CS-4031
Compiler Construction
Lecture 6

Mahzaib Younas

Lecturer, Department of Computer Science

FAST NUCES CFD

Bottom Up Parser

Bottom-up parsing is a type of syntax analysis method where the parser from input symbols(tokens) and attempts to reduce them to start symbol of grammar (S).

- Basic Components
  - Start with tokens
  - Shift and reduce

Components

- Start with tokens
  - The Parser begins with the terminal symbols (the input tokens), which are the leaves of the parse tree.
- Shift
  - The next token pushed onto a stack.
- Reduce
  - A sequence of symbols on the stack is replaced by a non-terminal according to the production rules of grammar.

Example

$E \rightarrow T$

$E \rightarrow T * F$

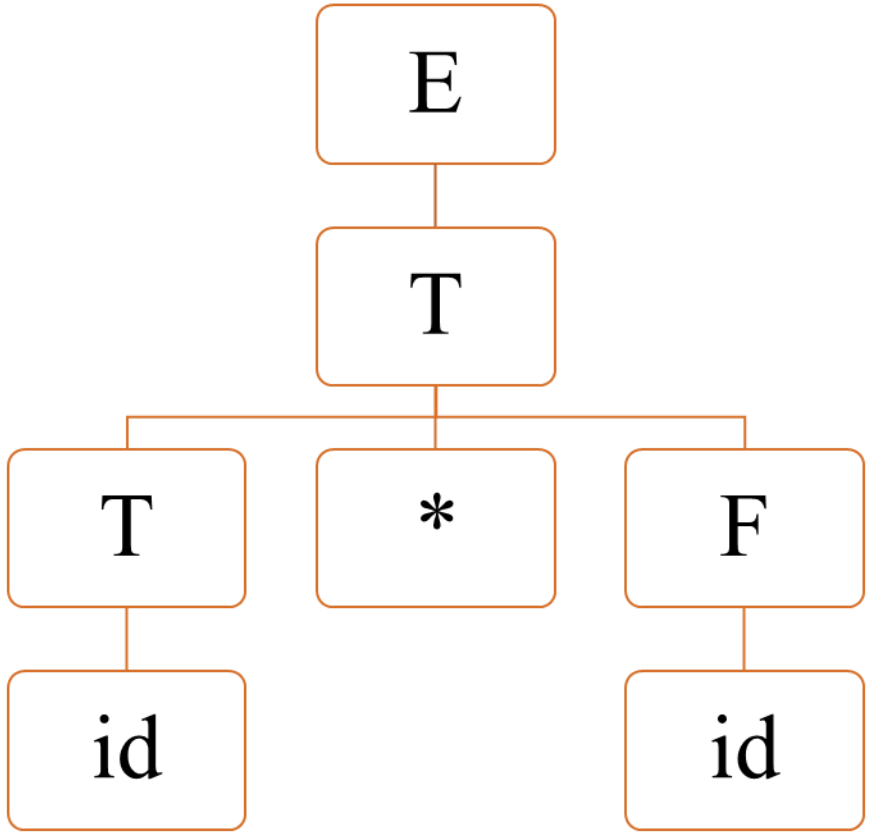
$T \rightarrow id$

$F \rightarrow T$

$F \rightarrow id$

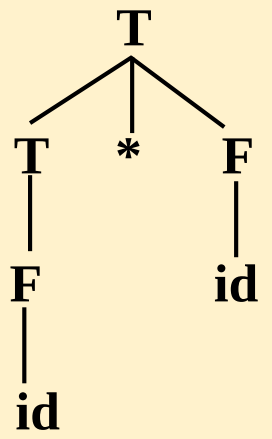
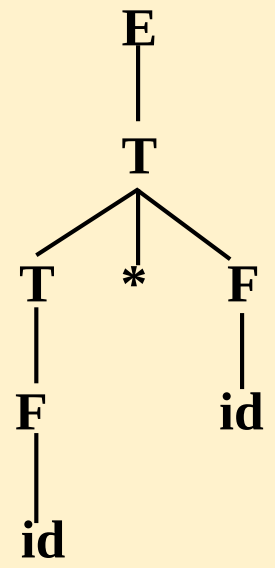
Input string = id * id

