CLR(1)

Dr. Meera Thapar Khanna

CSE Department

Pandit Deendayal Energy University

Drawbacks of SLR(1) Parser

1. On single input, State may be included a Final Item and a Non- Final Item. This may result in a Shift-Reduce Conflict .
2. A State may be included Two Different Final Items. This might result in a Reduce-Reduce Conflict
3. SLR(1) Parser reduces only when the next token is in Follow of the left-hand side of the production.
4. SLR(1) can reduce shift-reduce conflicts but not reduce-reduce conflicts

CLR(1)

- ✓These two conflicts are reduced by CLR(1) Parser by keeping track of look-ahead information in the states of the parser.
- ✓This is also called as LR(1) grammar.
- ✓LR(1) Parser greatly increases the strength of the parser, but also the size of its parse tables.
- ✓The LR(1) techniques does not rely on FOLLOW sets, but it keeps the Specific Look-ahead with each item.

CLR(1)-Canonical LR(1) Parsing

- ✓Write the Context free Grammar for the given input string.
- ✓Check for the Ambiguity.
- ✓Add Augment production.
- ✓Create Canonical collection of LR (1) items.
- ✓Draw DFA.
- ✓Construct the CLR (1) Parsing table.
- ✓Based on the information from the Table, with help of Stack and Parsing algorithm generate the output.

LR(1)

The LR (1) item is defined by

- production,
- position of data
- and a terminal symbol.

The terminal is called as Look ahead symbol.

General form of LR (1) item is

S -> α•Aβ , $

A-> •γ, FIRST(β,$)

- The Look-ahead Component (after ,) represents a possible look-ahead after the entire right-hand side has been matched.
- The $ appears as look-ahead only for the augmenting production because there is no look-ahead after the end-marker

Rules to create canonical collection:

- Every element of I is added to closure of I.
- If an LR (1) item [X-> A•BC, a] exists in I, and there exists a production B->b1b2….., then add item [B->• b1b2, z] where z is a terminal in FIRST(Ca), if it is not already in Closure(I). keep applying this rule until there are no more elements added.

Example

Grammar

$S \rightarrow CC$

$C \rightarrow cC$

$C \rightarrow d$

Step1: Augmented Grammar

Augmented Grammar:

S’ →•S$	0
S → •CC	1
C → •cC	2
C → •d	3

Step2: Construct LR(1) closure items

I0:

S’ → •S, $	0	(Look-ahead of 1st production is always $)
S → •CC, $	1	(nothing after S in 0th production so look-ahead for 1st prodution is same $)
C → •cC, c|d	2	(C is there after C so look-ahead for 2nd production is First(C) i.e. {c,d})
C → •d, c|d	3	(C is there after c so look-ahead for 3rd production is First(C) i.e. {c,d})

- Goto(i0, S) = S’ → S•, $ = i1

- Goto(i0, C) = S → C•C, $ = i2
  C → •cC, $
  C → •d, $

- Goto(i0, c) = C → c•C, c|d = i3
  C → •cC, c|d
  C → •d, c|d

- Goto(i0, d) = C → d•, c|d = i4
- Goto(i2, C) = C → CC•, $ = i5
- Goto(i2, c) = C → c•C, $ = i6
  C → •cC, $
  C → •d, $
- Goto(i2, d) = C → d•, c|d = i7

