LR (0) Parsing

Steps involved in the LR (0) Parsing:

1. Add Augment production to the Grammar G.

2. Construct Canonical collection of LR (0) items

3. Construct the LR (0) Parsing table

4. Based on the information from the Table, with help of Stack and Parsing algorithm, generate the output

Example:

Construct LR (0) Parsing table for the following grammar

S → aB

B → bB | b

Solution:

Step-1: Add Augment production S' → S to the Grammar G.

Augmented Grammar:

S' → S

S → aB

B → bB | b

Step-2: Canonical collection of LR (0) items

LR (0) items:

An LR (0) item of a Grammar is a production G with dot at some position on the right side of the production. An item indicates how much of the input has been scanned up to a given point in the process of parsing.

For example, if the Production is X → AB then, The LR (0) items are:

1. X → •AB, indicates that the parser expects a string derivable from AB.

2. X → A•B, indicates that the parser has scanned the string derivable from A and expecting the string from B.

3. X → AB•, indicates that the parser has scanned the string derivable from AB.

If the grammar is X → ϵ then, the LR (0) item is

X → •, indicating that the production is reduced one.

(i). Assign production number to each production in grammar G and insert • symbol at the first position for every production in G

0. S' → .S

1. S → .aB

2. B → .bB

3. B → .b

(ii). Constructing Canonical collection of LR(0) Items is the process of grouping the LR (0) items together based on the **Closure** and **Goto** operations

State I_0:

Add augment production S' → .S to I_0 state and Compute the Closure

Since, the • symbol in the Right hand side production is followed by a Non terminal S, find the **Closure(S' → .S)**. Add productions starting with S in to I_0 state. So, the I_0 state becomes

S' → .S

S → .aB

Here, in the S production "." symbol is followed by a terminal value so close the state.

I_0:

S' → .S

S → .aB

I_1= Goto (I_0, S)

S' → S•

Since "." is at the end of the production, the production is reduced so close the state I_1.

I_1:

S' → S•

I_2= Goto (I_0, a) = closure (S → a•B)

Here, the "•" symbol is followed by the Non terminal B. So, add the productions which are Starting B.

I_2:

B → •bB

B → •b

Here, the "•" symbol in the B production is followed by the terminal value. So, close the state I_2. Hence I_2 becomes

I_2:

S → a•B

B → •bB

B → •b

I_3 = Goto (I_2, B) = Closure (S → aB•) = S → aB• Since • is at the end of the production, close the state.

I_3:

S → aB•

I_4= Goto(I_2, b) = Closure({B → b•B, B → b• }). Add productions starting with B in I_4. They are

B → •bB

B → •b

The Dot Symbol is followed by the terminal value. So, close the State I_4.

I_4:

B → b•B,

B → •bB

B → •b

B → b•

I_5= Goto (I_2,b) = Closure(B → b•)= B → b•

I_5:

B → b•

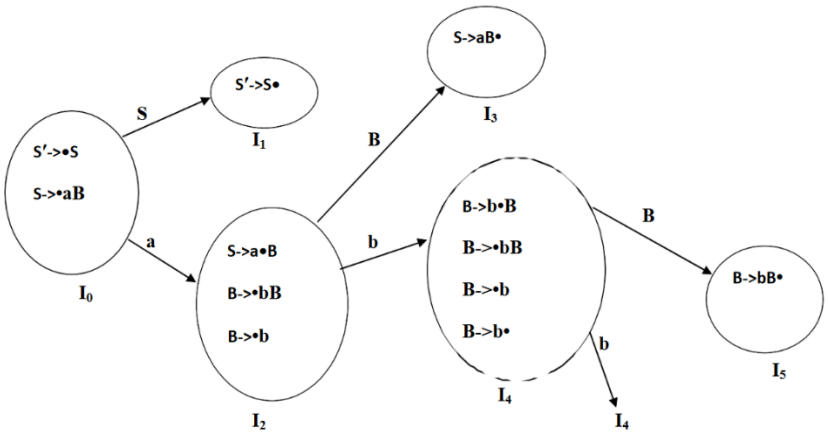
I_6= Go to (I_4, B) =Closure(B → bB•)=B → bB•

I_6:

B → bB•

Goto(I_4,b)= I_4

Following DFA gives the state transitions of the parser and is useful in constructing the LR parsing table.



