Compiler Design

Today's Class Topics

-CLR(1) Parsing
-LALR(1) Parsing
-S/R and R/R Conflict
-How to Test CLR(1) and LALR(1) Grammar etc.
-Relationship amongst LL(1), LR(0), SLR(1), CLR(1) and LALR(1)
-YACC Tool

Compiler Design

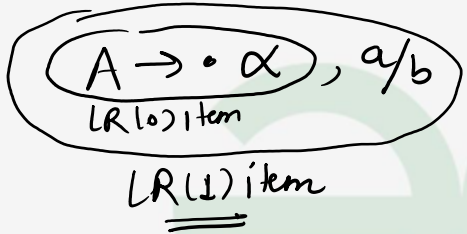
CLR(1) or LR(1) Parsing :- It can be divided into two Parts

1- Construction of Canonical set of items or LR(1) items
   - Closure operation
   - goto operation

2 - Construction of Parsing Table
   - Action Part
   - Goto part

LR(0) items
- LR(0) Parsing
- SLR(1) Parsing

LR(1) items = LR(0) items + lookahead Symbols
- CLR(1) parsing
- LALR(1) parsing



Compiler Design

Construction of closure operation:

1- Initially we add $S' \to \cdot S, \$$

2- if $A \to \alpha \cdot B\beta, a$ is in I_i and if $B \to \gamma$ is a production then Add $B \to \cdot\gamma, x$ in I_i where $x \in First(\beta a)$

I_i: $A \to \alpha \cdot B\beta, a$, $B \to \gamma$
$B \to \cdot\gamma, x$

$x = First(\beta a)$

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

I_0:
$S' \to \cdot S, \$$
$S \to \cdot AB, \$$
$A \to \cdot aA, b$
$A \to \cdot a, b$

$First(\in \$) = First(\$) = \$$

$First(B\$) = First(B) = \{b\}$

$First\in$

Construction of Goto operation:-

1) if $A \to \alpha \cdot X\beta, a$ is in I_i then $Goto(I_i, X) = A \to \alpha X \cdot \beta, a$

Here X can be either a terminal or non terminal

$Goto(I_i, X)$

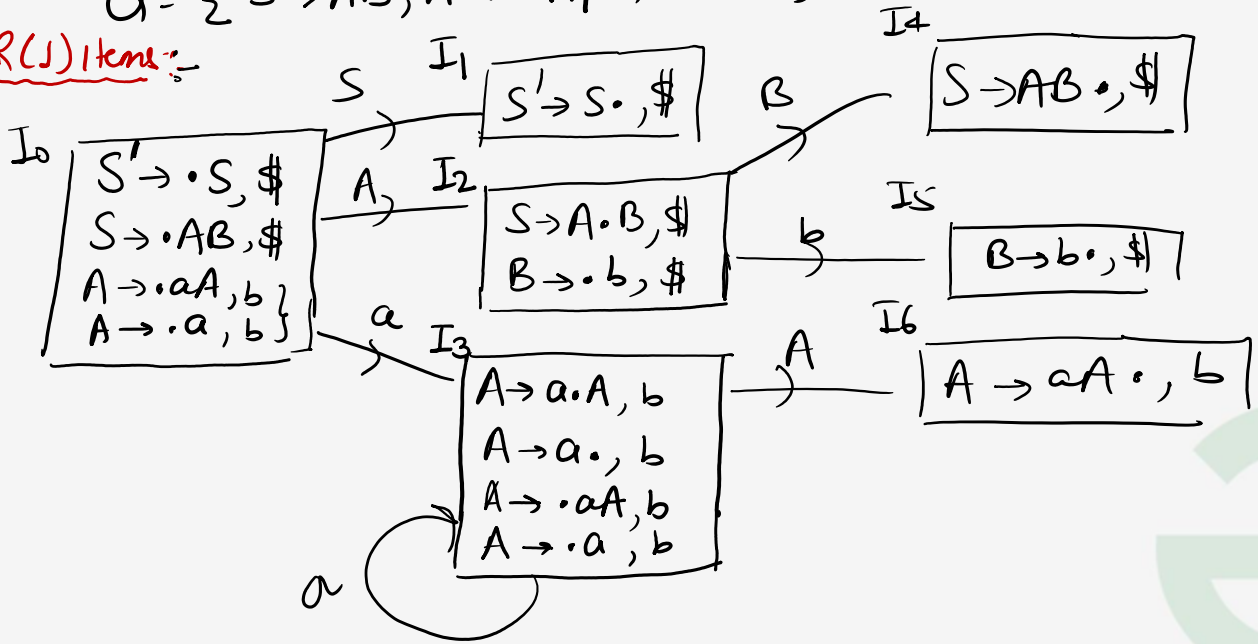
I_i: $A \to \alpha \cdot X\beta, a$ → I_j: $A \to \alpha X \cdot \beta, a$