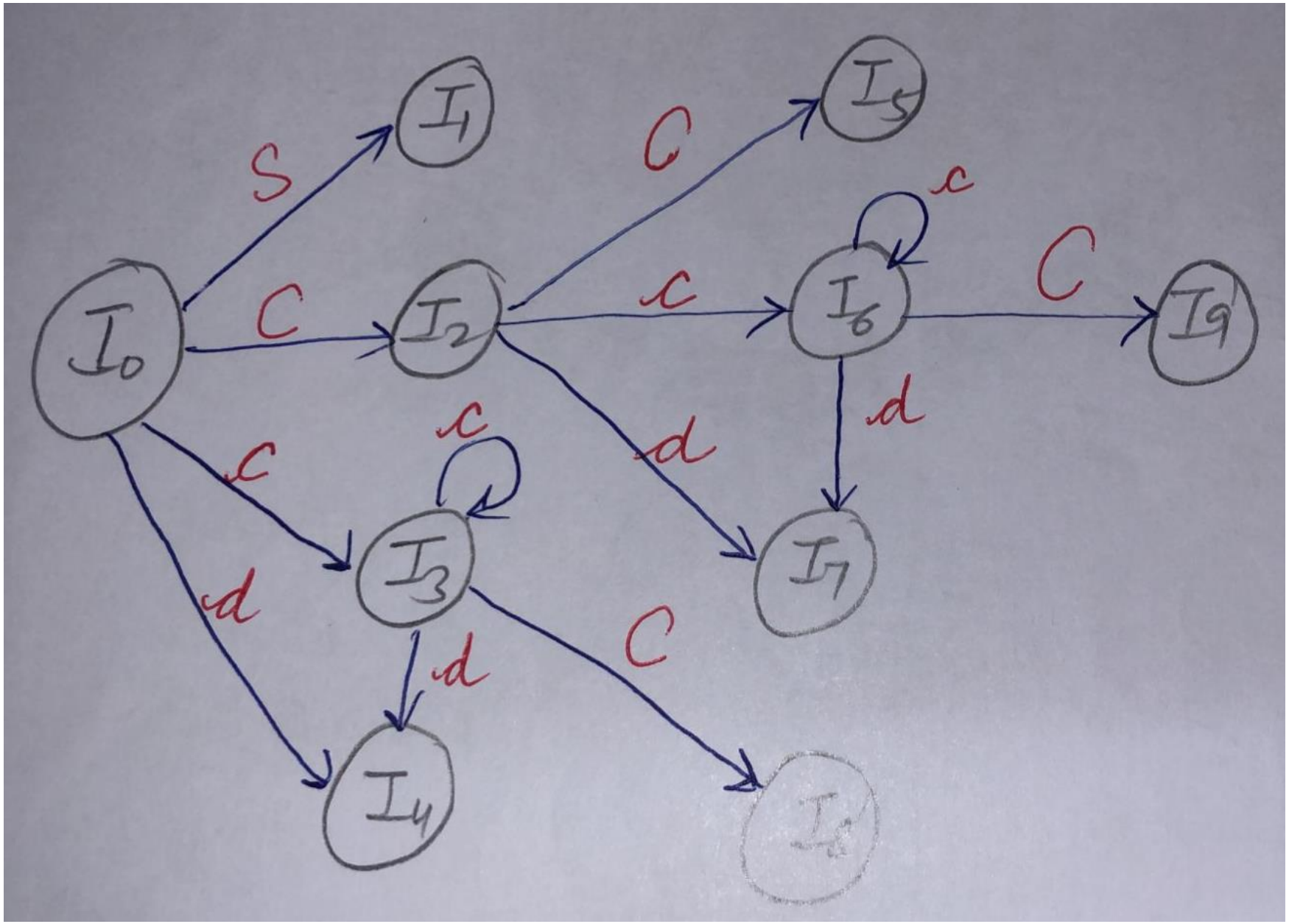
- Goto(i3, C) = C → cC•, c|d = i8

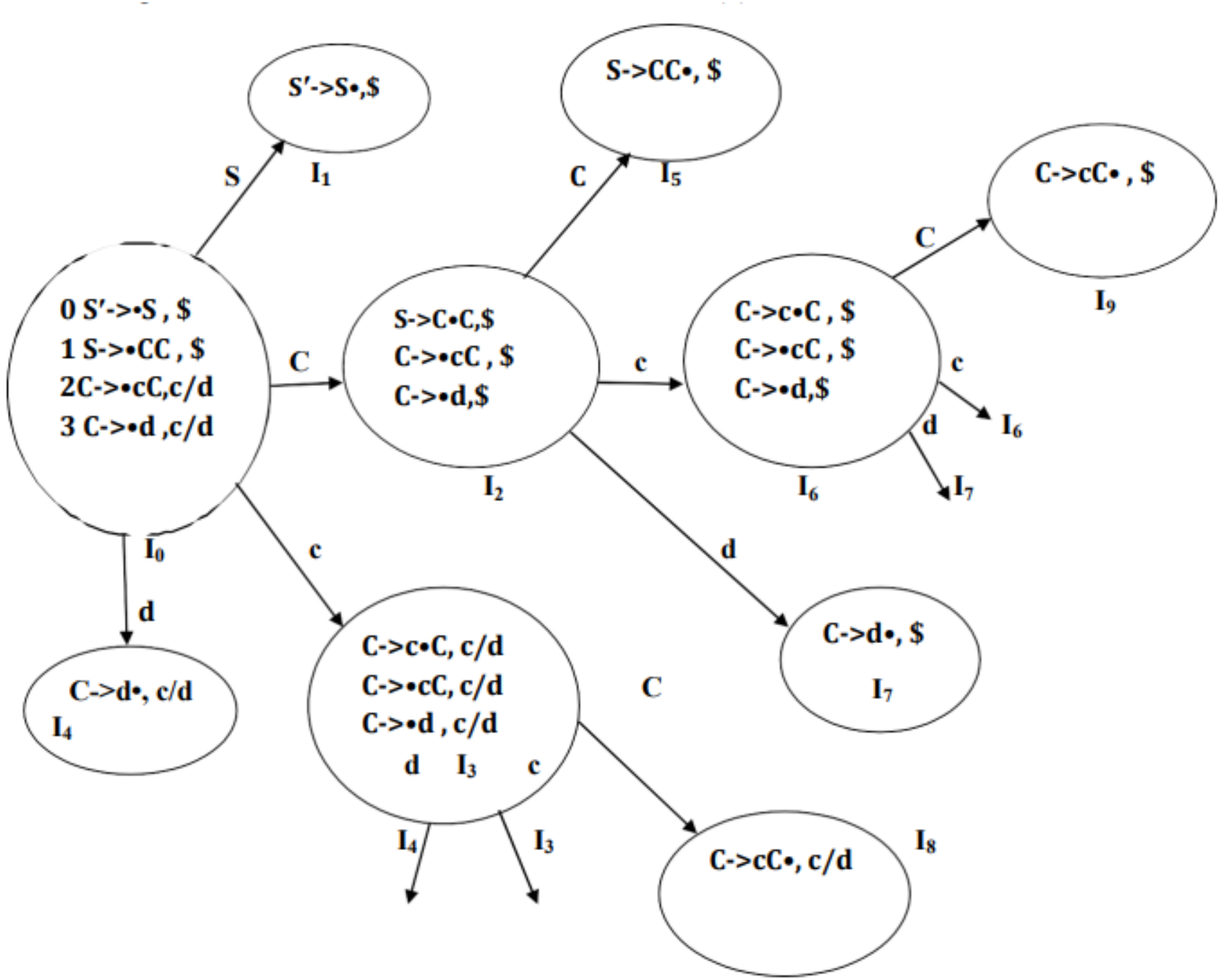
- Goto(i3, c) = C → c•C, c|d = i3
  C → •cC, c|d
  C → •d, c|d

- Goto(i3, d) = C → d•, c|d = i4

- Goto(i6, C) = C → cC•, $ = i9
- Goto(i6, c) = C → c•C, $ = i6
  C → •cC, $
  C → •d, $
- Goto(i6, d) = C → d•, $ = i7

DFA




S′->S•,$
S
I1
S->CC•, $
C
I5
C->cC• , $
C
I9
0 S′->•S , $
1 S->•CC , $
2C->•cC,c/d
3 C->•d ,c/d
C
S->C•C,$
C->•cC , $
C->•d,$
I2
c
C->c•C , $
C->•cC , $
C->•d,$
I6
c
I6
d
I7
I0
d
C->d•, c/d
I4
c
C->c•C, c/d
C->•cC, c/d
C->•d , c/d
d I3 c
C
d
C->d•, $
I7
I4
I3
C->cC•, c/d
I8

CLR(1) Table Construction

- if there is an item [A->α•Xβ,b] in Ii and goto(Ii,X) is in Ij then action [Ii, X]= Shift j, Where X is Terminal.
- if there is an item [A->α•, b] in Ii and (A≠S') set action [Ii, b]= reduce along with the production number.
- if there is an item [S'->S•, $] in Ii then set action [Ii, $]= Accept.
- if there is an item [A->α•Xβ,b] in Ii and goto(Ii,X) is in Ij then goto [Ii,X]= j, Where X is Non Terminal.