Step-3: LR (0) Parsing Table:

States	ACTION			GOTO	
	a	B	$	S	B
I_0	S_2			1	
I_1			ACC		
I_2		S_4			3
I_3	R_1	R_1	R_1		
I_4	R_3	S_4/R_3	R_3		5
I_5	R_2	R_2	R_2		

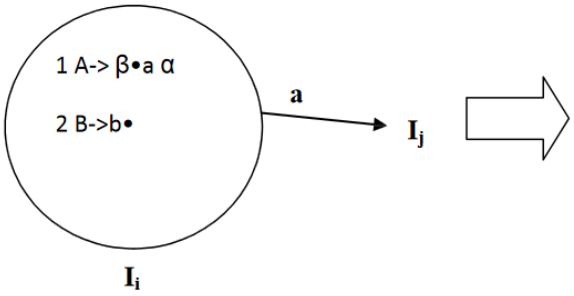
Note: If there are multiple entries in the LR (1) parsing table, then it will not accepted by the LR(1) parser. In the above table I3row is giving two entries for the single terminal value b' and it is called as Shift-Reduce conflict.

Shift-Reduce Conflict in LR(0) Parsing:

Shift Reduce Conflict in the LR(0) parsing occurs when a state has

A Reduced item of the form A → α• and

An incomplete item of the form A → β•aα as shown below:



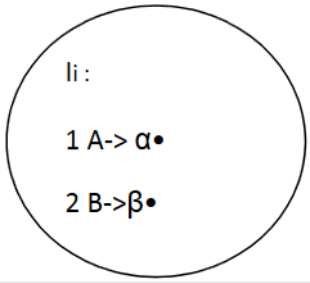
States	Action		GOTO	
	a	$	A	B
I_i	Sj/r2	r_2		
I_j				

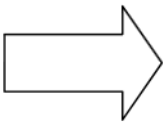
Reduce -Reduce Conflict in LR (0) Parsing:

Reduce-Reduce Conflict in the LR (0) parsing occurs when a state has two or more reduced items of the form

1. A → α•

2. B → β• as shown below:





States	Action		GOTO	
	a	$	A	B
I_i	r_1/r2	r_1/ r_2		

SLR Parser

Steps involved in construction of the SLR Parsing:

1. Add Augment production to the Grammar G.

2. Construct Canonical collection of LR (0) items

3. Construct the SLR Parsing table

4. Based on the information from the Table, with help of Stack and Parsing algorithm, generate the output

Example:

Construct SLR Parsing table for the following grammar

S → aB

B → bB | b

Solution:

Step-1: Add Augment production S' → S to the Grammar G.

Augmented Grammar:

S' → S

S → aB

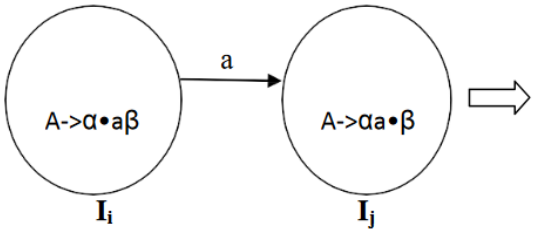
B → bB | b

Step-2: Canonical collection of LR (0) items

This step can be referred from LR(0) parser.

