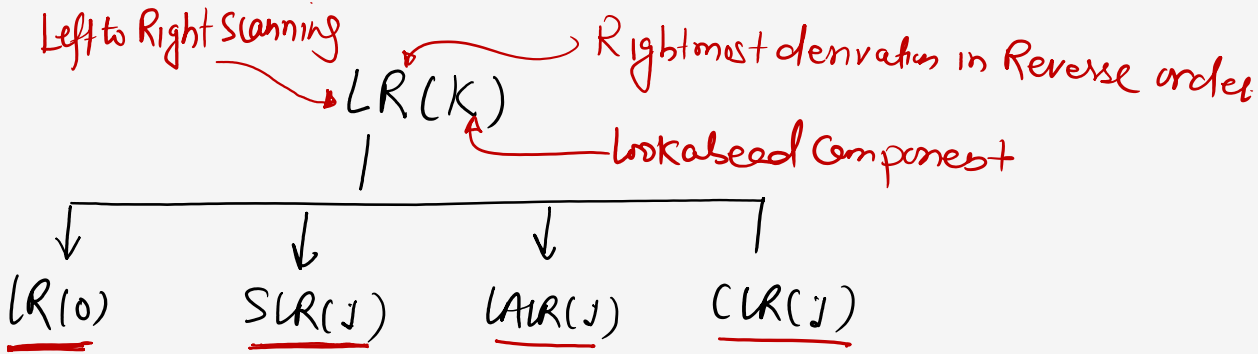
Compiler Design

Today's Class Topics

-LR(k) Parsing
-LR(0) Parsing
-SLR(1) Parsing
-S/R and R/R Conflict
-How to Test LR(0) and SLR(1) Grammar etc.

Compiler Design

LR(k) Parsing



Items:-

$A \rightarrow xyz$

$A \rightarrow xyz$ Production

$A \rightarrow \cdot xyz$
$A \rightarrow x \cdot yz$
$A \rightarrow xy \cdot z$
$A \rightarrow xyz \cdot$

} Items

xyz

Compiler Design

Augmented Grammar: -

$S' \rightarrow S$ as a new production

$\Downarrow$

$S' \rightarrow S.$ Successful parsing

Compiler Design

SLR(1) Parsing: - Dividing into two Parts

1) Construction of canonical set of items (or) LR(0) items
   - Construction of closure operation
   - Construction of goto operation
2) Construction of Parsing Table
   - Action Part
   - Goto part

Compiler Design

Construction of closure operation:-

$A \to \alpha \cdot \underline{B} \mid \beta$

i) Initially add $S' \to \cdot S$

ii) If $A \to \alpha \cdot B\beta$ is in I_i and if $B \to \gamma$ is a production then Add $B \to \cdot\gamma$ in I_i

I_i | $A \to \alpha \cdot B\beta$
 $B \to \cdot\gamma$ | $B \to \gamma$

Ex: Let $G = \{S \to AB, A \to aA \mid a, B \to bB \mid b\}$

I_0 | $S' \to \cdot S$
 $S \to \cdot AB$
 $A \to \cdot aA$
 $A \to \cdot a$

Goto$(I_0, S) \Rightarrow S' \to S\cdot$

Goto$(I_0, A) \Rightarrow (S \to A \cdot B)$ = $S \to A \cdot B$, $B \to \cdot bB$, $B \to \cdot b$

Closure (Goto(I_0, a)) = $\{A \to a \cdot A,\ A \to a\cdot,\ A \to \cdot aA,\ A \to \cdot a\}$

Construction of Goto operation:- If $A \to \alpha \cdot X\beta$ (I_i') is in I_i then Goto(I_i', X) $A \to \alpha X \cdot \beta$

Compiler Design

Ex:- Construct a canonical set of items or LR(0) items for the following Grammar

$G = \{ S \to AB, A \to aA/a, B \to b \}$ LR(0) items $\{ I_0, I_1, I_2, I_3, I_4, I_5, I_6 \}$

LR(0) items:-

Closure $(S' \to \cdot S)$

I_0

$S' \to \cdot S$
$S \to \cdot AB$
$A \to \cdot aA$
$A \to \cdot a$

I_1: Closure (Goto (I_0, S))

$S' \to S \cdot$

I_2: Closure (Goto (I_0, A))

$S \to A \cdot B$
$B \to \cdot b$

I_3: Closure (Goto (I_0, a))

$A \to a \cdot A$
$A \to a \cdot$
$A \to \cdot aA$
$A \to \cdot a$

I_4: Closure (Goto (I_2, B))

$S \to AB \cdot$

I_5: Closure (Goto (I_2, b))

$B \to b \cdot$

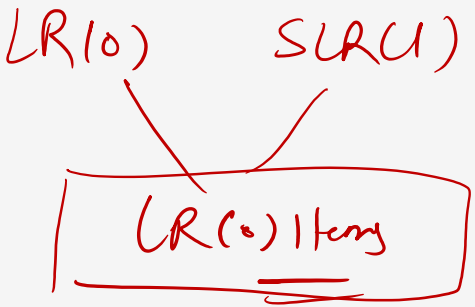
I_6: Closure (Goto (I_3, A))