CLR(1) Parsing Table

states	c	d	$	S	C
	Action Part			GOTO Part	
I0	s3	s4		1	2
I1			acc		
I2	s6	s7			5
I3	s3	s4			8
I4	r3	r3			
I5			r1		
I6	s6	s7			9
I7			r3		
I8	r2	r2			
i9			r2		

In the LR parsing table of a grammar G has no Conflict, Therefore the Grammar is called LR(1) Grammar.

String Acceptance ‘cdd’

Stack	Input	Action
0	cdd$	Shift S3
0c3	dd$	Shift S4
0c3d4	d$	Reduce with R3,C->d, pop 2*β symbols from the stack
0c3C	d$	Goto (I3, C)=8Shift S6
0c3C8	d$	Reduce with R2 ,C->cC, pop 2*β symbols from the stack
0C	d$	Goto (I0, C)=2
0C2	d$	Shift S7
0C2d7	$	Reduce with R3,C->d, pop 2*β symbols from the stack
0C2C	$	Goto (I2, C)=5
0C2C5	$	Reduce with R1,S->CC, pop 2*β symbols from the stack
0S	$	Goto (I0, S)=1
0S1	$	Accept

Example

S -> L=R

S -> R

L -> *R

L -> id

R -> L

$S \rightarrow L=R \mid R$
$L \rightarrow *R \mid id$
$R \rightarrow L$

$S' \rightarrow S$
$S \rightarrow L=R$
$S \rightarrow R$

$L \rightarrow *R$
$L \rightarrow id$
$R \rightarrow L$

$S' \rightarrow .S, \$$
$S \rightarrow .L=R, \$$
$S \rightarrow .R, \$$
$L \rightarrow .*R, =$
$L \rightarrow .id, =$
$R \rightarrow .L, \$$
$L \rightarrow .*R, \$$
$L \rightarrow .id, \$$

can be written as

I_0: $S' \rightarrow .S, \$$; $S \rightarrow .L=R, \$$; $S \rightarrow .R, \$$; $L \rightarrow .*R, =/\$$; $L \rightarrow .id, =/\$$; $R \rightarrow .L, \$$

I_1: $S' \rightarrow S., \$$

I_2: $S \rightarrow L.=R, \$$; $R \rightarrow L., \$$

I_3: $S \rightarrow R., \$$

I_4: $L \rightarrow *.R, =/\$$; $R \rightarrow .L, \$/=$; $L \rightarrow .*R, \$/=$; $L \rightarrow .id, \$/=$

I_5: $L \rightarrow id., =/\$$

I_6: $S \rightarrow L=.R, \$$; $R \rightarrow .L, \$$; $L \rightarrow .*R, \$$; $L \rightarrow .id, \$$

I_7: $S \rightarrow L=R., \$$

I_8: $L \rightarrow id., \$$

I_9: $R \rightarrow L., \$$

I_{10}: $L \rightarrow *.R, \$$; $R \rightarrow .L, \$$; $L \rightarrow .id \$$; $L \rightarrow .*R, \$$

I_{11}: $R \rightarrow L., =/\$$

I_{12}: $L \rightarrow *R., =/\$$

I_{13}: $L \rightarrow *R., \$$

I_{10}

CLR table conflicts

When multiple entries occur in the table. That is said to be a Conflict.

- Shift-Reduce Conflict in CLR (1) Parsing
- Reduce-Reduce Conflict in CLR (1) Parsing

Shift-Reduce Conflict in CLR (1) Parsing

Shift Reduce Conflict in the CLR (1) parsing occurs when a state has :

1. A Reduced item of the form A-> α•, a
2. An incomplete item of the form A-> β•aα as shown below

    A-> β•a α, $ (on symbol ‘a’ going to next state say i_j)

    B-> b•, a

Reduce - Reduce Conflict in CLR (1) Parsing

Reduce- Reduce Conflict in the LR (1) parsing occurs when a state has two or more reduced items of the form:

1. A -> α•

2. B -> β• , two productions in a state (I) reducing on same look ahead symbol as shown below:

A-> α• ,a

B->β•,a