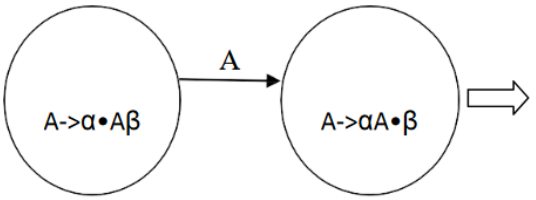
Step-3: SLR Parsing Table Construction:

If there is a transaction from one state ($\mathbf{I_i}$) to another state($\mathbf{I_j}$) on a terminal value then, we should write the shift entry in the action part as shown below:



States	ACTION		GO TO
	a	$	A
$\mathbf{I_i}$	$\mathbf{S_j}$		
$\mathbf{I_j}$			

If there is a transaction from one state ($\mathbf{I_i}$) to another state ($\mathbf{I_j}$) on a Non terminal value then, we should write the subscript value of $\mathbf{I_i}$ in the GO TO part as shown below: part as shown below:



States	ACTION		GO TO
	a	$	A
$\mathbf{I_i}$			**j**
$\mathbf{I_j}$			

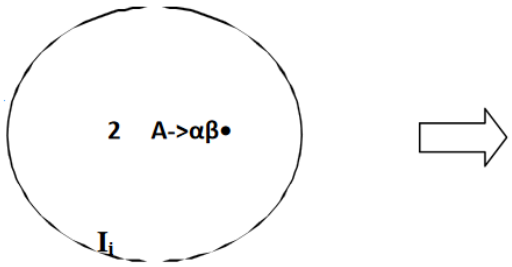
If there is one state ($\mathbf{I_{i)}}$, where there is one production **(A->αβ•)** which has no transitions to the next State. Then, the production is said to be a reduced production. For all terminals X in FOLLOW (A), write the reduce entry along with their production numbers. If the Augment production is reducing then write accept.

1 S -> •aAb

2 A->αβ•

Follow(S) = {$}

Follow (A) = (b}



States	ACTION			GO TO	
	a	b	$	S	A
$\mathbf{I_i}$		$\mathbf{r_2}$			

SLR (1) table for the Grammar

$$S \rightarrow aB$$
$$B \rightarrow bB \mid b$$

Follow (S) = {$}, Follow (B) = {$}

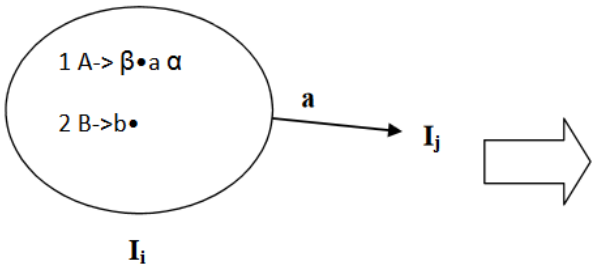
States	ACTION			GOTO	
	A	**b**	**$**	**S**	**B**
I_0	S_2			1	
I_1			ACCEPT		
I_2		S_4			3
I_3			R_1		
I_4		S_4	R_3		5
I_5			R_2		

Note: When Multiple Entries occurs in the SLR table. Then, the grammar is not accepted by SLR (1) Parser.

Conflicts in the SLR (1) Parsing: When multiple entries occur in the table. Then, the situation is said to be a Conflict.

Shift-Reduce Conflict in SLR (1) Parsing : Shift Reduce Conflict in the LR (1) parsing occurs when a state has

1. A Reduced item of the form $A \rightarrow \alpha\bullet$ and Follow(A) includes the terminal value 'a'.
2. An incomplete item of the form $A \rightarrow \beta\bullet a\alpha$ as shown below:



States	Action		GOTO	
	a	$	A	B
I_i	Sj/r2			

Reduce - Reduce Conflict in SLR (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• and Follow (A) ∩ Follow(B) ≠ null** as shown below:

If The Grammar is

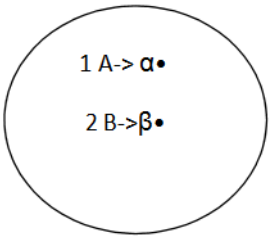
S-> αAaBa

A-> α

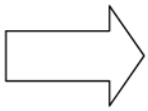
B-> β

Follow(S)= {$}

Follow(A)={a} and Follow(B)= {a}



I_i



States	Action		GOTO	
	a	$	A	B
I_i	$r_1/r2$			

Canonical LR Parsing

Steps involved in construction of the LR (1) Parsing:

1. Add Augment production to the Grammar G.

2. Construct Canonical collection of LR (1) items

3. Construct the CLR Parsing table

4. Based on the information from the Table, with help of Stack and Parsing algorithm, generate the output

LR (1) items :

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,$)

Rules to create canonical collection:

1. Every element of I is added to closure of I

2. If an LR (1) item **[X → A•BC, a]** exists in I, and there exists a production B → .b_1b_2...., then add item **[B → • b_1b_2, 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:

Construct CLR Parsing table for the following grammar

S → CC

C → cC

C → d

The canonical collection of LR (1) items can be created as follows:

0. S' → S

1. S → CC

2. C → cC

3. C → d

I0 State: Add Augment production and compute the Closure, the look ahead symbol for the Augment Production is $.

S′→ •S, $

Closure(S′->•S, $): The dot symbol is followed by a Non terminal S. So, add productions starting with S in I_0 State.

S → •CC, FIRST ($)

Closure(S → •CC): The dot symbol is followed by a Non terminal C. So, add productions starting with C in I_0 State.

C → •cC, FIRST(C, $)

C → •d, FIRST(C, $)

FIRST(C) = {c, d} so, the items are

C → •cC, c/d

C → •d, c/d

The dot symbol is followed by a terminal value. So, close the I_0 State. So, the productions in the I_0 state are

I_0:

S′→ •S, $

S → •CC, $

C → •cC, c/d

C → •d, c/d

I_1=Goto (I_0, S) =S′ → S•, $

I_2=Goto(I_0, C) = Closure (**S → •CC, $)**

C → •cC, $

C → •d,$

Now,

I2:

S → C•C, $

C → •cC, $

C → •d, $

I3 = Goto (I0,c) = Closure(C → c•C, c/d)

C → •cC, c/d

C → •d , c/d

So, the I3 State is

C → c•C, c/d

C → •cC, c/d

C → •d , c/d

I4=Goto(I0, d)= Closure(C → d•, c/d) =

C → d•, c/d

I5= Goto (I2,C)= Closure(S → CC•,$)=

S → CC•, $

I6= Goto (I2, c)= Closure(C → c•C , $)=

C → •cC, $

C → •d , $

So, the I6 State is

C → c•C , $

C → •cC , $

C → •d, $

I7 = Goto(I2 , d)= Closure(C → d•,$) =

C → d•, $

Goto(I3, c)= Closure(C → •cC, c/d)= I3.

