$T \rightarrow T,S \mid S$

$T \rightarrow ST'$

$T' \rightarrow ,ST' \mid \epsilon$

Compute the FIRST & FOLLOW for

$S \rightarrow a \mid b \mid (T)$

$T \rightarrow ST'$

$T' \rightarrow ,ST' \mid \epsilon$

FIRST(S) = {a, b, (}

FISTT(T) = FIRST(S) = {a, b, (}

FIRST(T') = {, , ε}

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

$T \rightarrow \alpha B \beta$ (ST')

FOLLOW(S) = FIRST(T') − ε ∪ FOLLOW(T)

{, , ε} − {ε} ∪ FOLLOW(T)

{,} ∪ FOLLOW(T) ⇒ {,)}

$T' \rightarrow , \alpha B \beta$ (, ST')

FOLLOW(S) = FIRST(T') − ε ∪ FOLLOW(T')

{,} ∪ FOLLOW(T') ⇒ {,)}

FOLLOW(T) = {)}

FOLLOW(T') = {)}

In $T \rightarrow ST'$ after T' there is nothing ⇒ FOLLOW(T')

FOLLOW(T)

$T \rightarrow \alpha B \beta$ (ST')

FOLLOW(T) = FIRST(ε) − {ε} ∪ FOLLOW(T)

{ε} − {ε} ∪ {)}

{)}

	FIRST(α)	PPT
S → a	a	M[S, a]
S → b	FIRST(b) = b	M[S, b]
S → (T)	FIRST((T)) = (	M[S, b]
T → ST'	FIRST(ST') = {a, b, (}	M[T, a], M[T, b], M[
T' → , ST'	FIRST(, ST') = ,	M[T', ;]
T' → ϵ	FOLLOW(T') = {)}	M[T',)]

terminals	a	b	(	)	,
S	S → a	S → b	S → (T)		
T	T → ST'	T → ST'	T → ST'		
T'				T' → ϵ	T' → , ST'

Stack	input	o/p
$ S	(b) $	S → (T)
$) T (	(b) $	T → ST'
$) T' S	b) $	S → b
$) T' b	b) $	T' → ϵ
$)	) $	
$	$	successful

checking whether it is LL(1) or not

$S \rightarrow a \mid b \mid (T)$

$T \rightarrow T,S \mid S$

$S \rightarrow (T) \quad \rightarrow S$

$S \rightarrow (S) \quad S \rightarrow b$

$\rightarrow (b)$

2) Construct LL(1) Grammer

Predictive parser i.e recursive descent parser needs no backtracking, can be constructed for a cls of Grammer called LL(1). The first L indicates the scanning the ilp from Left to right & 2nd L for producing a left most derivation & one for using one ilp symbol of look ahead at each step to make parsing acting decision

unique entries if of LL(1)
multiple entries if is not LL(1)

Bottom Up Parsing: (LR(1))

Constructing a parse tree from leaf node to the root node is called bottom up parsing

ex: $E \rightarrow E+T/T$

$T \rightarrow T*F/F$

$F \rightarrow (E)/id$

"id + id"

1) id * id

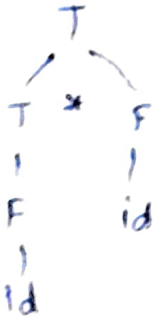
2) F * id
   |
   id

3) T * id
   |
   F
   |
   id

4) T * F
   | |
   F id
   |
   id

5)

6)



** Handle Pruning (2m)

Bottom up parsing during a left to right scan of the constructs a right most derivation (RMD) in reverse

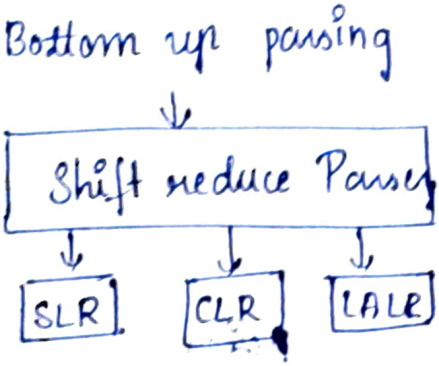
A handle is a substring that matches the body of a production & whose reduction represents one step along the reverse of RMD

For the above Qn

Right sentential form	Handle	Reduced production
id * id	id	F → id
F * id	F	T → F
T * id	id	F → id
T * F	T * F	T → T * F
T	T	E → T

E - Start symbol of the grammar

Bottom up parsing techniques



SLR → Simple LR Parsers

CLR → Canonical LR Parser

LALR → look ahead LR Parser.