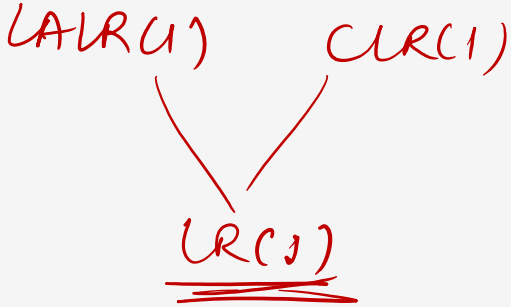
$A \to aA \cdot$

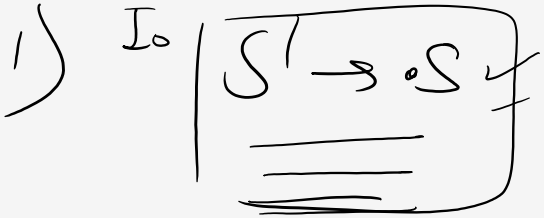
I_7: Closure (Goto (I_3, a)) $= I_3$

$A \to a \cdot A$
$A \to a \cdot$
$A \to \cdot aA$
$A \to \cdot a$

Compiler Design







Compiler Design

Question:- Construct SLR(1) Parsing table for the following grammar G:

Start → $E' \to E$

$E \to E+T \mid T$

$T \to T*F \mid F$

$F \to (E) \mid id$

Soln:-

$I_0, \ldots I_{11}$ ⇒ 12 states

LR(0) Items:- Closure $(E' \to \cdot E)$

I_0:
- $E' \to \cdot E$
- $E \to \cdot E+T$
- $E \to \cdot T$
- $T \to \cdot T*F$
- $T \to \cdot F$
- $F \to \cdot (E)$
- $F \to \cdot id$

I_1: Closure (Goto (I_0, E))
- $E' \to E \cdot$
- $E \to E \cdot +T$

I_2: Closure (Goto (I_0, T))
- $E \to T \cdot$
- $T \to T \cdot *F$

I_3: $T \to F \cdot$

Goto $(I_0, ($) = I_4

I_4:
- $F \to (\cdot E)$
- $E \to \cdot E+T$
- $E \to \cdot T$
- $T \to \cdot T*F$
- $T \to \cdot F$
- $F \to \cdot (E)$
- $F \to \cdot id$

I_5: $F \to id \cdot$

I_6: Closure (Goto $(I_1, +)$)
- $E \to E+ \cdot T$
- $T \to \cdot T*F$
- $T \to \cdot F$
- $F \to \cdot (E)$
- $F \to \cdot id$

I_7: Closure (Goto $(I_2, *)$)
- $T \to T* \cdot F$
- $F \to \cdot (E)$
- $F \to \cdot id$

I_8: Closure (Goto (I_4, E))
- $F \to (E \cdot)$
- $E \to E \cdot +T$

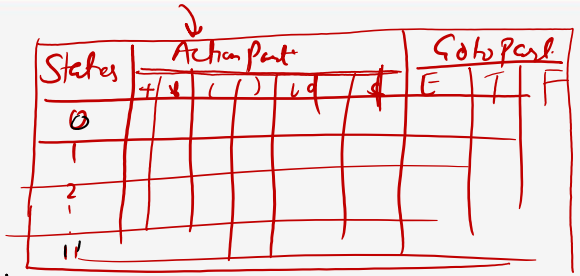
I_9: Closure (Goto (I_6, T))
- $E \to E+T \cdot$
- $T \to T \cdot *F$

I_{10}: Closure (Goto (I_7, F))
- $T \to T*F \cdot$

I_{11}: Closure (Goto $(I_8,)$)
- $F \to (E) \cdot$

	(	)	+	*	id	$	E	T	F
I_0	S4						1	2	3
I_1			S6						
I_2				S7					
I_3									
I_4	S4						8	2	3
I_5									
I_6	S4							9	3
I_7	S4								10
I_8		S11	S6						
I_9				S7					
I_{10}									
I_{11}									

Compiler Design



Construction of SLR(1) Parsing Table :- — Action Part, Goto Part.

The Action Part of the table will be filled as follows

1) If $A \to \alpha \cdot a \beta$ is in I_i and if Goto $(I_i, a) = I_j$, Then Set Action $[i, a] = S_j$ (Shift j)

2) If $A \to \alpha \cdot$ is in I_i Then Set Action $[i, x]$ = Reduce by $A \to \alpha$ i.e R_1 $= R_1$

$A \to \alpha \cdot$ must not be an Augmented Production where $x \in$ follow(A)