Compiler Design

Ex:- Construct LR(1) items for the following Grammar

$G = \{ S \to AB, A \to aA/a, B \to b \}$

LR(1) items :-

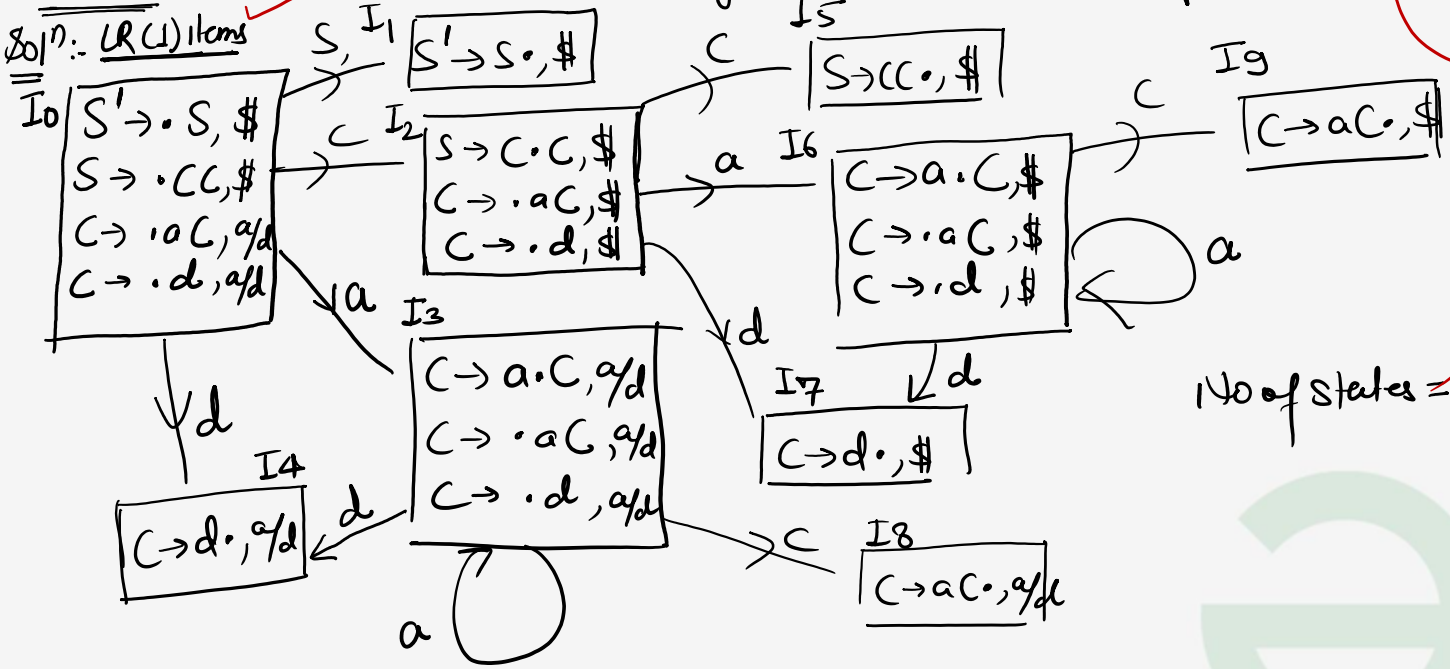


LR(1) items ⇒ States = ?

Compiler Design

Question:- Construct CLR(1) Parsing Table for the following Grammar $G: \{ S \to CC,\ C \to aC/d \}$ ⇒

Soln:- LR(1) items



1) S → CC
2) C → aC
3) C → d

No of states = 10

Compiler Design

CLR(1) Parsing Table Construction: It has 2-parts → Action Part, Goto part

The Action Part of the Table will be filled as follows:

i) if $A \to \alpha \cdot a\beta$, b is in I_i and if Goto$(I_i, a) = I_J$ Then Set Action $[i, a] = S_J$ (shift-J)

ii) if $A \to \alpha \cdot$, a is in I_i then set Action $[i, a]$ = Reduce by $A \to \alpha$, Here $A \to \alpha \cdot$ must not be an Augmented Production

