beta, b]$ is in I_i and GOTO$(I_i, a) = I_j$ then set ACTION$[i, a]$ = "shift j".

b.) If $[A \to \alpha ., a]$ is in I_i, $A \neq S'$ then set ACTION$[i, a]$ to "reduce $A \to \alpha$".

c.) If $[S' \to S., \$]$ is in I_i, then set ACTION$[i, \$]$ to "accept".

Step-3 :

The GOTO transition for state i are constructed for all non-terminals A using the rule :

if GOTO$(I_i, A) = I_j$ then
GOTO$(I_i, A) = j$

Step-4 :

All entries not defined by rules 2 & 3 are made "error".

Step-5 :

The initial state of the parser is constructed from the set of items containing $[S' \to .S, \$]$.

	ACTIONS			GOTO	
I	a	b	$	S	A
0	S3	S4		1	2
1			accept		
2	S6	S7			5
3	S3	S4			8
4	r3	r3			
5			r1		
6	S6	S7			9
7			r3		
8	r2	r2			
9			r2		

iv. LALR (Look-ahead LR):

* Construction of LALR parsing table:

for the previous CLR(1) grammar states 3&6, 4&7, 8&9 have same LR(0) productions but different look ahead symbols. So, for LALR combine them as follows:

(only for shift productions not for reduce)

I_3, I_6
I_4, I_7
I_8, I_9

I_{36} : $A \to a \cdot A$, a|b|$
$A \to \cdot aA$, a|b|$
$A \to \cdot b$, a|b|$

I_{47} : $A \to b \cdot$, a|b|$

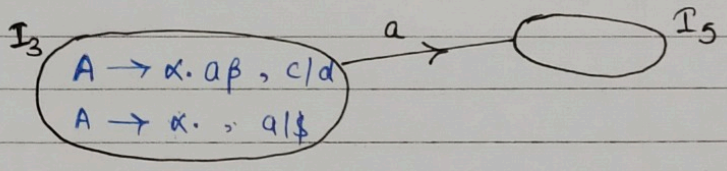
I_{89} : $A \to aA \cdot$, a|b|$

	ACTION			GOTO	
I	a	b	$	S	A
0	S_{36}	S_{47}		1	2
1			accept		
2	S_{36}	S_{47}			5
36	S_{36}	S_{47}			89
47	r_3	r_3	r_3		
5			r_1		
89	r_2	r_2	r_2		

6th March '17 Lecture-34

Conflicts in CLR(1) & LALR(1):

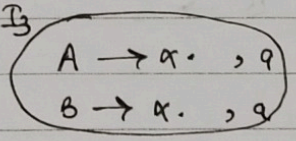
CLR:



S-R conflict

I	a
3	S5/r1

Shift-reduce conflict



RR-conflict

I	a
3	r1/r2

→ Reduce-reduce conflict

If it is a reduce operation & look-aheads of the productions in any item match then there will be RR conflict

* **Note:**

1.) If any grammar is not CLR then it will definately not be a LALR.

2) If any grammar is CLR then it may or

may not be LALR.

Example :

I_3: $A \to \alpha\cdot, a$; $B \to \beta\cdot, b$

I_5: $A \to \alpha\cdot, b$; $B \to \beta\cdot, a$

It is CLR(1)

for LALR :

I_{35}: $A \to \alpha\cdot, a/b$; $B \to \beta\cdot, a/b$

It is not LALR because there will be RR-conflict. Because A & B will perform reduce opn & they both will perform reduce on the same a & b So, RR-conflict not LALR

Q. consider a grammar :

$S \to AaAb \mid BbBa$

$A \to \epsilon$

$B \to \epsilon$

LL(1)
LR(0)
SLR(1)
CLR(1)
LALR(1)

- LL(1)

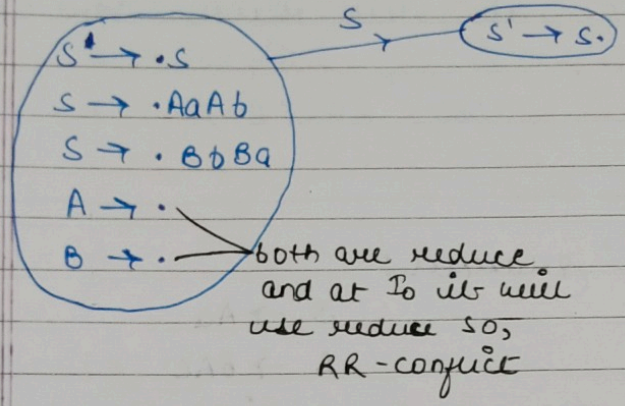
	FIRST	FOLLOW
$S \to AaAb$	{a, b}	{$}
$S \to BbBa$		
$A \to \epsilon$	{a}	{a, b}
$B \to \epsilon$	{b}	{a, b}

LL(1)

Date: __/__/__
Page:

- **LR(0):** Augmented grammar:

$S' \to S$
$S \to AaAb$ – 1
$S \to BbBa$ – 2
$A \to \epsilon$ – 3
$B \to \epsilon$ – 4



∴ Not LR(0).