3) If $S' \to S \cdot$, is in I_i Then Set Action $[i, \$]$ = Accept

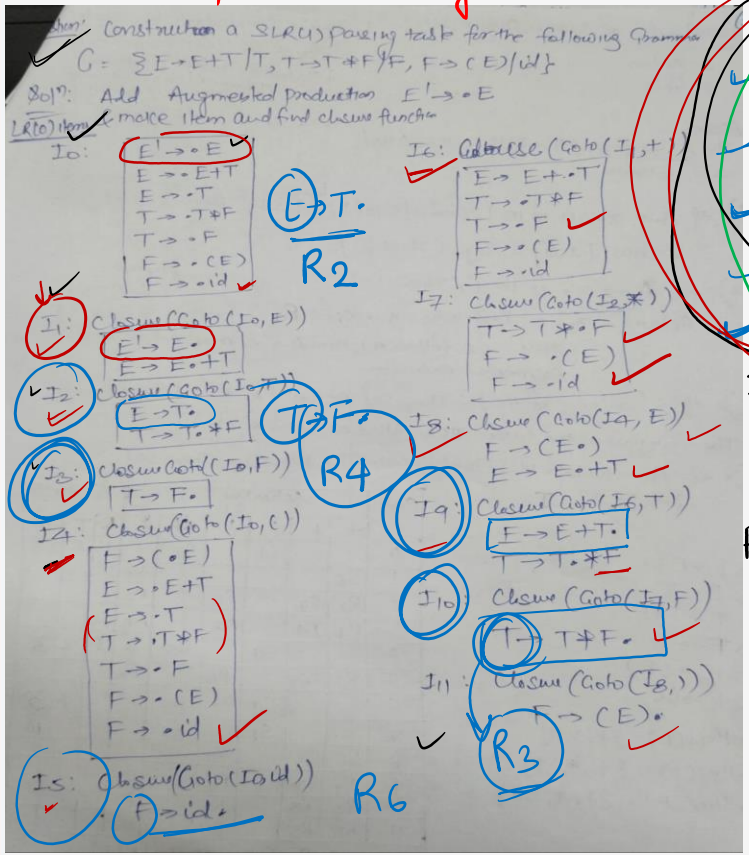
The Goto Part of the table will be filled as follows:

1) If $A \to \alpha \cdot B \beta$ is I_i and if Goto $(I_i, B) = I_j$ Then Set Action $[i, B] = j$

Let suppose: 1) $A \to \alpha$ 2) $B \to \gamma$

Compiler Design

Construction of SLR(1) Parsing Table :-



1) E→E+T
2) E→T
3) T→T*F
4) T→F
5) F→(E)
6) F→id

SLR(1)

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

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

Follow(F) = {+, *,), $}

E→E+T. R1

F→(E). R5

SLR(1) Parsing Table

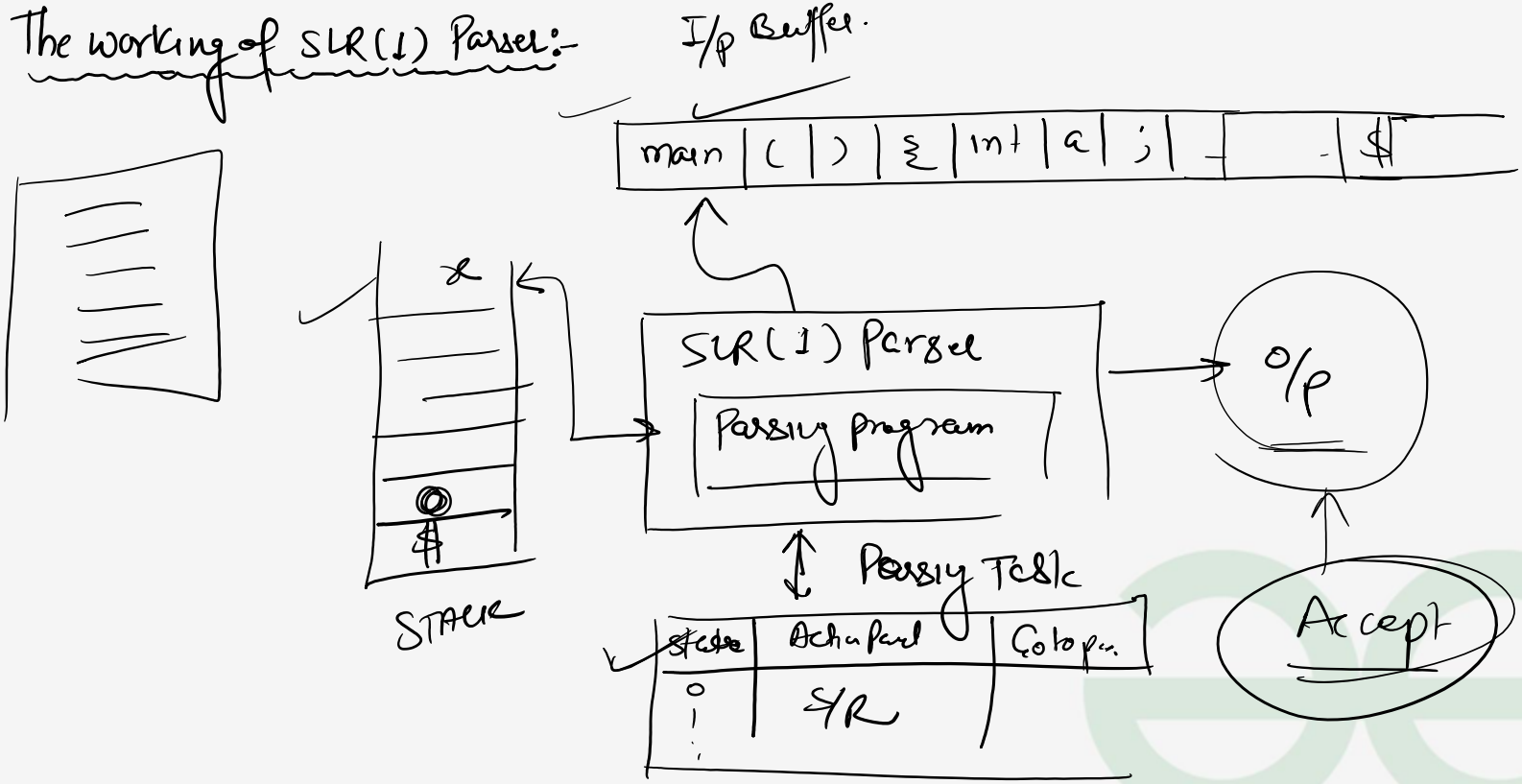
States	Action Part						Goto Part		
	+	*	(	)	id	$	E	T	F
0	-	-	S4	-	S5	-	1	2	3
1	S6		-	-	-	Accept	-	-	-
2	R2	S7		R2	-	R2	-	-	
3	R4	R4		R4	-	R4	-	-	-
4	-	-	S4	-	S5	-	8	2	3
5	R6	R6	-	R6	-	R6	-	-	-
6	-	-	S4	-	S5		-	9	3
7	-	-	S4		S5	-	-		10
8	S6			S11					
9	R1	S7		R1		R1			
10	R3	R3		R3		R3			
11	R5	R5		R5		R5			