iii) if $S' \to S \cdot$, \$ is in I_i then set Action $[i, \$]$ = Accept

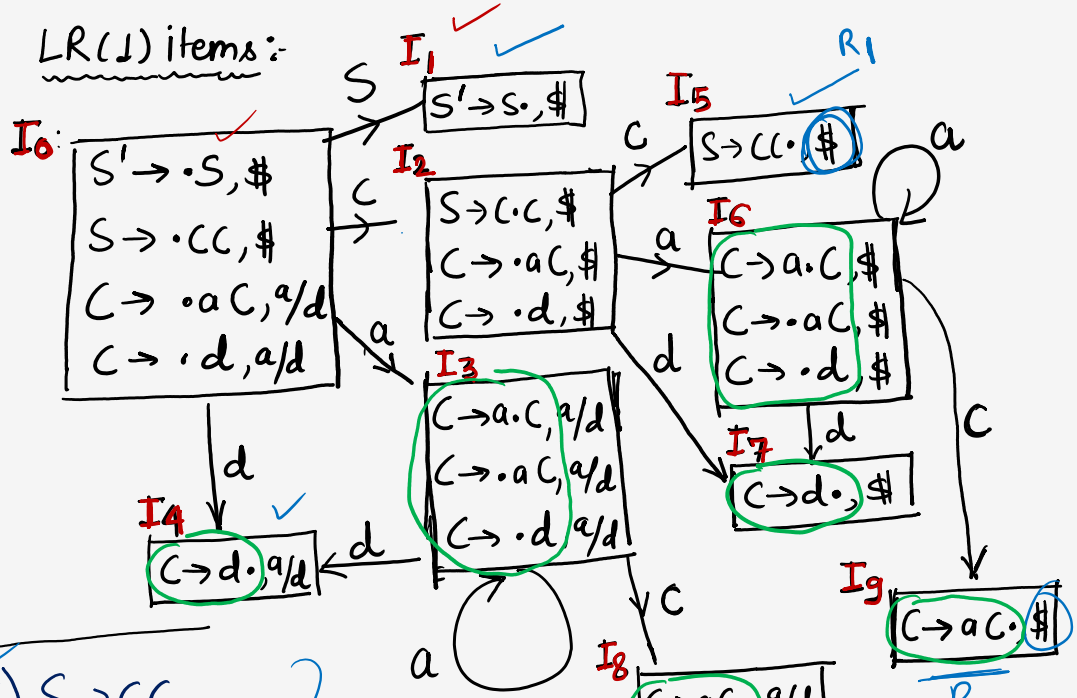
The Goto Part of the Table will be filled as follows:

i) if $A \to \alpha \cdot B\beta, a$ is in I_i and if Goto$(I_i, B) = I_J$ Then

I_i Set goto $[i, B] = J$ $\quad [i, B] = J$

Compiler Design

LR(1) items :-



1) S → CC
2) C → aC
3) C → d

R3

CLR(1) Parsing Table

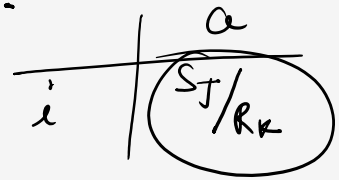
States	Action Part			Goto part	
	a	d	$	S	C
0	S3	S4		1	2
1			Accept		
2	S6	S7			5
3	S3	S4			8
4	R3	R3			
5			R1		
6	S6	S7			9
7			R3		
8	R2	R2			
9			R2		

CLR(1) Grammar.

Compiler Design

Conflict in CLR(1) :-

- S/R Conflict
- R/R Conflict



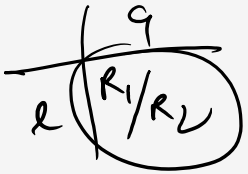
S/R Conflict:

I_i | $A \to \alpha . a\beta, b$
$B \to \gamma . , a$

S/R Conflict $\Rightarrow$ not CLR(1)

R/R Conflict:

I_i | $A \to \alpha . , a$
$B \to \gamma . , a$



R/R Conflict $\Rightarrow$ not CLR(1)

Compiler Design

Question:- Check whether the following grammar is CLR(1) or not?

$G = \{ S \to A \mid a, A \to a \}$

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

Soln:

I_0

$S' \to .S, \$$

$S \to .A, \$$

$S \to .a, \$$

$A \to .a, \$$

$\xrightarrow{a}$ I_i

$S \to a., \$$

$A \to a., \$$

	$
a	R_2/R_3

R/R Conflict not CLR(1)

Compiler Design

NOTE :-

1) An Ambiguous Grammar can never be LR(K) for any K

2) Every LR(K) Grammar is UnAmbiguous but every UnAmbiguous Grammar is not LR(K)

3) Every LR(1) Grammar has an equivalent DPDA → UnAmbiguous

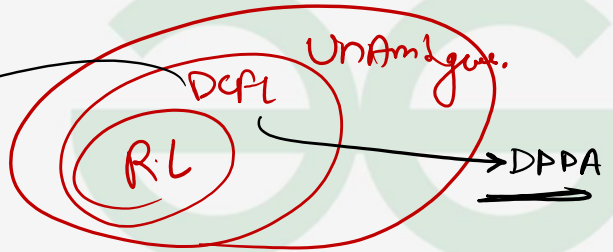
4) Every Regular Language has a LR(1) grammar