Shift Reduce Parser

The following actions are performed in shift reduce parser.

i) Shift:
The next input symbol is shifted onto the top of the stack.

ii) reduce:
The parser replaces the handle within a stack with a non terminal.

iii) Accept:
The parser announces the successful completion of parsing

iv) Error:
The parser discovers a error & calls an error recovery routine

Q) Construct shift Reduce Parser for the following

$E \rightarrow E + T / T$

$T \rightarrow T * F / F$

$F \rightarrow (E) / id$

ip : id * id

Stack	I/p	Action
$ id	id * id $	shift
$ id	* id $	F → id (reduce)
$ F	* id $	T → F (reduce)
$ T	* id $	shift
$ T *	id $	shift
$ T * id	$	F → id (reduce)
$ T * F	$	T → T * F (reduce)
$ T	$	T → E (reduce)
$ E	$	Accept

Q) Construct shift Reduce Parser for the following

$M \rightarrow R + R / R + C / R$

$R \rightarrow C$

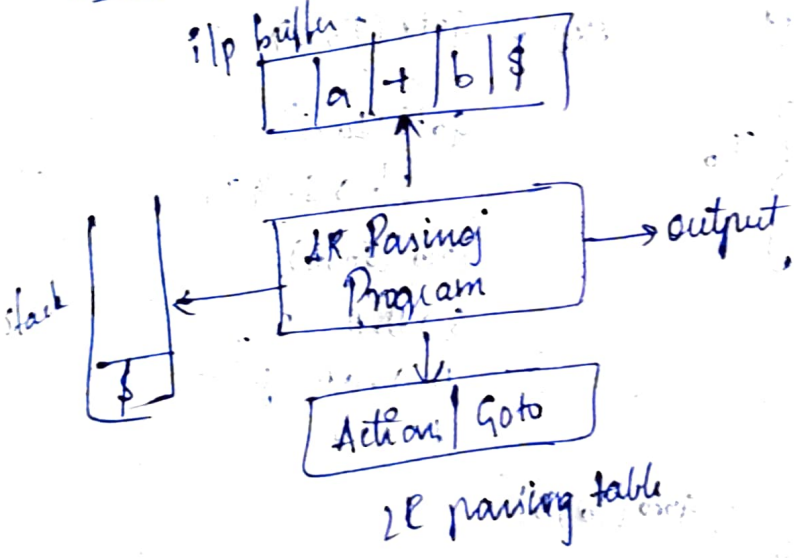
string "C + C"

Stack	id	Action
$	c+c$	shift
$c	+c$	R→c (reduce)
$R	+c$	shift
$R+	c$	shift
$R+c	$	M→R+c
$M	$	Accept

Conflicts during shift Reduce Parser (2m)

1) shift reduce conflict
2) Reduce-Reduce conflict

LR Parsing:



LR parsing table

SLR (Simple LR Parser)

Algorithm for SLR Parser.

St1: Augmented grammer(G')

If G is a grammer with start symbol S then the augmented grammar (G') with a new start symbol S'

i.e S'→S

Step 1: LR(0) items

An LR(0) item of a grammar G with a dot at position on the right hand side

ex. $A \to XYZ$

$A \to .XYZ$ (or) $A \to X.YZ$ (or) $A \to XY.Z$ (or) $A \to XYZ.$

Step 3: closure set of items(I)

$A \to \alpha . B\beta$ 2) GOTO(I0, X)

$B \to .\gamma$

Step 4: SLR passing table

5) Construct SLR passing table for given grammar

$S \to AA$

$A \to aA$

$A \to b$

Step 1: Augmented Grammar G'