SLR(1) Grammar

Compiler Design

Compiler Design

The working of SLR(1) Parser:-



Compiler Design

How input string id*id+id Parse

id+id*id

STACK	Input	Action
$0	id *id +id $	Shift
$0id5	*id +id $	Reduce F→id
$0F3	*id +id $	Reduce T→F
$0T2*7	id+id $	Shift, Shift
$0T2*7id5	+id $	Reduce F→id
$0T2*7F10	+id $	Reduce T→T*F
$0T2	+id $	Reduce E→T
$0E1	+id $	Shift
$0E1+6	id $	Shift
$0E1+6id5	$	Reduce F→id
$0E1+6F3	$	Reduce T→F
$0E1+6T9	$	Reduce E→E+T
$0E1	$	Accepted

SLR(1) Parsing Table

id*id+id+id

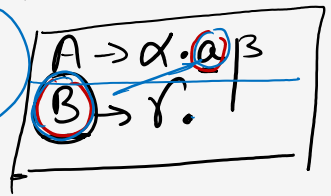
States	Action Part						Goto Part		
	+	*	(	)	id	$	E	T	F
0			S4		S5		1	2	3
1	S6	R2				Acc			
2	R2	S7		R2		R2			
3	R4	R4		R4		R4			
4			S4		S5		8	2	3
5	R6	R6		R6		R6			
6			S4		S5			9	3
7			S4		S5				10
8	S6			S11					
9	R1	S7		R1		R1			
10	R3	R3		R3		R3			
11	R5	R5		R5		R5			

Compiler Design

Conflict in SLR(1) Parsing:-

- Shift-Reduce Conflict (S/R-Conflict)
- Reduce-Reduce Conflict (R/R Conflict)

S/R Conflict:



$Goto(I_i, @) = I_j$

k) $B \to \gamma .$

R_k

follow(B): {a, $}

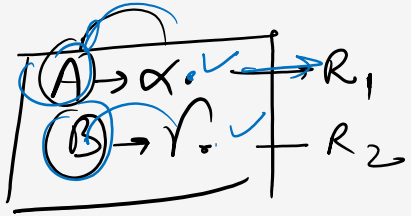
	a	b	$
I_i	S_j / R_k	R_k	

$\{a\} \cap \{follow(B)\} \neq \phi \Rightarrow$ S/R Conflict $\Rightarrow$ not SLR(1) Grammar.

R/R Conflict:

Let 1) $A \to \alpha .$

2) $B \to \gamma .$



	a	b	$
I_i	R_2		R_1 / R_2

follow(A) $\cap$ follow(B) $\neq \phi \Rightarrow$ R/R Conflict $\Rightarrow$ not SLR(1)

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

Compiler Design

Ex:- Check whether the following Grammar is SLR(1) or not ?

