THEORY OF COMPUTATION AND COMPILERS

Unit - II

CONTEXT FREE GRAMMARS AND PARSING

- Introduction
- Context-Free Grammars - Derivation, Parse trees, Ambiguity
- Types of Parsers
- LL(K) grammars and LL(1) parsing
- Bottom-up Parsing - handle pruning
- **LR Grammar Parsing** 
- LALR parsing
- Parsing ambiguous grammars
- Error Recovery in Parsing
- YACC programming specification

Dr. R. Madana Mohana
Professor, Artificial Intelligence & Data Science | I/c-Head, Artificial Intelligence & Machine Learning
CHAITANYA BHARATHI INSTITUTE OF TECHNOLOGY
Hyderabad - 500 075, Telangana, INDIA
www.cbit.ac.in

Unit-II: Syntax Analysis (or) Parser
Constructing CLR Parsing Table

Outline:

- Constructing CLR-Parsing Table
- Algorithm for Constructing CLR-Parsing Table
- Example problem

Constructing CLR-Parsing Table

For a given grammar **G**, the **CLR parsing table** can be constructed using the *following steps*:

1. For given grammar **G**, obtain the augmented grammar **G′** .
2. From the augmented grammar **G′** , construct the canonical collection of **LR(1) items** and **LR(1) automaton**.
3. Obtain the parsing table using the algorithm to construct **CLR parsing table**.

Algorithm for Constructing CLR-Parsing Table

The algorithm to construct **Canonical LR (CLR)** parsing table is almost similar to the construction of **SLR** parsing table with minor modifications. The algorithm is shown below:

Algorithm: **CLR_PARSING_TABLE (G′)**

INPUT: An augmented grammar **G′**.

OUTPUT: The **CLR parsing table** consisting of **ACTION** and **GOTO**.

Algorithm for Constructing CLR-Parsing Table

METHOD: The procedure is shown below:

1. Obtain **LR(1) items** for the **augmented grammar G′** and store in **C**.
2. Let the **LR(1)** items be **C = {I_0, I_1, I_2, …, I_n}**. Then **0, 1, 2,.. , n** will be the states of the parser. The parsing **ACTION** for each state **I**, can be obtained as shown below:

Algorithm for Constructing CLR-Parsing Table

METHOD:

2. a) If $[A \rightarrow \alpha.a\beta, b] \in I_i$ and
   $GOTO\ (I_i, a) = I_j$ then
   $ACTION\ [i, a] =$ "shift j" where 'a'
   must be a terminal.

   b) If $[A \rightarrow \alpha., a] \in I_i$, then
   $ACTION\ [i, a] =$ "reduce $A \rightarrow \alpha$".

Algorithm for Constructing CLR-Parsing Table

METHOD:

2. c) If **[S′ → S., $]∈$I_i$**, then **ACTION [i, $] = accept**.

3. For each state **i**, **GOTO** transitions can be obtained as shown below:

   a) if **GOTO (I_i, A)= I_j** then **GOTO [i, A] = j**

   end if

   a) If **I_i** contains **[S′ → S., $]** then **'i'** is the initial or start state of the parser.

Algorithm for Constructing CLR-Parsing Table

Note: In **CLR parsing table**, if there are multiple entries in **ACTION**, then the grammar is not **CLR(1)** grammar.

The **parsing table** obtained using this algorithm is **CLR parsing table** and the **parser** which uses this table is called **CLR parser** and the **grammar** is said to be **CLR(1)**.