5) Every Regular Grammar can not be LR(1) grammar

R.G < Ambiguous ✓, UnAmbiguous

R.L → UnAmbiguous

DCFL → UnAmbiguous

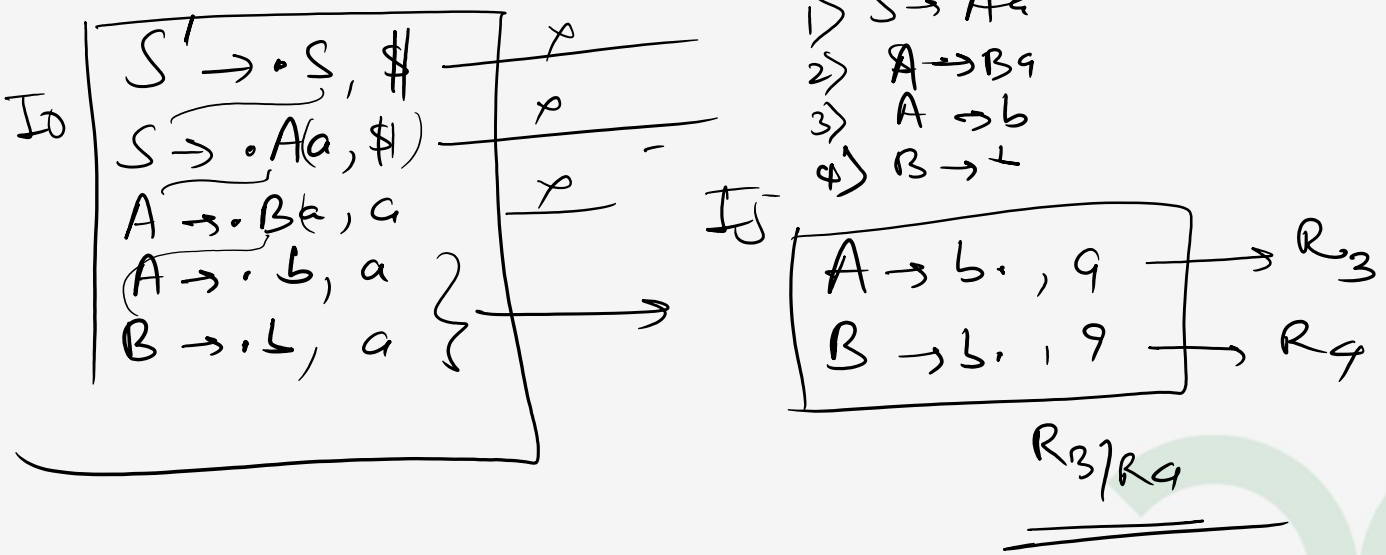


DPDA

Compiler Design

Question: $G = \{ S \to Aa, A \to Ba/b, B \to b \}$ CLR(1) = ?

Soln:



R_3/R_4

Compiler Design

Question: $G = \{ S \rightarrow aSa \mid Aa, A \rightarrow aA \mid b \}$

Compiler Design

LR(1) Items :-

LALR(1) Parsing :- minimized form of CLR(1)

Ex: $G = \{S \to CC, C \to aC/d\}$

I_3: $C \to a.C, a/d$; $C \to .aC, a/d$; $C \to .d, a/d$

I_6: $C \to a.C, \$$; $C \to .aC, \$$; $C \to .d, \$$

$I_3 + I_6 \Rightarrow I_{36}$: $C \to a.C, a/d/\$$; $C \to .aC, a/d/\$$; $C \to .d, a/d/\$$

I_4: $C \to d., a/d$

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

$I_4 + I_7 \Rightarrow I_{47}$: $C \to d., a/d/\$$

I_8: $C \to aC., a/d$

I_9: $C \to aC., \$$

$I_8 + I_9 \Rightarrow I_{89}$: $C \to aC., a/d/\$$

LALR(1) Parsing Table

States	Action Part			Goto Part	
	a	d	$	S	C
0	S_3	S_4		1	2
1			Accept		
2	S_6	S_7			5
36	S_{36}	S_{47}			89
47	R_3	R_3	R_3		
5			R_1		
89	R_2	R_2	R_2		

Compiler Design

Conflicts in LALR(1) :-

- S/R Conflict
- R/R Conflict

} ⇒ similar to CLR(1)

Compiler Design

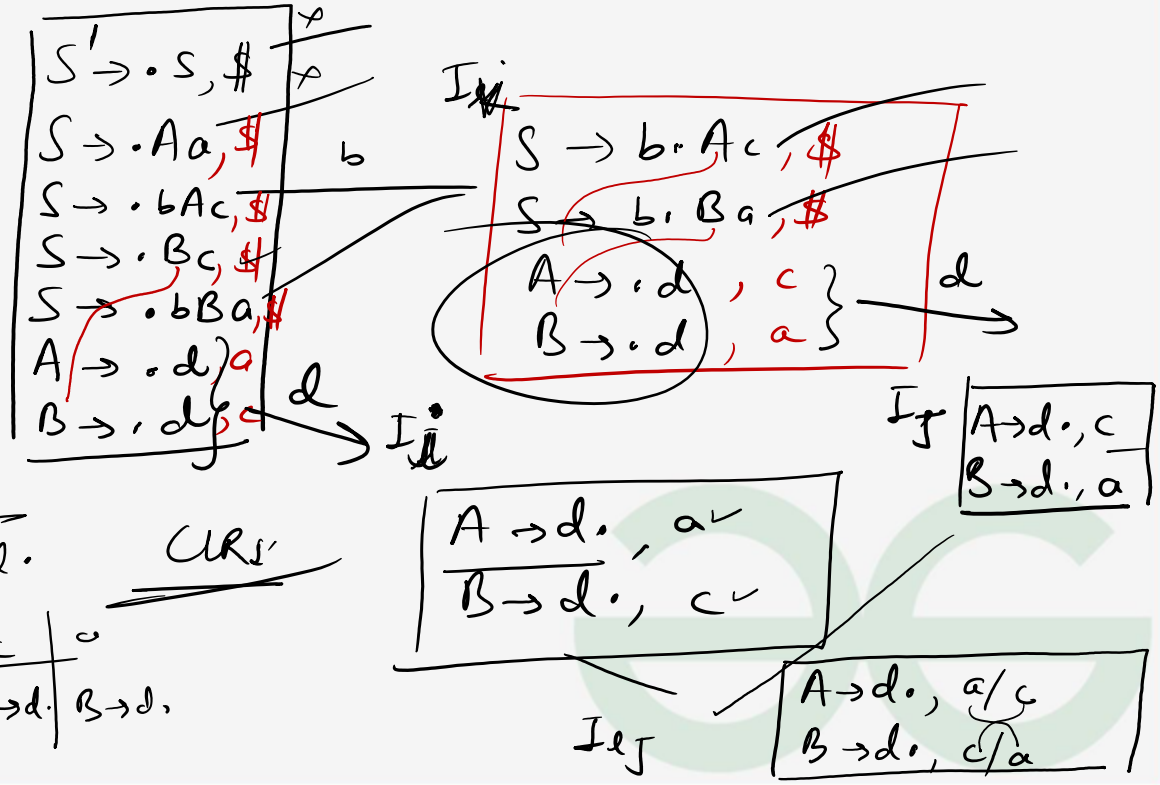
Question: The following Grammar is $G = \{S \to Aa \mid bAc \mid Bc \mid bBa,\ A \to d,\ B \to d\}$

