3.1.2.2 CONSTRUCTING CANONICAL LR(CLR) PARSING TABLES

- In SLR method, the state i makes a reduction by $A \rightarrow \alpha$ when the current token is **a**:
  - if the $A \rightarrow \alpha.$ is in the I_i and **a** is FOLLOW(A)
  - In some situations, when state i appears on stack top, the viable prefix $\beta\alpha$ on the stack is such that βA cannot be followed by a in right sentential form.
- Thus the reduction by $A \rightarrow \alpha$ would be invalid on input a.
- Because of that we go for : Canonical LR Parser
  - In this, it is possible to carry more information in the state that will allow us to avoid some of these invalid reductions
- To avoid some of invalid reductions, the states need to carry more information.
- Extra information is put into a state by including a terminal symbol as a second component in an item.
- A LR(1) item is:

  $A \rightarrow \alpha.\beta,a$ where **a** is the look-head of the LR(1) item

  (**a** is a terminal or end-marker.)

- Such an object is called LR(1) item.
  - 1 refers to the length of the second component
  - The lookahead has no effect in an item of the form $[A \rightarrow \alpha.\beta,a]$, where β is not $\in$.
  - But an item of the form $[A \rightarrow \alpha.,a]$ calls for a reduction by $A \rightarrow \alpha$ only if the next input symbol is a.
- When β (in the LR(1) item $A \rightarrow \alpha.\beta,a$) is not empty, the look-head does not have any affect.
- When β is empty ($A \rightarrow \alpha.,a$), we do the reduction by $A \rightarrow \alpha$ only if the next input symbol is **a** (not for any terminal in FOLLOW(A)).

Canonical Collection of Sets of LR(1) Items

- The construction of the canonical collection of the sets of LR(1) items are similar to the construction of the canonical collection of the sets of LR(0) items, except that closure and goto operations work a little bit different.

goto operation

- If I is a set of LR(1) items and X is a grammar symbol (terminal or non-terminal), then goto(I,X) is defined as follows:
  - If $A \rightarrow \alpha.X\beta,a$ in I then every item in **closure({$A \rightarrow \alpha X.\beta,a$})** will be in goto(I,X).

closure(I) is: (where I is a set of LR(1) items)

- every LR(1) item in I is in closure(I)
- if $A \rightarrow \alpha.B\beta,a$ in closure(I) and $B \rightarrow \gamma$ is a production rule of G; then $B \rightarrow .\gamma,b$ will be in the closure(I) for each terminal b in FIRST(βa) .

Construction of CLR (canonical LR) Parsing Tables

Algorithm: Construction of canonical- LR parsing tables.

INPUT: An augmented grammar G′.

OUTPUT: the canonical-LR parsing table functions ACTION AND GOTO for G′.

METHOD:

1. **Construct C′ = { I_0, I_1, ……., I_n}, the collection of sets of LR(1) items for G′.**
2. **State i of the parser is constructed from I_i. The parsing action for state i is determined as follows.**

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

   (b) If [$A \rightarrow \alpha.$, a] is in I_i, $A \neq S'$, then set ACTION[i, a] to "reduce $A \rightarrow \alpha$."

   (c) If [$S' \rightarrow S.$, $] is in I_i, then set ACTION[i, $] to "accept".

   If any conflicting actions result from the above rules, we say the grammar is not LR(1). The algorithm falls to produce a parser in this case.
3. **The goto transitions for state i are constructed for all non terminals A using the rule: if GOTO(I_i, A) = I_j, then GOTO[i, A]= j.**
4. **All entries not defined by rules (2) and (3) are made "error".**
5. **The initial state of the parser is the one constructed from the set of items containing [$S' \rightarrow .S$, $].**

EXAMPLE

Construct CLR parsing table for the flowing grammar

S→ CC

C→ cC | d

AUGMENTED GRAMMAR

S′→ S

S→ CC

C→ cC

C→ d

CANONICAL COLLECTION OF LR(1) ITEMS

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, $

I_3 = GOTO (I_0, c)
C → c•C, c | d
C → •c C , 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 → •c C ,$
C → •d , $