$G = \{ S \to A \mid a, A \to a \}$ ⇒ SLR(1)

1) $S \to A$
2) $S \to a$
3) $A \to a$

Unambiguous

Ambiguous

I0:

$S' \to \cdot S$
$S \to \cdot A$
$S \to \cdot a$
$A \to \cdot a$

S → I1
A → I2
a → I3

I3:

$S \to a \cdot$ → R2 (follow(S))
$A \to a \cdot$ → R3 (follow(A))

⇒

follow(S) ∩ follow(A) ≠ ϕ ⇒ not SLR(1) ⇒ R/R Conflict

{$} ∩ {$}

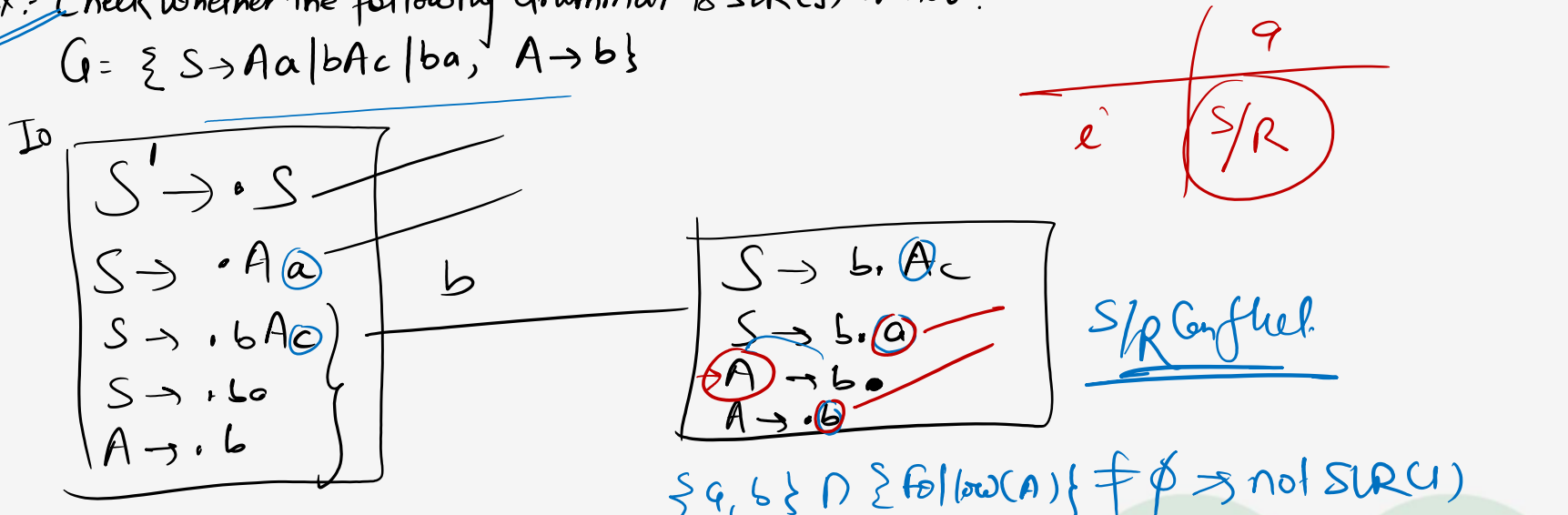
= $

⇒ $

Compiler Design

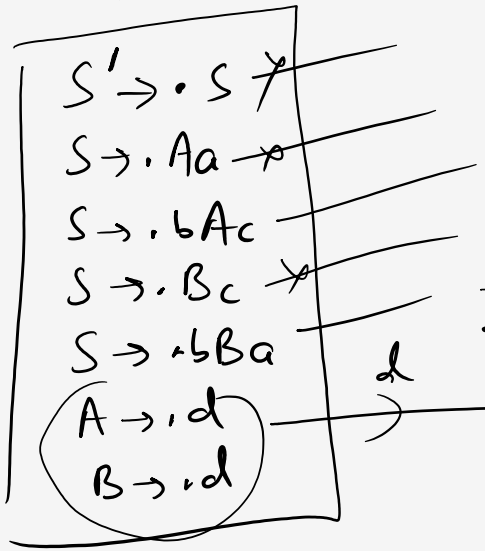
Ex:- Check whether the following Grammar is SLR(1) or not?

$G = \{ S \to Aa \mid bAc \mid ba,\ A \to b \}$

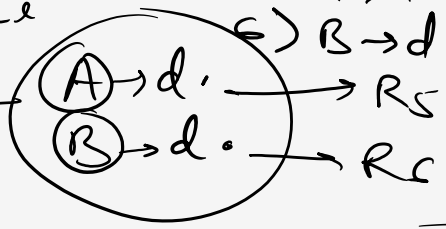


Compiler Design

Ex:- $G = \{ S \to A@ \mid bA© \mid Bc \mid bBa,\ A \to d,\ B \to d \}$



d



1) $S \to Aa$
2) $S \to bAc$: Follow(A) ∩ Follow(B)
3) $S \to Bc \Rightarrow \{a, c\} \cap \{a\} = a \neq \phi$
4) $S \to bBa$
5) $A \to d$
6) $B \to d$