a) LALR(1) but not CLR(1)

b) LR(1) but not LALR(1)

c) Both LALR(1) & LR(1)

d) Neither LALR(1) nor LR(1)



Compiler Design

NOTE:

	CLR(1)	LALR(1)
S/R Conflict	No	No
	Yes	Yes
R/R Conflict	Yes	Yes
	No	Yes / No

$x \to y \equiv \neg y \to \neg x$

$x \to y \not\equiv y \to x$

LALR(1) $\Rightarrow$ CLR(1)

CLR(1) $\Rightarrow$ LALR(1)

Compiler Design

Ex:- I_i + I_j = I_{ij}

I_i: $A \to \alpha . a\beta, b$; $B \to \gamma . , c$ — No S/R

I_j: $A \to \alpha . a\beta, a$; $B \to \gamma . , d$ — No S/R

I_{ij}: $A \to \alpha . a\beta, b/a$; $B \to \gamma . , c/d$ — No S/R Conflict

Ex: $A \to \alpha . a\beta, b$; $B \to \gamma . , c$ — No S/R

+ $A \to \alpha . a\beta, a$; $B \to \gamma . , a$ — S/R Conflict

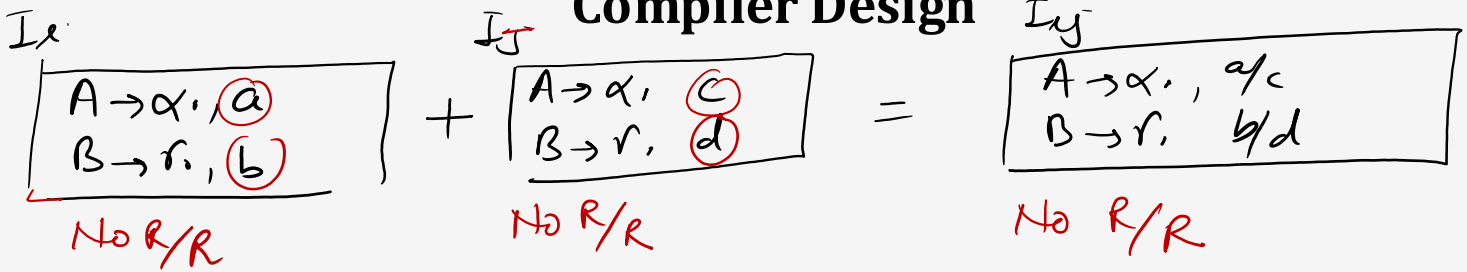
= $A \to \alpha . a\beta, b/a$; $B \to \gamma . , c/a$ — S/R Conflict

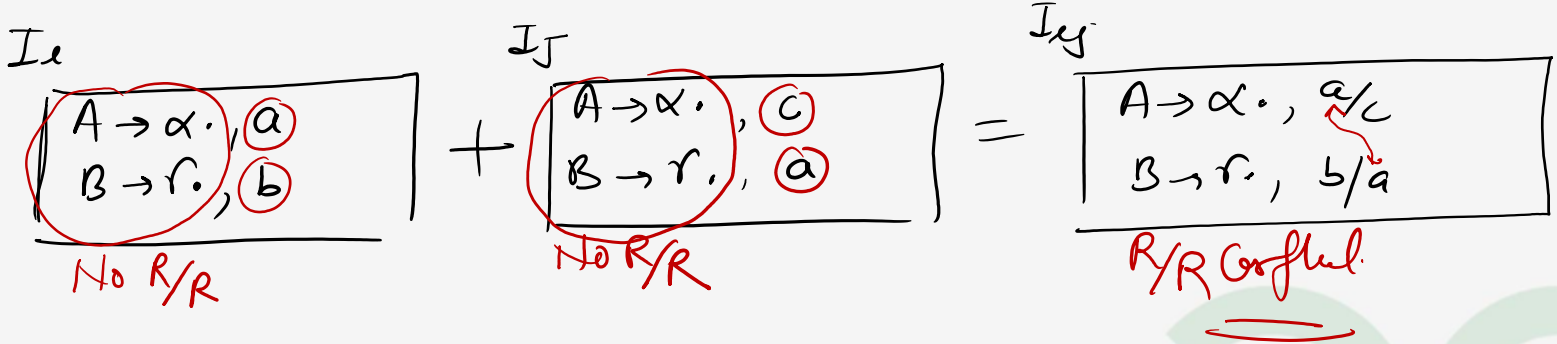
Ex: I_i $A \to \alpha . , a$; $B \to \gamma . , b$ — No R/R