I8= Goto(I3 , C)= Closure(C → cC•, c/d) =

C → cC•, c/d

Goto(I3 , c)= Closure(C → c•C, c/d) = I3

Goto(I3, d)= Closure(C → d•, c/d) = I4

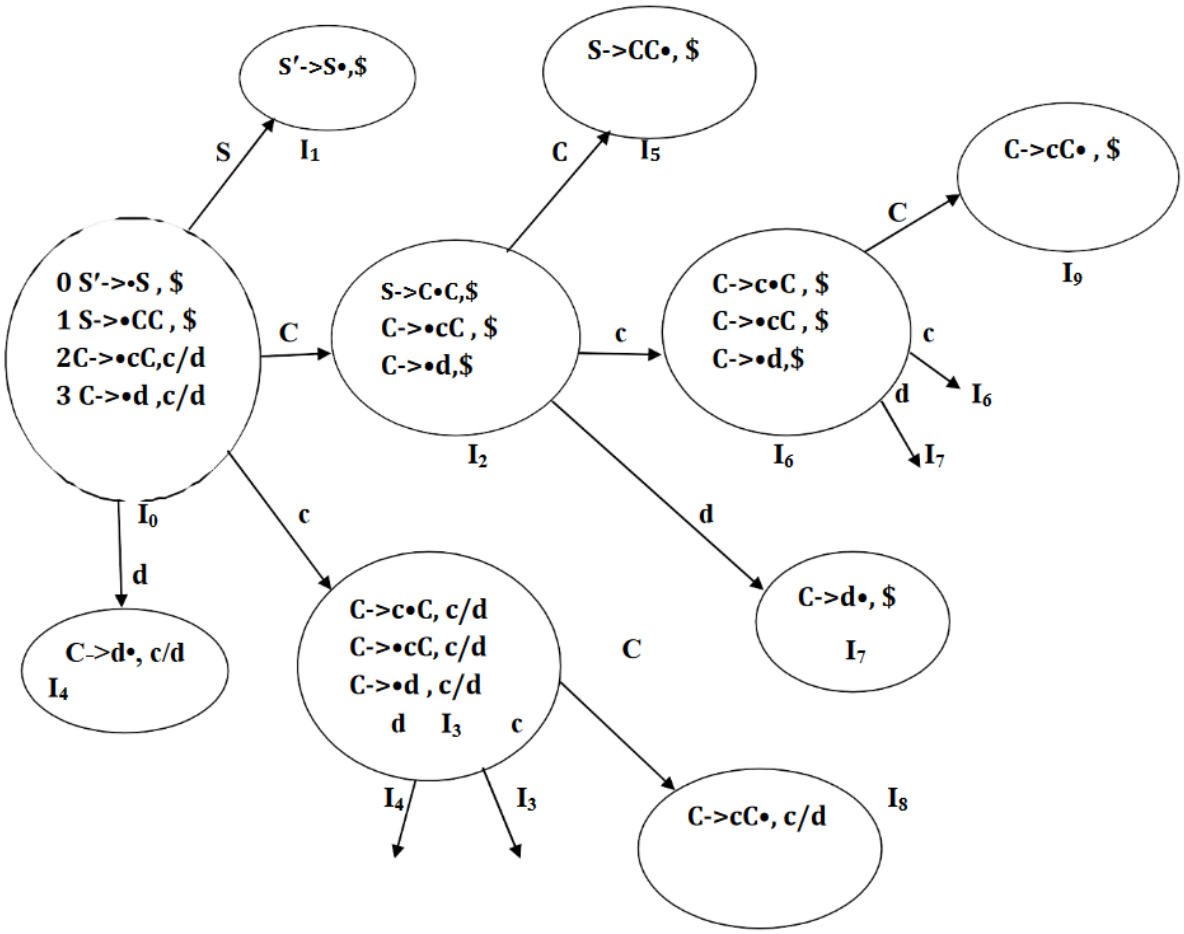
I9 = Goto(I6, C)= Closure(C → cC• , $) =

C → cC•, $

Goto (I6, c) = Closure(C → c•C , $) = I6

Goto(I6 , d)= Closure(C → d•,$) = I7

Following DFA gives the state transitions of the parser and is useful in constructing the LR parsing table.



Step-3: Construction of CLR (1) Table

Rule-1: If there is an item [A → α•Xβ,b] in I_i and goto(I_i,X) is in I_j then action [I_i][X]= Shift j, where X is Terminal.

Rule-2: If there is an item [A → α•, b] in I_i and (A≠S') set action [I_i][b] = reduce along with the production number.

Rule-3: If there is an item [S' → S•, $] in I_i then set action [I_i][$]= Accept.

Rule-4: If there is an item [A → α•Xβ,b] in I_i and go to(I_i,X) is in I_j then goto [I_i][X]= j, where X is Non Terminal.