Constructing CLR-Parsing Table

Example:

Construct **CLR parsing table** for the given grammar **G** as shown below:

1. **S → CC**
2. **C → cC**
3. **C → d**

Constructing CLR-Parsing Table

Example: ***Solution***

Augmented Grammar **G′**:

1. **S′ → S**
2. **S → CC**
3. **C → cC**
4. **C → d**

Constructing CLR-Parsing Table

Example: ***Solution***

Step 1: The **LR(1)** items can be obtained as shown below:

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)
S → C.C, $
C → .cC, $
C → .d, $

Constructing CLR-Parsing Table

Example: ***Solution***

I_3: GOTO (I_0, c)
C → c.C, c | d
C → .cC, c | d
C → .d, c | d

I_4: GOTO (I_0, d)
C → d., c | d

I_5: GOTO (I_2, C)
S → CC., $

I_6: GOTO (I_2, c)
C → c.C, $
C → .cC, $
C → .d, $

I_7: GOTO (I_2, d)
C → d., $

I_8: GOTO (I_3, C)
C → cC., c | d

Constructing CLR-Parsing Table

Example: ***Solution***

GOTO (I_3, c) = I_3

C → c.C, c | d

C → .cC, c | d

C → .d, c | d

GOTO (I_3, d) = I_4

C → d., c | d

I_9: GOTO (I_6, C)

