Construction of CLR(1)

The canonical set of items is the parsing technique in which a lookahead symbol is generated while constructing set of items. it can be referred as LR(1).

Steps for CLR parsing technique

1. construction of canonical set of items along with lookahead
2. Building canonical LR parsing table
3. parsing the input string

Construction of canonical set of items

1. For the grammar G initially add $S' \rightarrow .S, \$$ in the Set of item C
2. For each set of items I_i in C and for each grammar Symbol X (may be terminal or non-terminal) add closure (I_i, X) This process should be repeated by applying goto(I_i, X) for each X in I_i such that goto(I_i, X) is not empty and not in C

3. The closure function is

   for each item $A \rightarrow \alpha . X\beta, a$ and rule $X \rightarrow .\gamma, b$

   $b \in FIRST(\beta a)$

4. goto function is

   for each item $[A \rightarrow \alpha . X\beta, a]$ is in I and rule $[A \rightarrow \alpha X . \beta, a)$ is not in goto items then add $[A \rightarrow \alpha X . \beta, a]$ to goto items.

This process is repeated until no more set of items can be added to the collection C

Example

$S \rightarrow CC$
$C \rightarrow aC$
$C \rightarrow d$

Closure (I)

$I_0 : S' \rightarrow \cdot S, \$$ $\quad \alpha = \epsilon$
$A \rightarrow \alpha \cdot X\beta, a$ $\quad X = S$
$\beta = \epsilon$
$S \rightarrow \cdot CC, \$$ $\quad a = \$$
$A \rightarrow \alpha \cdot X\beta, a$
$C \rightarrow \cdot aC, a/d$ $\quad b \in FIRST(\beta a)$
$FIRST(\epsilon \$)$
$C \rightarrow \cdot d, a/d$ $\quad FIRST(\$)$
$b \in \{\$\}$

$b \in FIRST(\beta a)$
$b \in FIRST(C\$)$
$b \in FIRST(aC\$ / d\$)$
$b \in \{a, d\}$

$\Rightarrow$

$I_0 : S' \rightarrow \cdot S, \$$
$S \rightarrow \cdot CC, \$$
$C \rightarrow \cdot aC, a/d$
$C \rightarrow \cdot d, a/d.$

$I_1 : goto(I_0, S)$

$S' \rightarrow S \cdot, \$$

$I_2 : goto(I_0, C)$

$S \rightarrow C \cdot C, \$$ $\quad b \in FIRST(\epsilon \$)$
$A \rightarrow \alpha \cdot X\beta, a$ $\quad b \in FIRST(\$)$
$C \rightarrow \cdot aC, \$$ $\quad b \in \{\$\}$
$C \rightarrow \cdot d, \$$

$I_3 : goto(I_0, a)$

$C \rightarrow a \cdot C, a/d$ $\quad b \in FIRST(\epsilon a/d)$
$A \rightarrow \alpha \cdot X\beta, a$ $\quad b \in FIRST(a/d)$
$C \rightarrow \cdot aC, a/d$ $\quad b \in \{a/d\}$
$C \rightarrow \cdot d, a/d$

$I_4 : goto(I_0, d)$

$C \rightarrow d \cdot, a/d$

$I_5 : goto(I_2, C)$

$S \rightarrow CC \cdot, \$$

$I_6 : goto(I_2, a)$

$C \rightarrow a \cdot C, \$$ $\quad b \in FIRST(\epsilon \$)$
$A \rightarrow \alpha \cdot X\beta, a$ $\quad b \in FIRST(\$)$
$C \rightarrow \cdot aC, \$$ $\quad b \in \{\$\}$
$C \rightarrow \cdot d, \$$

$I_7 : goto(I_2, d)$

$C \rightarrow d \cdot, \$$

$I_8 : goto(I_3, C)$

$C \rightarrow aC \cdot, a/d$

$I_3 : goto(I_3, a)$

$C \rightarrow a \cdot C, a/d$ $\quad b \in FIRST(\beta a)$
$A \rightarrow \alpha \cdot X\beta, a$ $\quad b \in FIRST(\epsilon a/d)$
$C \rightarrow \cdot aC, a/d$ $\quad b \in FIRST(a/d)$
$C \rightarrow \cdot d, a/d$ $\quad b \in \{a/d\}$

I_4 : goto (I_3, d)