States	ACTION			GOTO	
	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			5
I_5			R_1		
I_6	S_6	S_7			9
I_7			R_3		
I_8	R_2	R_2			
I_9			R_2		

STACK	INPUT	ACTION
$0	cdd$	Shift S_3
$0c3	dd$	Shift S_4
$0c3d4	d$	Reduce with R3,C->d, pop 2*β symbols from the stack
$0c3C	d$	Goto (I_3, C)=8Shift S_6
$0c3C8	d$	Reduce with R_2 ,C->cC, pop 2*β symbols from the stack
$0C	d$	Goto (I_0, C)=2
$0C2	d$	Shift S_7
$0C2d7	$	Reduce with R3,C->d, pop 2*β symbols from the stack
$0C2C	$	Goto (I_2, C)=5
$0C2C5	$	Reduce with R_1,S->CC, pop 2*β symbols from the stack
$0S	$	Goto (I_0, S)=1
$0S1	$	Accept

LALR Parsing

The CLR Parser avoids the conflicts in the parse table. But it produces more number of States when compared to SLR parser. Hence more space is occupied by the table in the memory. So LALR parsing can be used. Here, the tables obtained are smaller than CLR parse table. But it also as efficient as CLR parser. Here LR (1) items that have same productions but different look-aheads are combined to form a single set of items.

For example, consider the grammar in the previous example. Consider the states I_4 and I_7 as given below:

I_4= Goto(I_0, d)= Colsure(C->d•, c/d) = C->d•, c/d

I_7 = Go to (I_2 , d)= Closure(C->d•,$) = C->d•, $

These states are differing only in the look-aheads. They have the same productions. Hence these states are combined to form a single state called as I_{47}.

Similarly the states I_3 and I_6 differing only in their look-aheads as given below:

I_3= Goto(I_0,c)=

C->c•C, c/d
C->•cC, c/d
C->•d , c/d

I_6= Goto (I_2, c)=
C->c•C , $
C->•cC , $
C->•d,$

These states are differing only in the look-aheads. They have the same productions. Hence these states are combined to form a single state called as I_{36}.

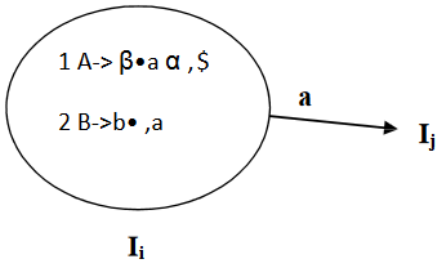
Similarly the States I_8 and I_9 differing only in look-aheads. Hence they combined to form the state I_{89}.

States	ACTION			GOTO	
	c	d	$	S	C
I_0	S_{36}	S_{47}		1	2
I_1			ACCEPT		
I_2	S_{36}	S_{47}			5
I_{36}	S_{36}	S_{47}			89
I_{47}	R_3	R_3	R_3		5
I_5			R_1		
I_{89}	R_2	R_2	R_2		

Shift-Reduce Conflict in CLR (1) Parsing

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

3. A Reduced item of the form A → α•, a and
4. An incomplete item of the form A → β•aα as shown below:

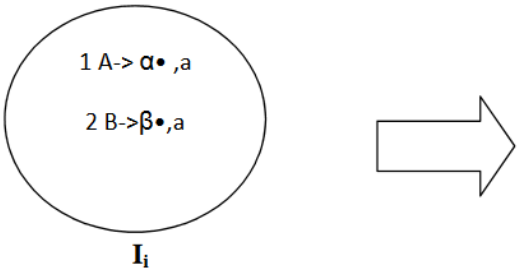


States	Action		GOTO	
	a	$	A	B
I_i	Sj/r2			

Reduce / Reduce Conflict in CLR (1) Parsing

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

3. **A → α•**
4. **B → β• If two productions in a state (I) reducing on same look ahead symbol as shown below:**



States	Action		GOTO	
	a	$	A	B
I_i	r_1/r2			