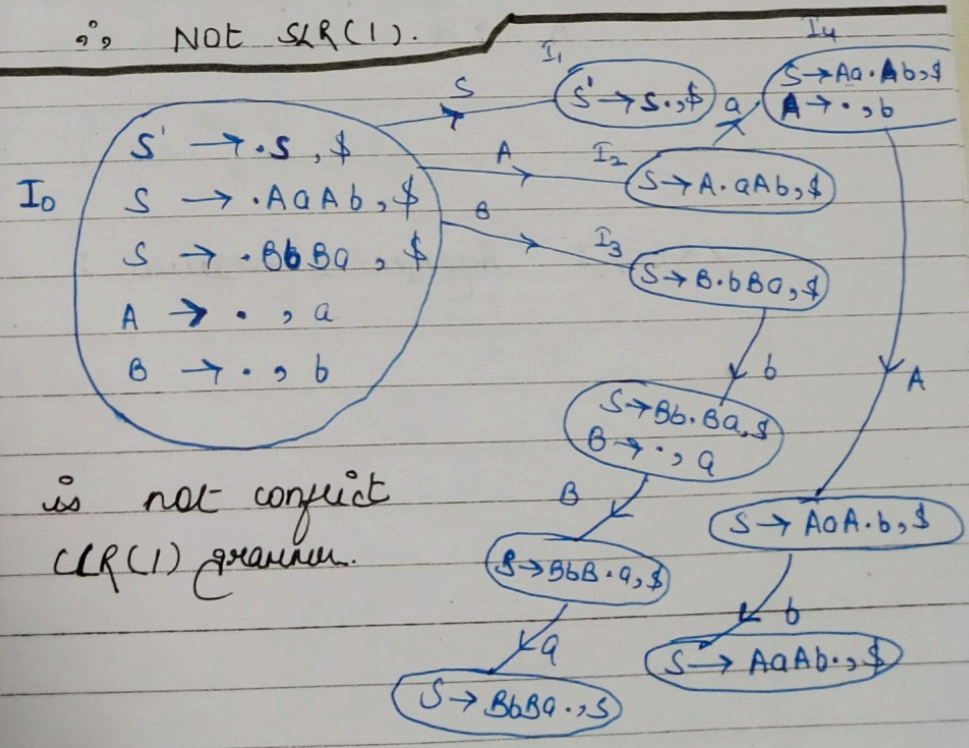
- **SLR(1):**

$A \to \cdot\epsilon$ FOLLOW(A) = {a, b} r3 at a,b
$B \to \cdot\epsilon$ FOLLOW(B) = {a, b} r4 at a,b

again R-R conflict at ~~at~~ at I_0 item a, b will contain r3 & r4 both

∴ Not SLR(1).

- **CLR(1):**



∴ There is not conflict
∴ It is CLR(1) grammar.

- LALR(1) :

∵ There are no common productions with different look ahead values.

So, it will remain same as CLR(1).

∴ It is also LALR(1).

Q. Consider the grammar:

LL(1)
LR(0)
SLR(1)
CLR(1)
LALR(1)

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

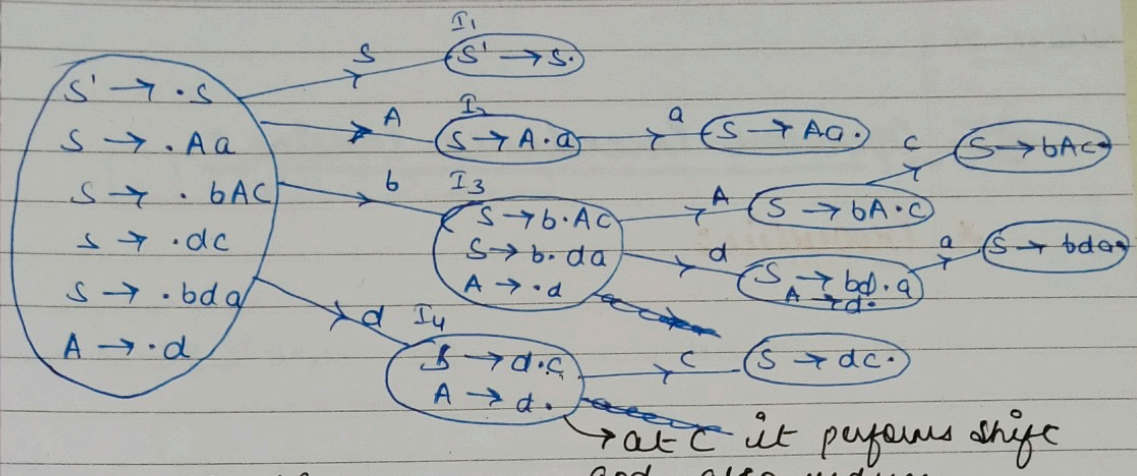
LL(1):