$C \rightarrow d.$, a/d

I_9 : goto (I_6, C)

$C \rightarrow aC.$, \$

I_6 : goto (I_6, a)

$C \rightarrow a.C$, \$ $b \in FIRST(\beta a)$

$A \rightarrow \alpha . X\beta$, a $b \in FIRST(\epsilon \$)$

$C \rightarrow .aC$, \$ $b \in \{\$\}$

$C \rightarrow .d$, \$

I_7 : goto (I_6, d)

$C \rightarrow d$, \$

Construction of CLR parsing table

1. $C = \{I_0, I_1, I_2, \ldots I_n\}$ where C is a collection of set of Canonical Items
2. The parsing action are based on each item I_i.
   a) if $[A \rightarrow \alpha . a\beta, b]$ is in I_i and goto $(I_i, a) = I_j$ then create an entry in the action table action$[I_i, a]$ = shift j.
   b) If there is a production $[A \rightarrow \alpha ., a]$ in I_i then in the action table action$[I_i, a]$ = reduce by $A \rightarrow \alpha$
   c) If there is a production $S' \rightarrow S.$, \$ in I_i then action[i, \$] = accept
3. The goto part is for state i is considered for non-terminals only. If goto $(I_i, A) = I_j$ then goto $[I_i, A] = j$
4. All the entries not defined by rule 2 and rule 3 are considered to be "error"

Parsing table

	Action			Goto	
	a	d	$	S	C
0	S_3	S_4		1	2
1			Accept		
2	S_6	S_7			5
3	S_3	S_4			8
4	r_3	r_3			
5			r_1		
6	S_6	S_7			9
7			r_3		
8	r_2	r_2			
9			r_2		

Input parsing.

input string is aadd

r_1 $S \rightarrow CC$
r_2 $C \rightarrow aC$
r_3 $C \rightarrow d$

Stack	Input buffer	Action	goto	Parsing action
$0	aadd $	{action[0,a]=S_3		shift
$0a3	add$	[3,a] = S_3		shift
$0a3a3	dd$	[3,d]= S_4		shift
$0a3a3d4	d $	[4,d]= r_3	[3,C]=8	reduce $C \rightarrow d$
$0a3a3C8	d $	[8,d]= r_2	[3,C]=8	reduce $C \rightarrow aC$

$0a3c8	d $	[8,d] = r2	[0,c]=2	reduce C→ac
$0C 2	d $	[2,d] = S7		shift
$0C2d7	$	[7,$] = r3	[2,C]=5	reduce C→d
$0C2C5	$	[5,$] = r1	[0,S]=1	reduce S→CC
$0S1	$	[1,$] = Accept		Accepted.

LALR

Stepts for LALR parsing technique

1. Construction of canonical set of items along with lookahead
2. Building LALR parse table
3. Input parsing using CLR parse table

Construction of canonical set of items

The constructionLALR(1) items is same as CLR(1). But the only difference is that, in construction ofLALR(1) items we have differed the two states if the second component is different but in this case we will merge the two states by merging of first & second component (lookaheads) from both the states

$$I_i + I_j = I_{ij}$$

Example

$S \rightarrow CC$

$C \rightarrow aC$

$C \rightarrow d$

I_0 : $S' \to .S, \$$
$S \to .CC, \$$
$C \to .aC, a/d$
$C \to .d, a/d$

I_1 : goto(I_0, S)
$S' \to S., \$$

I_2 : goto(I_0, C)
$S \to C.C, \$$
$C \to .aC, \$$
$C \to .d, \$$

I_3 : goto(I_0, a)
$C \to a.C, a/d$
$C \to .aC, a/d$
$C \to .d, a/d$

I_4 : goto(I_0, d)
$C \to d., a/d$

I_5 : goto(I_2, C)
$S \to CC., \$$

I_6 : goto(I_2, a)
$C \to a.C, \$$
$C \to .aC, \$$
$C \to .d, \$$

I_7 : goto(I_2, d)
$C \to d., \$$

I_8 : goto(I_3, C)
$C \to aC., a/d$

I_9 : goto(I_6, C)
$C \to aC., \$$

Now we will merge states 3 & 6 and 4 & 7 and 8 & 9

I_0 : $S' \to .S, \$$
$S \to .CC, \$$
$C \to .aC, a/d$
$C \to .d, a/d$

I_1 : goto(I_0, S)
$S' \to S., \$$

I_2 : goto(I_0, C)
$S \to C.C, \$$
$C \to .aC, \$$
$C \to .d, \$$

I_{36} : goto$(I_0, a), (I_2, a)$
$C \to a.C, a/d/\$$
$C \to .aC, a/d/\$$
$C \to .d, a/d/\$$