C → cC., $

GOTO (I_6, c) = I_6

C → c.C, $

C → .cC, $

C → .d, $

GOTO (I_6, d) = I_7

C → d., $

Constructing CLR-Parsing Table

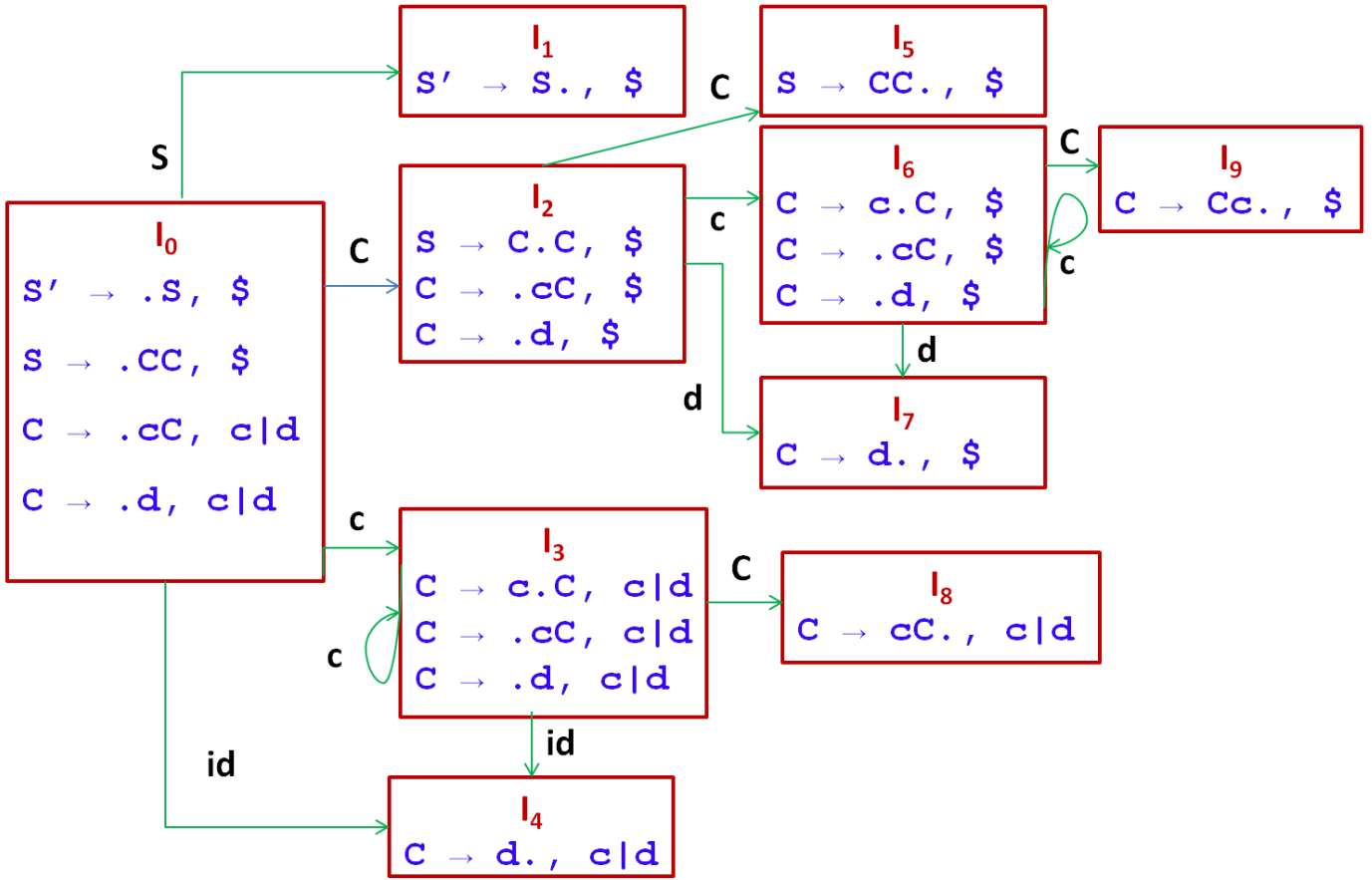

Fig: DFA for LR(1) items

Constructing CLR-Parsing Table

Example: ***Solution***

Step 2(a): Construction of **CLR parsing table**

The **ACTION** entries for terminals can be obtained as shown below: ***Algorithm Rule 2.a***

Transition GOTO $(I_i, a) = I_j$	ACTION [i, a] = shift j
$I_0, c = I_3$	$[0, c] = s_3$
$I_0, d = I_4$	$[0, d] = s_4$
$I_2, c = I_6$	$[2, c] = s_6$

Constructing CLR-Parsing Table

Example: ***Solution***

Step 2(a):

Transition GOTO $(I_i, a) = I_j$	ACTION [i, a] = shift j
$I_2, d = I_7$	$[2, d] = s_7$
$I_3, c = I_3$	$[3, c] = s_3$
$I_3, d = I_4$	$[3, d] = s_4$
$I_6, c = I_6$	$[6, c] = s_6$
$I_6, d = I_7$	$[6, d] = s_7$

Constructing CLR-Parsing Table

Example: ***Solution***

Step 2(b): Construction of **CLR parsing table**

The **ACTION** entries for the items ending with dot (.) are shown below: ***Algorithm Rule 2.b***

$[A \rightarrow \alpha.,a] \in I_i$	**ACTION** $[i, a] = r\ A \rightarrow \alpha$
$[C \rightarrow d., c\|d] \in I_4$	$[4, \{c, d\}] = r\ C \rightarrow d$ (i.e., r_3)
$[S \rightarrow CC., \$] \in I_5$	$[5, \{\$\}] = r\ S \rightarrow CC$ (i.e., r_1)
$[C \rightarrow d., \$] \in I_7$	$[7, \{\$\}] = r\ C \rightarrow d$ (i.e., r_3)

Constructing CLR-Parsing Table

Example: ***Solution***

Step 2(b):

$[A \rightarrow \alpha., a] \in I_i$	ACTION [i, a] = r $A \rightarrow \alpha$
$[C \rightarrow cC., c/d] \in I_8$	[8, {c, d}] = r $C \rightarrow cC$ (i.e., r_2)
$[C \rightarrow cC., \$] \in I_9$	[9, {$}] = r $C \rightarrow cC$ (i.e., r_2)

Constructing CLR-Parsing Table

Example: ***Solution***

Step 2(c): [S′ → S., $]∈$I_i$ then **ACTION [i, $] = accept**: ***Algorithm Rule 2.c***

[S′ → S., $]∈$I_i$	ACTION [i, $] = accept
[S′ → S., $]∈I_1	[1, $] = accept

Constructing CLR-Parsing Table

Example: ***Solution***

Step 2 (d): The **GOTO** states can be computed using *rule-3* are shown below: ***Algorithm Rule 3***