+ I_j $A \to \alpha . , c$; $B \to \gamma . , c$ — R/R Conflict

= I_{ij} $A \to \alpha . , a/c$; $B \to \gamma . , b/c$ — R/R Conflict

Compiler Design

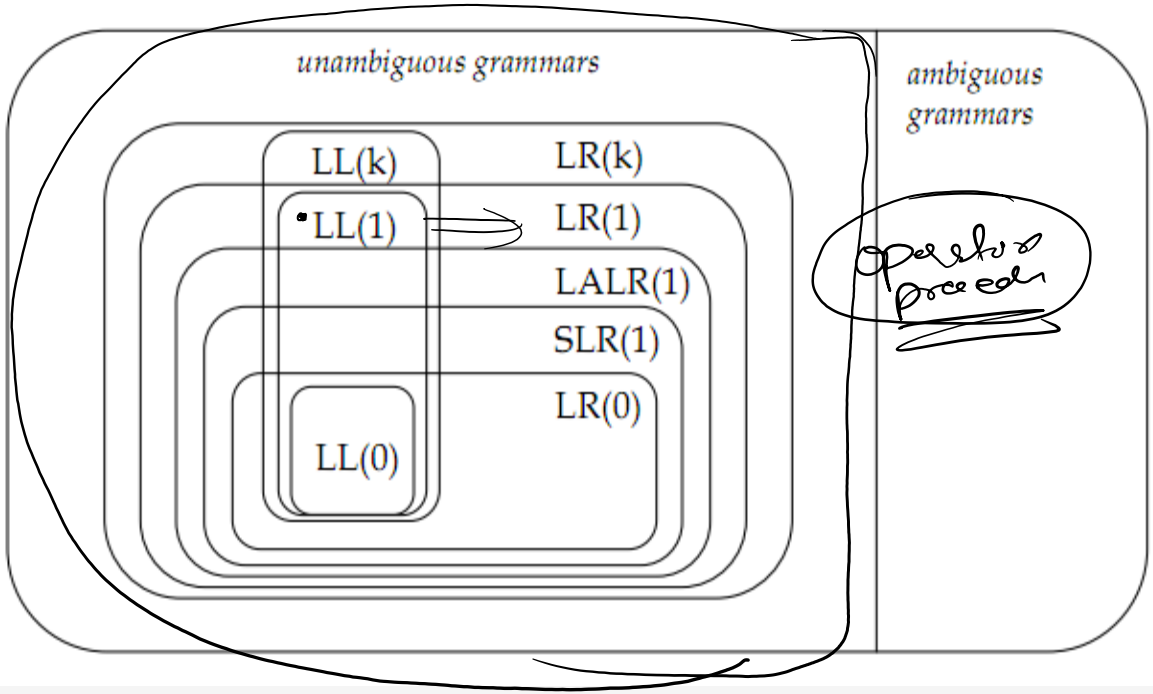




Compiler Design

LL(1) versus LR(k)

A picture is worth a thousand words:



$LL(K) \Rightarrow LR(K)$

$LL(0) \Rightarrow LR(0)$

$LL(1) \Rightarrow LR(1)$

$LL(2) \not\Rightarrow LR(2)$

~~$LL(1) \Rightarrow LR(0)$~~

$LL(0) \Rightarrow LR(0) = LR(0) \Rightarrow SLR(1) \Rightarrow LALR(1) \Rightarrow CLR(1)$

Compiler Design

LL(1) Vs LR(1) :-

LR(1) → LL(1)

$A \to \alpha_1 | \alpha_2 | \alpha_3 | \epsilon$

Case 1: LL(1) With No Conflicts

LL(1) → LR(1)

LR(1) → LL(1)

Imp

Case 2: ϵ-free LL(1) → LR(0) → SLR(1) → LALR(1) → CLR(1)

Question: G: $S \to CC$

$C \to cC | d$

LL(1) ⇒ CLR(1)

LL(1):

first(cC) ∩ first(d)