	FIRST	FOLLOW
$S \to Aa \mid bAc \mid dc \mid bda$	{d, b}	{$}
$A \to d$ (same entries)	{d}	{a, c}

It is not LL(1).

LR(0): Augmented grammar:

$S' \to S$
$S \to Aa$ — 1
$S \to bAC$ — 2
$S \to dc$ — 3
$S \to bda$ — 4
$A \to d$ — 5



not LR(0) :

SLR(1) :

$A \to d.$

FOLLOW(A) = { a, c }

→ at c it will perform reduce

& in the above diagram it also performs shift at c

∴ SR conflict ⇒ NOT SLR(1)

CLR(1) :

$S' \to .S, \$$

$S \to .Aa, \$$

$S \to .bAC, \$$

$S \to .dc, \$$

$S \to .bda, \$$

$A \to$

7th March '17

Lecture - 35.

Operator Precedence Parsing:

* Properties:

a.) NO production on the right side contains ϵ

b) NO production on right side contains 2 adjacent non-terminals.

Example: $E \rightarrow EAE \mid id$

$A \rightarrow + \mid *$

It is not operator grammar bcz (ii)nd condition is not satisfied "E" & "A" non-terminals are adjacent.

$E \rightarrow E+E \mid E*E \mid id$

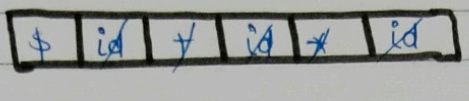
now, it is operator grammar

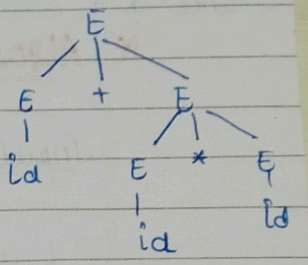
→ Operator Precedence Relation Table:

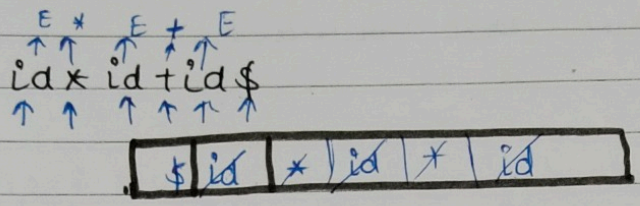
$E \rightarrow E+E \mid E*E \mid id$ (f)

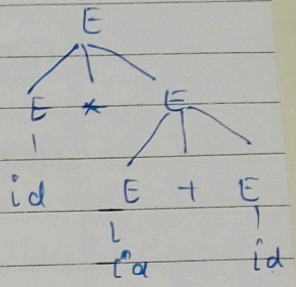
	id	+	*	$ (g)
id	-	$\gtrdot$	$\gtrdot$	$\gtrdot$
+	$\lessdot$	$\gtrdot$	$\lessdot$	$\gtrdot$
*	$\lessdot$	$\gtrdot$	$\gtrdot$	$\gtrdot$
$	$\lessdot$	$\lessdot$	$\lessdot$	(accept)

- id + id * id $

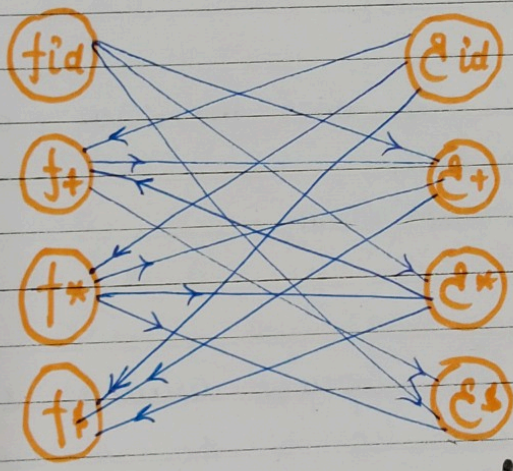








* Operator precedence graph:



* Operator precedence function table:

	+	*	id	$
f	2	4	4	0
g	1	3	5	0

~~f$~~ f+ = 2
g* = 3
g* has high value than f+ so, * high precedence than +

fid → g* → f+ → g+ → f$
gid → f* → g* → f+ → g+ → f$
} longest paths

8th March'17

Lecture-36

* Algorithm :

Input : The precedence relation from some operator precedence grammer & input string terminals from that grammer.