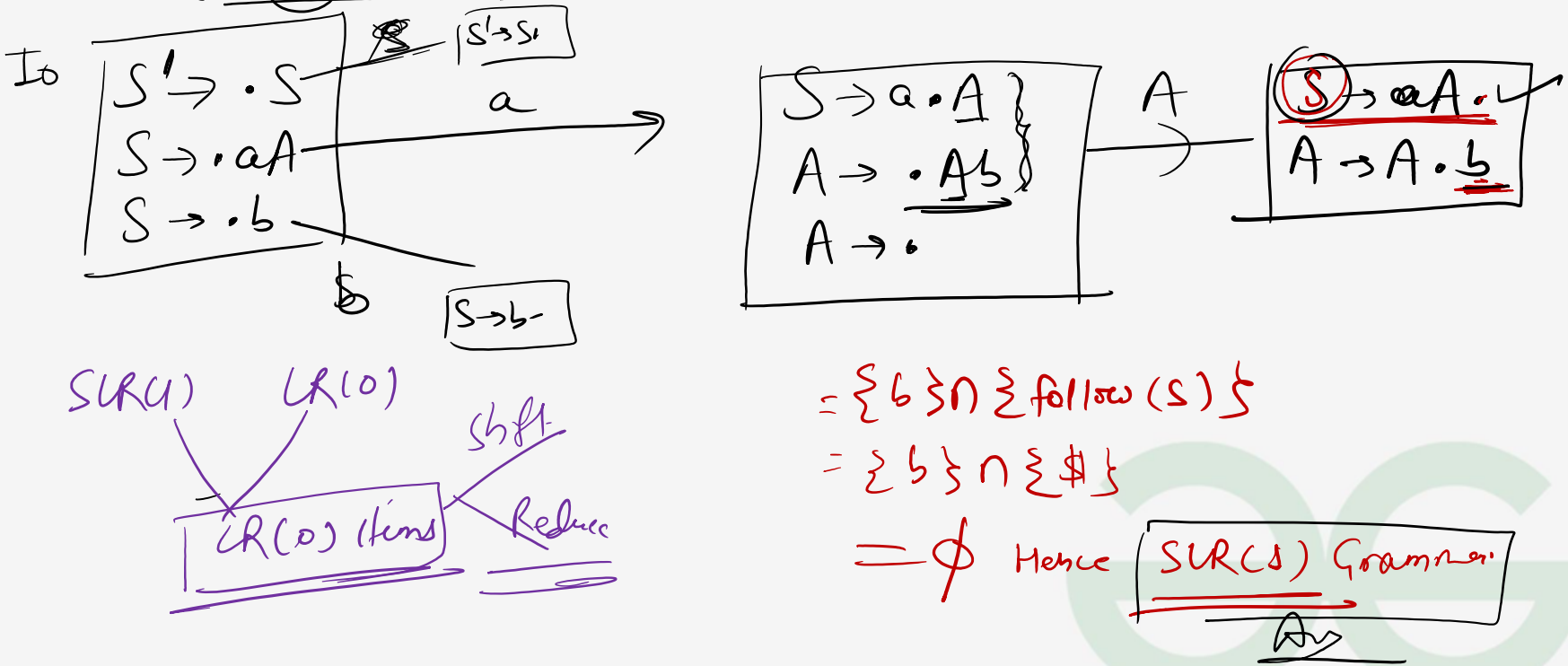
R/R

R5/R6

not SLR(1)

Compiler Design

Ex:- $G = \{ S \to aA \mid b,\ A \to Ab \mid \epsilon \}$



SLR(1) LR(0)