I_{47} : goto $(I_0, d)(I_2, d)$
$C \to d., a/d/\$$

I_5 : goto(I_2, C)
$S \to CC., \$$

I_{89} : goto $(I_3, C)(I_6, C)$
$C \to aC., a/d/\$$

Construction of LALR parsing table:-

1. construct the LR(1) set of items
2. Merge the two states I_i and I_j if the first component are matching and create a new state replacing one of the older state such as $I_{ij} = I_i \cup I_j$
3. Parsing actions are based on each item I_i
   (a) if $[A \rightarrow \alpha . a\beta, b]$ is in I_i and $goto(I_i, a) = I_j$ then create an entry in action table $action[I, a] = shift\ j$.
   (b) If there is a production $[A \rightarrow \alpha ., a]$ in I_i then in the action table is $action[I_i, a] =$ reduce by $A \rightarrow \alpha$.
   Here A should not be S'
   (c) If there is a production $S' \rightarrow S., \$$ in I_i then $action[i, \$] =$ accept.
4. The goto part of the LR table can be filled as: The goto part of the LR table can be filled as: for state i is considered for non-terminals only.
   If $goto(I_i, A) = I_j$ then $goto[I_i, A] = j$.
5. If the parsing action conflict then the algorithm fails to produce LALR parser and grammar is not LALR(1)

Parsing table

State	Action a	Action d	Action \$	goto S	goto C
0	S36	S47		1	2
1			Accept		
2	S36	S47			5
36	S36	S47			89
47	r_3	r_3	r_3		
5			r_1		
89	r_2	r_2	r_2		

r_1 $S \rightarrow CC$
r_2 $C \rightarrow aC$
r_3 $C \rightarrow d$

Input parsing

Stack	Input buffer	Action	goto	parsing Action
$0	aadd$	$[0,a] = S_{36}$		shift
$0a36	add$	$[36,a] = S_{36}$		shift
$0a36a36	dd$	$[36,d] = S_{47}$		shift
$0a36a36d47	d$	$[47,d] = r_{36}$	$[36,C] = 89$	Reduce by $C \rightarrow d$
$0a36a36C89	d$	$[89,d] = r_2$	$[36,C] = 89$	Reduce by $C \rightarrow aC$
$0a36C89	d$	$[89,d] = r_2$	$[0,C] = 2$	Reduce by $C \rightarrow aC$
$0C2	d$	$[2,d] = S_{47}$		shift
$0C2d47	$	$[47,\$] = r_3$	$[2,C] = 5$	Reduce by $C \rightarrow d$
$0C2C5	$	$[5,\$] = r_1$	$[0,S] = 1$	Reduce by $S \rightarrow CC$
$0S1	$	$[1,\$] =$ Accept		Accepted

8	s12							
10					r3			
11					r2			
12					r4			

The parsing table shows multiple entries in Action [59, a] and Action [59, c]. This is called **reduce/reduce conflict**. Because of this conflict we cannot parse input.

Thus it is shown that given grammar is LR(1) but not LALR(1)

5.3 Comparison of LR Parsers

It's a time to compare SLR, LALR and LR parser for the common factors such as size, class of CFG, efficiency and cost in terms of time and space.

Sr. No.	SLR parser	LALR parser	Canonical LR parser
1.	SLR parser is smallest in size.	The LALR and SLR have the same size.	LR parser or canonical LR parser is largest in size.
2.	It is an easiest method based on FOLLOW function.	This method is applicable to wider class than SLR.	This method is most powerful than SLR and LALR.
3.	This method exposes less syntactic features than that of LR parsers.	Most of the syntactic features of a language are expressed in LALR.	This method exposes less syntactic features than that of LR parsers.
4.	Error detection is not immediate in SLR.	Error detection is not immediate in LALR.	Immediate error detection is done by LR parser.
5.	It requires less time and space complexity.	The time and space complexity is more in LALR but efficient methods exist for constructing LALR parsers directly.	The time and space complexity is more for canonical LR parser.