Output :

Method :

```
While(1)
{
  if only $ is on the stack & only $ is on
  the input then
      accept and break;
  else
  {
    let 'a' be the topmost terminal symbol on
    the stack and let 'b' the current input
    symbol;
    if a <· b or a ≐ b then shift 'b' onto the
    stack;
    else if a ·> b then (/* reduce */)
        repeat pop the stack;
        until the top stack terminal is related by <·
        to the terminal most recently poped;
    else
        call the error correcting routine;
  }
}
```

* Algorithm for finding precedence function for a table :

Step-1 :

Create symbols f_a & g_a for each a; i.e; terminal or $.

Step-2 :

Partition the created symbol into as many group as possible in such a way that $a \doteq b$ then f_a & g_b in the same group.

Step-3 :

Created a direct graph whose nodes are the groups found in step 2. For any a & b if $a \lessdot b$, place an edge from the group of g_b to group f_a.

Step-4 :

If the graph constructed in step 3 has a cycle then no precedence function exist. If there is no cycle, let $f(a)$ be the length of the longest path beginning from the group of f_a and let $g(a)$ be the longest path from the group of g_a.

Example :

$P \rightarrow SR \mid S$
$R \rightarrow bSR \mid bS$
$S \rightarrow Wbs \mid W$
$W \rightarrow L * W \mid L$
$L \rightarrow id$

It is not operator grammer bcz 1st & 2nd production have 2 adjacent non-terminals.

Now, making it an operator grammer.

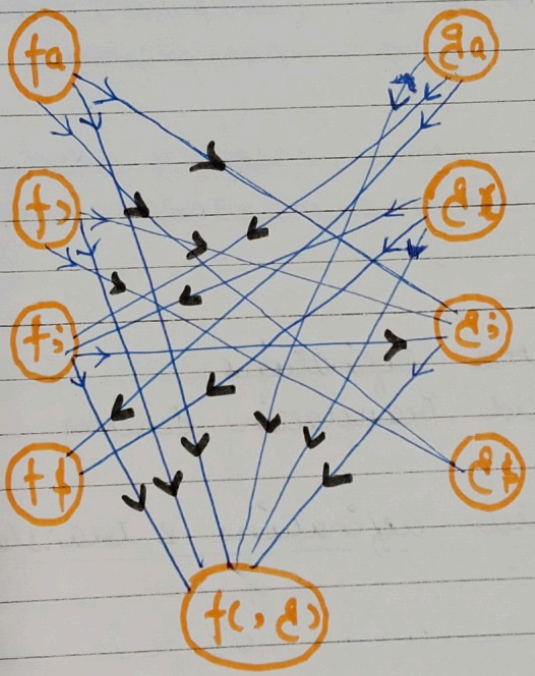
$P \rightarrow SbSR \mid SbS \mid S$
$P \rightarrow SR$

$P \rightarrow SbP \mid SbS \mid S$
$S \rightarrow WbS \mid W$
$W \rightarrow L * W \mid L$
$L \rightarrow id$

	id	*	b	$
id	-	$\dot{>}$	$\dot{>}$	$\dot{>}$
*	$\lessdot$	$\lessdot$	$\dot{>}$	$\dot{>}$
b	$\lessdot$	$\lessdot$	$\lessdot$	$\dot{>}$
$	$\lessdot$	$\lessdot$	$\lessdot$	accept

Q. Convert the given relation table into function table :

	a	(	)	;	$
a			⋗	⋗	⋗
(	⋖	⋖	≐	⋖	
)			⋗	⋗	⋗
;	⋖	⋖	⋗	⋗	
$	⋖	⋖			



	a	(	)	;	$
f	2	0	2	2	0
g	3	3	0	1	0

f(a) ⇒ fa → gi → f(,g)
f()) ⇒ f) → g; → f(,g)
f(;) ⇒ f; → g; → f(,g)
~~f($) ⇒~~
g(a) ⇒ ga → f; → gi → f(,g)
g(() ⇒ g(→ f; → gi → f(,g)
g(;) ⇒ gi → f(,g)