$S' \to S$

Step 2: LR(0) items

$S' \to .S$

$S \to .AA$

$A \to .aA$

$A \to .b$

} I_0

Step 3: goto(I_0, X)

Goto(I_0, S)

$S' \to S.$ } I_1

Goto(I_0, A)

$S \to A.A$

$A \to .aA$

$A \to .b$

} I_2

goto(I_0, a)

$A \to a.A$

$A \to .aA$

$A \to .b$

} I_3

goto(I_0, b)

$A \to b.$ } I_4

goto(I_2, A)

$S \to AA.$ } I_5

goto(I_2, a)

$A \to a.A$

$A \to .aA$

$A \to .b$

} I_3

goto (I_2, b)

$A \to b. \$ IA$

goto (I_3, b)

$A \to b. \$ IA$

goto (I_3, A)

$A \to aA. \$ I_6$

goto (I_3, a)

$A \to a.A$
$A \to .aA$
$A \to .b$
} I_3

Step 4: SLR parsing table

States	a	b	$	Goto S	Goto A
I_0	S_3	S_4	Error	1	2
I_1	Error	Error	Accept	Error	Error
I_2	S_3	S_4	Error	Error	5
I_3	S_3	S_4	Error	Error	6
I_4	r_3	r_3	r_3	Error	Error
I_5	Error	Error	r_1	Error	Error
I_6	r_2	r_2	r_2	Error	Error

To perform reduce operation the given production should be in

Reduce A. $A \to b$

$S' \to S. \$ I_1$
$A \to b. \$ I_4$
$S \to AA. \$ I_5$
$A \to aA. \$ I_6$

Given Question

(1) $S \to AA$
(2) $A \to aA$
(3) $A \to b$

In order to perform reduce operation find FIRST, FOLLOW for the given grammer

FIRST(S) = FIRST(A) = {a, b}

FIRST(A) = {a, b}

FOLLOW(S) = {$}

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

S → AA

S → A a A

FOLLOW(A) = a

S → A A

b

S → A b

FOLLOW(A) = b

3) A → b · } I_4

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

4 on $ = r_3

4 on a = r_3

4 on b = r_3

1) S → AA · } I_5 FOLLOW(S) = {$}

5 on $ = r_1

2) A → aA · } I_6 FOLLOWS(A) = {$, a, b}

6 on $ = r_2

6 on a = r_2

6 on b = r_2

Step 5 derive the string abb

stack		
0 :	abb$	S3 (0 on a)
0a3	bb$	S4 (3 on b)
0a3b4	b$	r_3 (A → b) pop this elt in RHS i.e 2*1 = 2
0a3A	b$	r_2 (A → aA) 2*2 = 4 elts pop
(goto A) 0A1	$	S4 (2 on b)
0A2b4	$	r_3 (A → b) 2*1 = 2
0A2A5	$	r_1 (S → AA) 2*2 = 4
0S1	$	Accept

17/10/23.

Construct SLR, CLR, LAR for the following.

$S \to CC$, $C \to cC$, $C \to d$

Step 1: Augumented Grammar

$S' \to .S$
$S \to .CC$
$C \to .cC$
$C \to .d$
} I_0

goto(I_0, S)

$S' \to S.$ } I_1

goto (I_0, C)

$S \to C.C$
$C \to .cC$
$C \to .d$
} I_2

goto (I_0, c)

$C \to c.C$
$C \to .cC$
$C \to .d$
} I_3

goto(I_0, d)

$C \to d.$ } I_4

goto (I_2, C)

$S \to CC.$ } I_5

goto (I_2, c)

$C \to c.C$
$C \to .cC$
$C \to .d$
} I_3

goto (I_2, d)

$C \to d.$ } I_4

goto (I_3, C)

$C \to cC.$ } I_6

goto (I_3, c)

$C \to c.C$
$C \to .cC$
$C \to .d$
} I_3

goto (I_3, d)