= c ∩ d = ϕ

a) LL(1) ✓

b) LL(1) but not LR(0) ✗

c) LR(0) but not SLR(1) ✗

d) SLR(1) but not LALR(1) ✗

Break

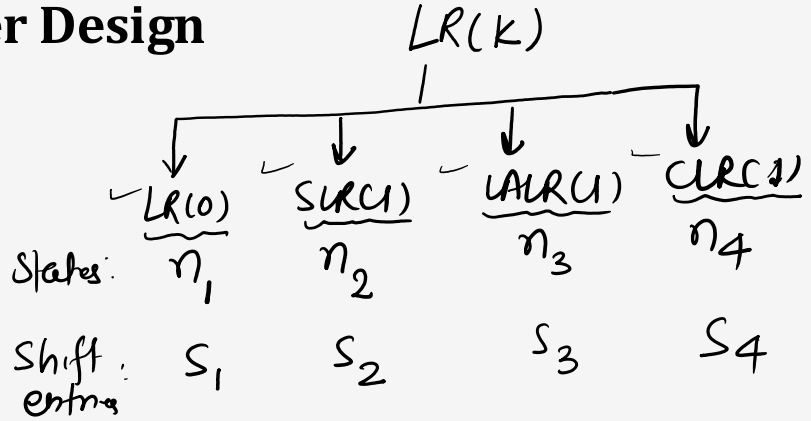
5 mins

Compiler Design

NOTE :-

1) $n_1 = n_2 = n_3 \leq n_4$

2) $|S_1| = |S_2| = |S_3| \leq |S_4|$



3) Shift entry rule is same for all LR(K) Tables

4) Reduce entry rule is different for all LR(K)

5) No. of Reduce entries $LR(0) \geq SLR(1) \geq LALR(1) \geq CLR(1)$

6) Power of $LR(0) < SLR(1) < LALR(1) < CLR(1)$

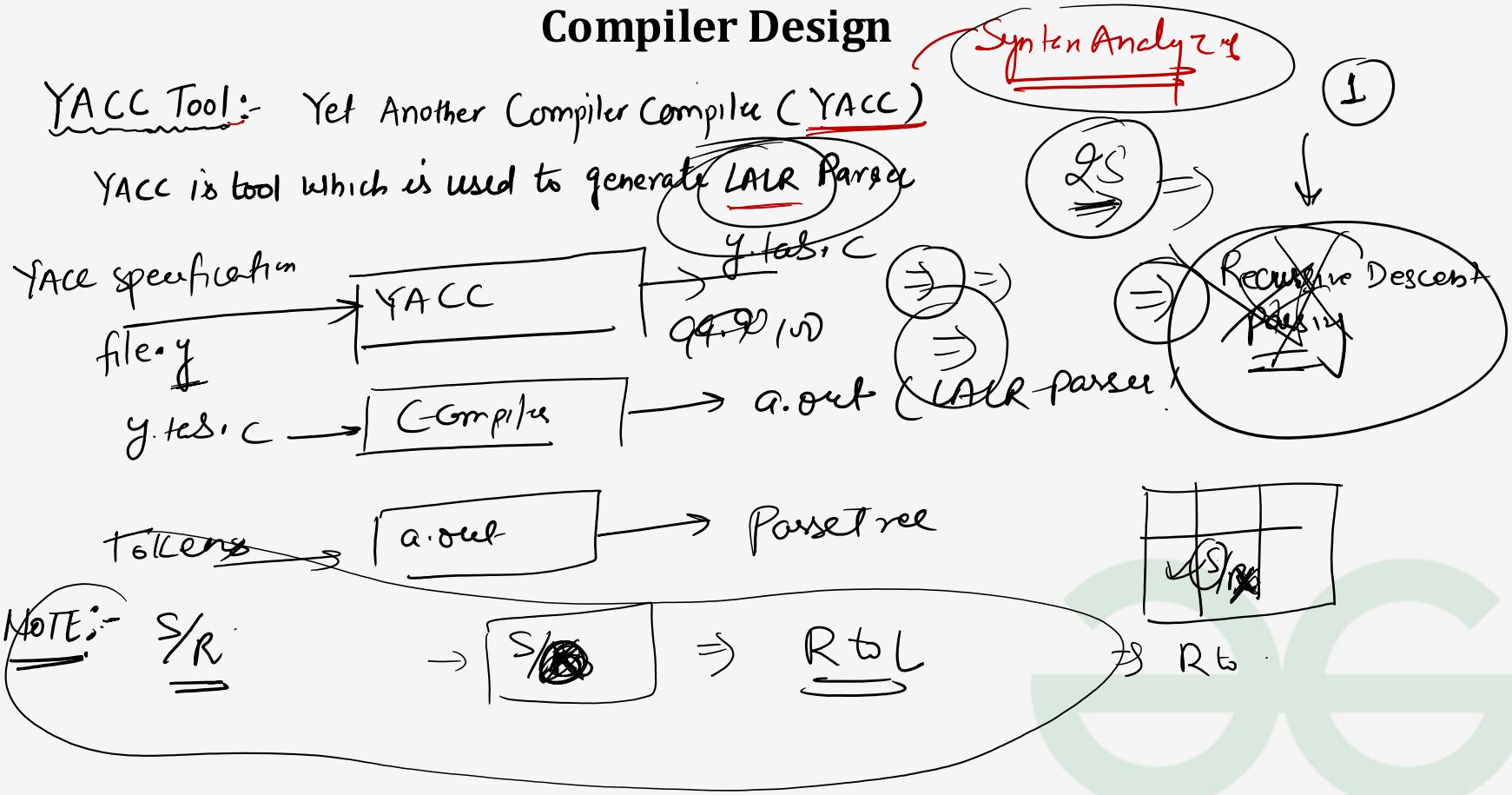
7) $LL(K) \subset LR(K)$

8) Among SLR(1), LALR(1) & CLR(1) the most easy method is SLR(1) and the most powerful method is CLR(1).