LR(0) items — Shift / Reduce

$= \{b\} \cap \{follow(S)\}$

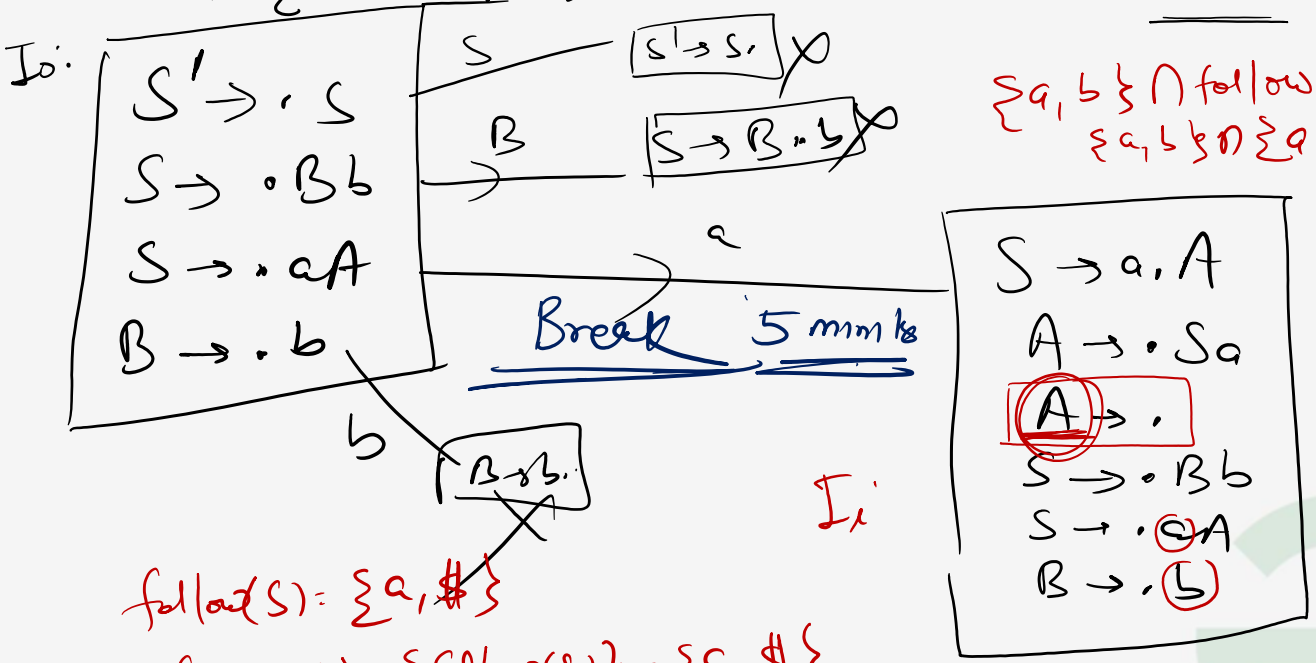
$= \{b\} \cap \{\$\}$

$= \phi$ Hence SLR(1) Grammar

Compiler Design

Ex:- $G = \{ S \to Bb \mid aA,\ A \to Sa \mid \epsilon,\ B \to b \}$

SLR(1) ⇒ UnAmbiguous



Break 5 mins

follow(S) = {a, $}

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

{a, b} ∩ follow(A) = {a, b} ∩ {a, $} = {a} ≠ ϕ

S/R Conflict

⇓

not SLR(1)

Compiler Design

LR(0) Parsing :- It can be divided into two Parts

1) Construction of Canonical set of items or LR(0) items :-
It is same as Construction of LR(0) items in SLR(1)

2) Construction of Parsing Table :- It is also same as Construction of Parsing Table in SLR(1) except that if $A \to \alpha.$ is in I_i then we write Reduce by $A \to \alpha.$ in entire Action Part of i^{th} row of the Table

i.e I_i $A \to \alpha.$ $[i, x]$ = Reduce by $A \to \alpha$ = R_1

	a	b	$
i	R_1	R_1	R_1

Let 1) $A \to \alpha$

Ex:- G : $E \to E+T \mid T$, $T \to T*F \mid F$, $F \to (E) \mid id$

I_2 = LR(0) (I_6, I_{11}) $\Rightarrow$ I_2 : $E \to T.$, $T \to T.*F$

LR(0) Items

2) $E \to T$ (R_2)

LR(0) Parsing Table

States	Action Part						Goto Part		
	+	*	(	)	id	$	E	T	F
0			S_4		S_5		1	2	3
1	S_6					Acc			
2	R_2	S_7/R_2	R_2	R_2	R_2	R_2			
3	R_4	R_4	R_4	R_4	R_4	R_4			
4			S_4		S_5		8	2	3
5	R_6	R_6	R_6	R_6	R_6	R_6			
6			S_4		S_5			9	3
7			S_4		S_5				10
8	S_6			S_{11}					
9	R_1	S_7/R_1	R_1	R_1	R_1	R_1			
10	R_3	R_3	R_3	R_3	R_3	R_3			
11	R_5	R_5	R_5	R_5	R_5	R_5			

Compiler Design

Conflicts in LR(0) :-

1- S/R Conflict

2- R/R Conflict

S/R Conflict

I_i : $A \to \alpha . a \beta$, $B \to \gamma .$ → S/R

R/R Conflict

I_i : $A \to \alpha . \to R_1$, $B \to \gamma . \to R_2$; follow(A) ∩ follow(B) ≠ ∅

Compiler Design

Question: The following grammar is $G = \{ S \to A(a) \mid bA(c) \mid d,\ A \to d \}$

a) LR(0) but not SLR(1)
b) ~~SLR(1) but not LR(0)~~
c) Both LR(0) & SLR(1)
d) Neither LR(0) nor SLR(1)

$S' \to .S$
$S \to .Aa$
$S \to .bAc$
$S \to .d$
$A \to .d$

I_i

$S \to d.$ — R_3
$A \to d.$ — R_4

SLR(1) but not LR(0)