$C \to d.$ } I_4

SLR parsing table

State	Action			Goto
	c	d	$	S
I_0	S_3	S_4	Error	1
I_1	Error	Error	Accept	Error
I_2	S_3	S_4	Error	Error
I_3	S_3	S_4	Error	Error
I_4	r_3	r_3	r_3	Error
I_5	Error	Error	r_1	Error
I_6	r_2	r_2	r_2	Error

To perform reduce operation the production should be

$A \to d.$

(3) $C \to d.$ } I_4

(1) $S \to CC.$ } I_5

(2) $C \to cC.$ } I_6

(1) $S \to CC.$

(2) $C \to cC$

(3) $C \to d$

In order to perform reduce operation FIRST of follow

for given grammar

FOLLOW(S) = {$}

FOLLOW(C) = {c, d, $}

stater	input	Remarks
0	cdd$	s_3
0c3	dd$	s_4
0c3d4	d$	r_3 C→d
0C32C6	d$	r_2, C→cC
0C2	d$	s_4
0C2d4	$	r_3 C→d
0C2C5	$	r_1 S→CC
0S1	$	Accept

☆ CLR (Canonical LR Parser)

S → CC

C → cC

C → d

On CLR given grammar should be in form of

A → α.Bβ, a

B → .γ, first(βa)

S' → .S, $

S → .CC, $

C → .cC, c/d

C → .d, c/d

A → α.Bβ, a

S → .S, $

S → .CC, $

C → .cC, c/d

C → .d, c/d

I_0

goto (I0, S)

$S' \to S.,\ \$$ } I_1

goto (I0, C)

$S \to C.C,\ \$$

$C \to .cC,\ \$$ } I_2

$C \to .d,\ \$$

goto (I0, c)

$C \to c.C,\ c/d$

$C \to .cC,\ c/d$ } I_3

$C \to .d,\ c/d$

goto (I0, d)

$C \to d.,\ c/d$ } I_4

goto (I2, C)

$S \to CC.,\ \$$ } I_5

goto (I2, c)

$C \to c.C,\ \$$

$C \to .cC,\ \$$ } I_6

$C \to .d,\ \$$

goto (I2, d)

$C \to d.,\ \$$ } I_7

goto (I3, C)

$C \to cC.,\ c/d$ } I_8

goto (I3, c)

$C \to c.C,\ c/d$

$C \to .cC,\ c/d$ } I_3

$C \to .d,\ c/d$

goto (I3, d)

$C \to d.,\ c/d$ } I_4

goto (I6, C)

$C \to cC.,\ \$$ } I_9

goto (I6, c)

$C \to c.C,\ \$$

$C \to .cC,\ \$$ } I_6

$C \to .d,\ \$$

goto (I6, d)

$C \to d.,\ \$$ } I_7

	c	d	$	S	C
I_0	S_3	S_4		1	2
I_1			Accept		
I_2	S_6	S_7			5
I_3	S_3	S_4			8
I_4	r_3	r_3			
I_5			r_1		
I_6	S_6	S_7			9
I_7			r_3		
I_8	r_2	r_2			
I_9		r_3	r_2		

(3) $C \to d\cdot, c/d$ } I_4

(1) $S \to CC\cdot, \$$ } I_5

(2) $C \to d\cdot, \$$ } I_7

(2) $C \to cC\cdot, c/d$ } I_8

(2) $C \to cd\cdot, \$/d$ } I_9

Input string dd

State	I/P buffer	Remarks
0	dd$	S4
04	d$	r3, C→d
02	d$	S7
027	$	r3 C→d
025	$	r1 S→CC
01	$	Accept

LALR (Look ahead LR Parser)

if look ahead are diff we can combine

	c	d	$	S	C
0	S36	S47		1	2
1			accept		
2	S36	S47			5
36	S36	S47			89
47	r3	r3	r3		
5			r1		
89	r2	r2	r2		

Check whether the given string is accepted or not

dd$

state	I/p Buffer	Remark
0	dd$	S4
0d4	d$	r3 C→d
0C2	d$	S4
0C2d4	$	r3 C→d
0C2C5	$	r1 S→CC
0S1	$	accepted