Compiler Design

Compiler Design

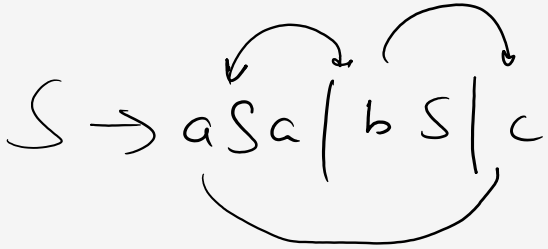
Compiler Design



Compiler Design

The grammar S → aSa | bS | c is

- [ ] LL(1) but not LR(1)
- [ ] LR(1)but not LR(1)
- [x] Both LL(1)and LR(1)
- [ ] Neither LL(1)nor LR(1)



LL(1) ⇒ LR(1)

Compiler Design

An LALR(1) parser for a grammar G can have shift-reduce (S-R) conflicts if and only if

- [ ] the SLR(1) parser for G has S-R conflicts
- [ ] the LR(1) parser for G has S-R conflicts
- [ ] the LR(0) parser for G has S-R conflicts
- [ ] the LALR(1) parser for G has reduce-reduce conflicts

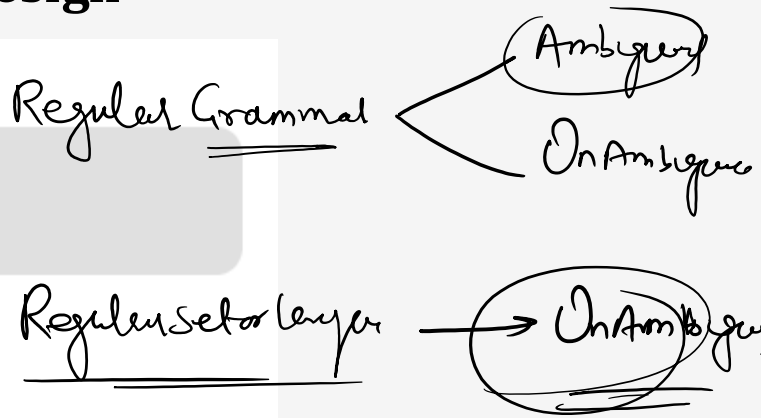
Compiler Design

Consider the following two statements:

```
P: Every regular grammar is LL(1)
Q: Every regular set has a LR(1) grammar
```

Which of the following is TRUE?

- [ ] Both P and Q are true
- [ ] P is true and Q is false
- [x] P is false and Q is true
- [ ] Both P and Q are false



Compiler Design

Consider the grammar

LR(1) items

```
S → (S) | a
```

n_1 n_2 n_3

Let the number of states in SLR(1), LR(1) and LALR(1) parsers for the grammar be n1, n2 and n3 respectively. The following relationship holds good

- [ ] n1 < n2 < n3
- [x] n1 = n3 < n2
- [ ] n1 = n2 = n3
- [x] n1 ≥ n3 ≥ n2

$n_1 = n_3 \leq n_2$

Compiler Design

Consider the grammar shown below.

```
S → C C
C → c C | d
```

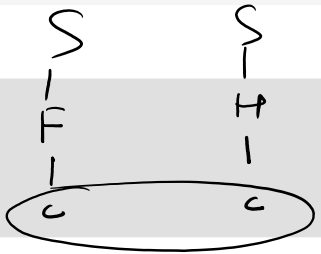
The grammar is

- [ ] LL(1)
- [ ] SLR(1) but not LL(1)
- [ ] LALR(1) but not SLR(1)
- [ ] LR(1) but not LALR(1)

Compiler Design

Consider the following grammar G.

G:
$$S \rightarrow F \mid H$$
$$F \rightarrow p \mid c$$
$$H \rightarrow d \mid c$$



Where S, F and H are non-terminal symbols, p, d and c are terminal symbols. Which of the following statement(s) is/are correct?

S1: LL(1) can parse all strings that are generated using grammar G.
S2: LR(1) can parse all strings that are generated using grammar G.

- [ ] Only S1
- [ ] Only S2
- [ ] Both S1 and S2
- [ ] Neither S1 and S2

Compiler Design

Which is True about SR and RR-conflict:

- [x] If there is no SR-conflict in CLR(1) then definitely there will be no SR-conflict in LALR(1).
- [x] RR-conflict might occur if lookahead for final items(reduce-moves) is same.
- [x] Known that CLR(1) has no RR-conflict, still RR-conflict might occur in LALR(1).
- [x] All of the above.

Ans

Thank You !