Transition GOTO (I_i, A) = I_j	Table GOTO [i, A] = j
I_0, S = I_1	[0, S] = 1
I_0, C = I_2	[0, C] = 2
I_2, C = I_5	[2, C] = 5
I_3, C = I_8	[3, C] = 8
I_6, C = I_9	[6, C] = 9

Constructing CLR-Parsing Table

Example: ***Solution***

The final CLR parsing table:

	ACTION			GOTO	
	c	d	$	S	C
0	S_3	S_4		1	2
1			acc		
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		

Constructing CLR-Parsing Table

Example: ***Solution***

In the above **CLR parsing table**, there are no multiple entries in **ACTION**, then the grammar is **CLR(1)** grammar.

Note: Since there are no multiple entries in the parsing table, the above parsing table is **CLR(1)** parsing table and the parser which uses this table is called **CLR** parser and the grammar is said to be **LR(1)** grammar.

Constructing CLR-Parsing Table

Sequence of moves:

The sequence of moves made by **CLR** parser for the string ***ccdd*** using the given grammar and **CLR** parsing table is shown below:

Stack	Input	Action
0	ccdd$	S_3 => shift 3 onto the stack.
03	cdd$	S_3 => shift 3 onto the stack.
033	dd$	S_4 => shift 4 onto the stack.

Constructing CLR-Parsing Table

Sequence of moves:

Stack	Input	Action
0334	d$	r_3 => Reduce using 3rd production C → d. Pop |d| = 1 state from stack i.e., 4 and push GOTO (3, C) = 8 onto the stack.
0338	d$	r_2 => Reduce using 2nd production C → cC. Pop |cC| = 2 states from stack i.e., 8 & 3 and push GOTO (3, C) = 8 onto the stack.

Constructing CLR-Parsing Table

Sequence of moves:

Stack	Input	Action
0338	d$	r_2 => Reduce using 2nd production C → cC. Pop |cC| = 2 states from stack i.e., 8 & 3 and push GOTO (3, C) = 8 onto the stack.
038	d$	r_2 => Reduce using 2nd production C → cC. Pop |cC| = 2 states from stack i.e., 8 & 3 and push GOTO (0, C) = 2 onto the stack.

Constructing CLR-Parsing Table

Sequence of moves:

Stack	Input	Action
038	d$	r_2 => Reduce using 2nd production C → cC. Pop |cC| = 2 states from stack i.e., 8 & 3 and push GOTO (0, C) = 2 onto the stack.
02	$	S_7 => shift 7 onto the stack.
027	$	r_3 => Reduce using 3rd production C → d. Pop |d| = 1 state from stack i.e., 7 and push GOTO (2, C) = 5 onto the stack.

Constructing CLR-Parsing Table

Sequence of moves:

Stack	Input	Action
027	$	r_3 => Reduce using 3rd production C → d. Pop |d| = 1 state from stack i.e., 7 and push GOTO (2, C) = 5 onto the stack.
025	$	r_1 => Reduce using 1st production S → CC. Pop |CC| = 2 states from stack i.e., 5 & 2 and push GOTO (0, S) = 1 onto the stack.
01	$	ACCEPT, Parsing successful

Constructing CLR-Parsing Table

Practice Problem:

Consider the following grammar

1. **S → AA**
2. **A → Aa | b**

a) Construct sets of LR(1) items

b) Construct CLR(1) parsing table

c) Show the parsing steps for the string **“baaba”**

Summary...

Bottom-Up Parsing: **Constructing CLR Parsing Table**

- Constructing CLR-Parsing Table
- Algorithm for Constructing CLR-Parsing Table
- Example problem

Reading: Aho2, Section 4.7.1 to 4.7.3

Next Lecture: Look-Ahead LR (LALR) Parser