I_7 = GOTO (I_2, d)
C → d• , $

I_8 = GOTO (I_3, C)
C → c C•, c | d

I_3 = GOTO (I_3, c)
C → c• C , c | d
C→ •cC, c | d
C→ •d, c | d

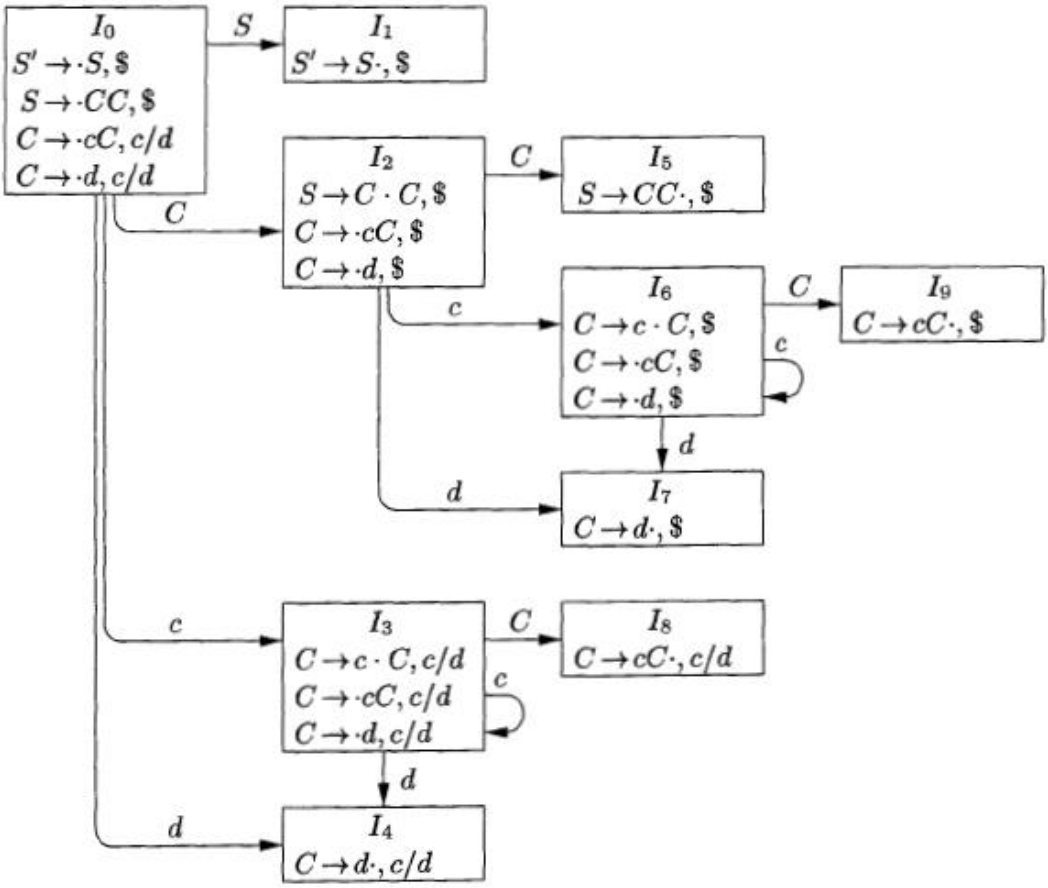
I_4 = GOTO (I_3, d)
C→ d •, c | d

I_9 = GOTO (I_6, C)
C → c C• , $

I_6 = GOTO (I_6, c)
C → c• C ,$
C→ •cC, $
C→ •d, $

I_7 = GOTO (I_6, d)
C→ d•, $

GOTO GRAPH FOR THE GRAMMAR



NUMBER THE PRODUCTIONS

1. S→ CC
2. C→ cC
3. C→ d

PARSING TABLE

I_1 = S′ → S• , $

I_4 = C → d• , c | d

I_5 = S → CC•, $

I_7 = C → d• , $

I_8 = C → c C•, c | d

I_9 = C → c C• , $

STATE	ACTION			GOTO	
	c	*d*	$	*S*	*C*
0	s3	s4		1	2
1			acc		
2	s6	s7			5
3	s3	s4			8
4	r3	r3			
5			r1		
6	s6	s7			9
7			r3		
8	r2	r2			
9			r2		