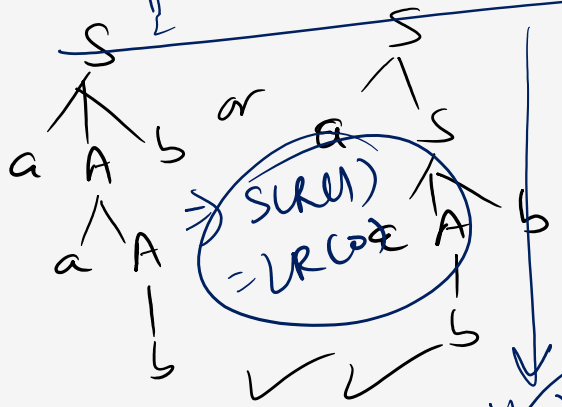
not LR(0) Parser

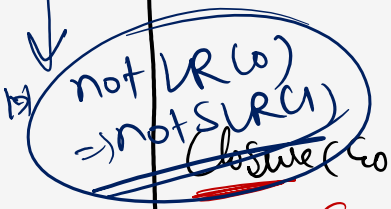
$follow(S) \cap follow(A) = \phi$

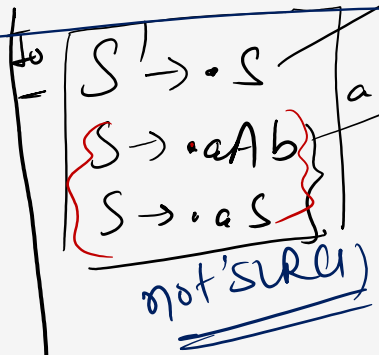
$\{\$\} \cap \{a, c\} = \phi$

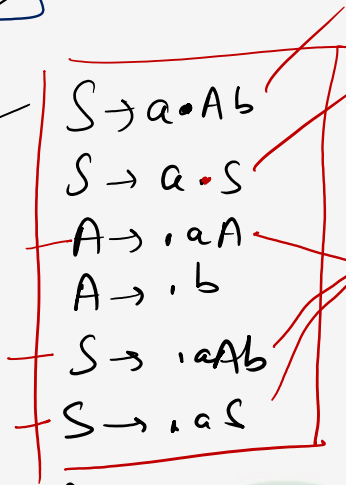
Compiler Design

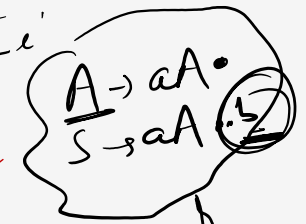
Question:- $G = \{ S \to aAb \mid aS,\ A \to aA \mid b \}$

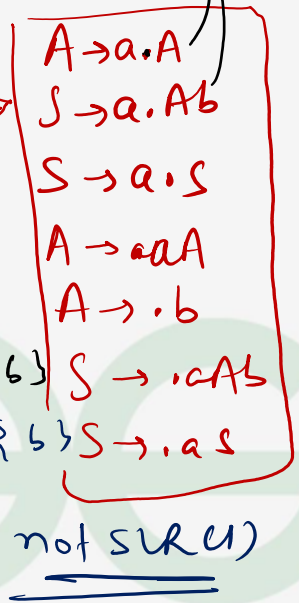












follow(A) ∩ {b}
{b, $} ∩ {b} ⇒ S → ·aS

not SLR(1)

Compiler Design

Compiler Design

Consider the following grammar.

```
S -> S * E
S -> E
E -> F + E
E -> F
F -> id
```

$S \rightarrow S*E \mid E$

$E \rightarrow F+E \mid F$

$F \rightarrow id$

2003

Consider the following LR(0) items corresponding to the grammar above.

```
(i) S -> S * .E
(ii) E -> F. + E
(iii) E -> F + .E
```

1) $S \rightarrow S*\cdot E$
2) $E \rightarrow F\cdot +E$
3) $E \rightarrow F+\cdot E$

Given the items above, which two of them will appear in the same set in the canonical sets-of-items for the grammar?

- [ ] (i) and (ii)
- [ ] (ii) and (iii)
- [ ] (i) and (iii)
- [ ] None of the above

Compiler Design

A canonical set of items is given below

```
S --> L.> R
Q --> R.
```

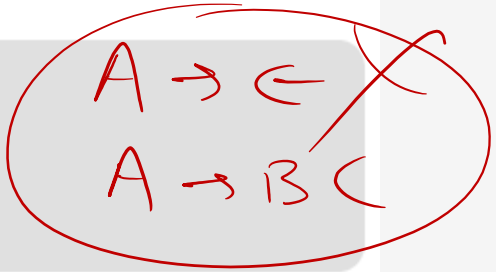
On input symbol < the set has

- [ ] a shift-reduce conflict and a reduce-reduce conflict.
- [ ] a shift-reduce conflict but not a reduce-reduce conflict.
- [ ] a reduce-reduce conflict but not a shift-reduce conflict.
- [ ] neither a shift-reduce nor a reduce-reduce conflict.

Compiler Design

Which of the following grammar rules violate the requirements of an operator grammar ? P, Q, R are nonterminals, and r, s, t are terminals.

1. P → Q R
2. P → Q s R
3. P → ϵ
4. P → Q t R r

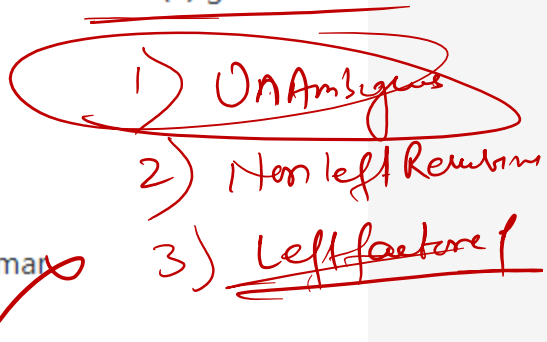


- [ ] 1 only
- [x] 1 and 3 only
- [ ] 2 and 3 only
- [ ] 3 and 4 only

Compiler Design

Which of the following suffices to convert an arbitrary CFG to an LL(1) grammar?

- [ ] Removing left recursion alone
- [ ] Factoring the grammar alone
- [ ] Removing left recursion and factoring the grammar
- [ ] None of these



Compiler Design

Thank You !