Input id * id

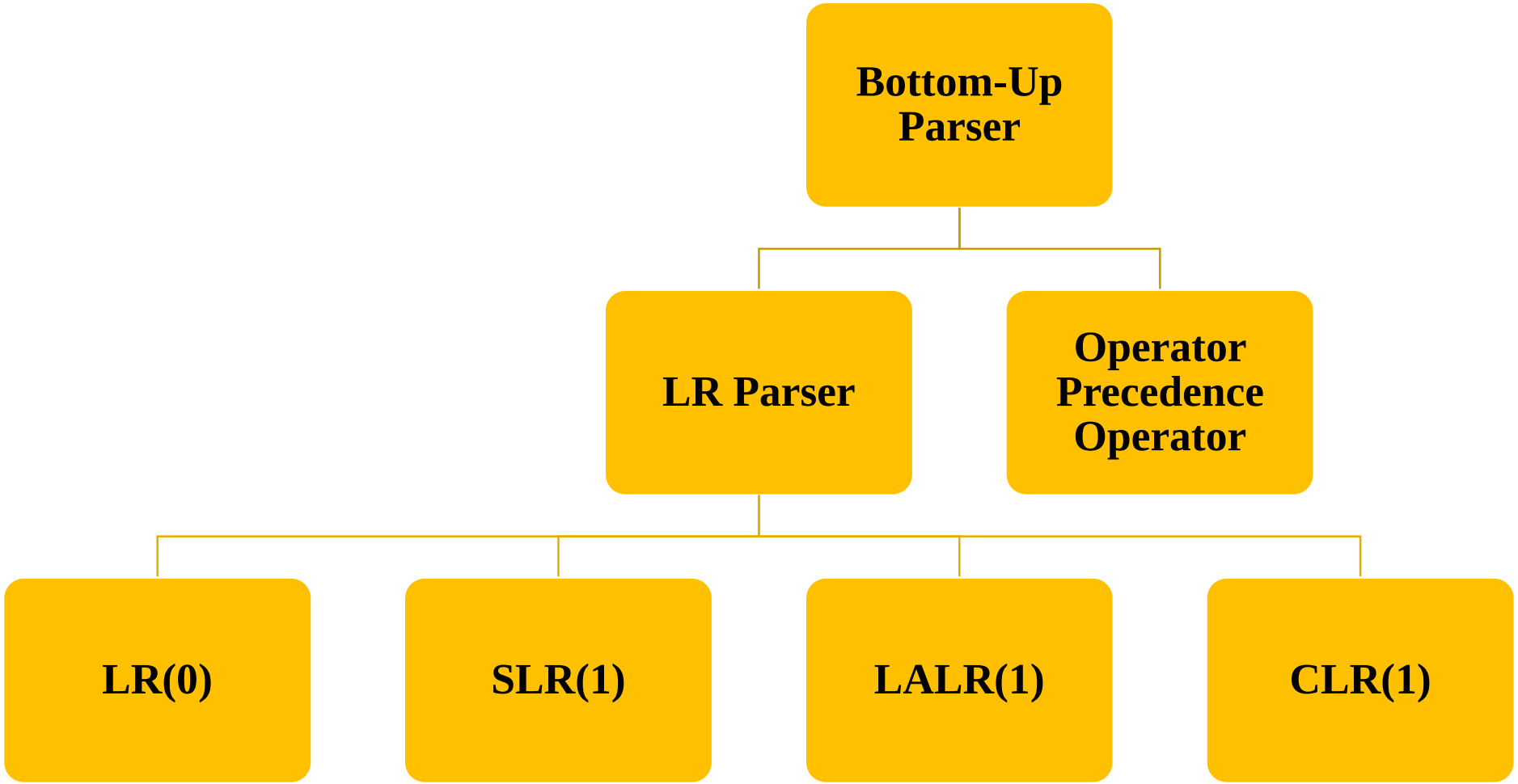


Input id * id

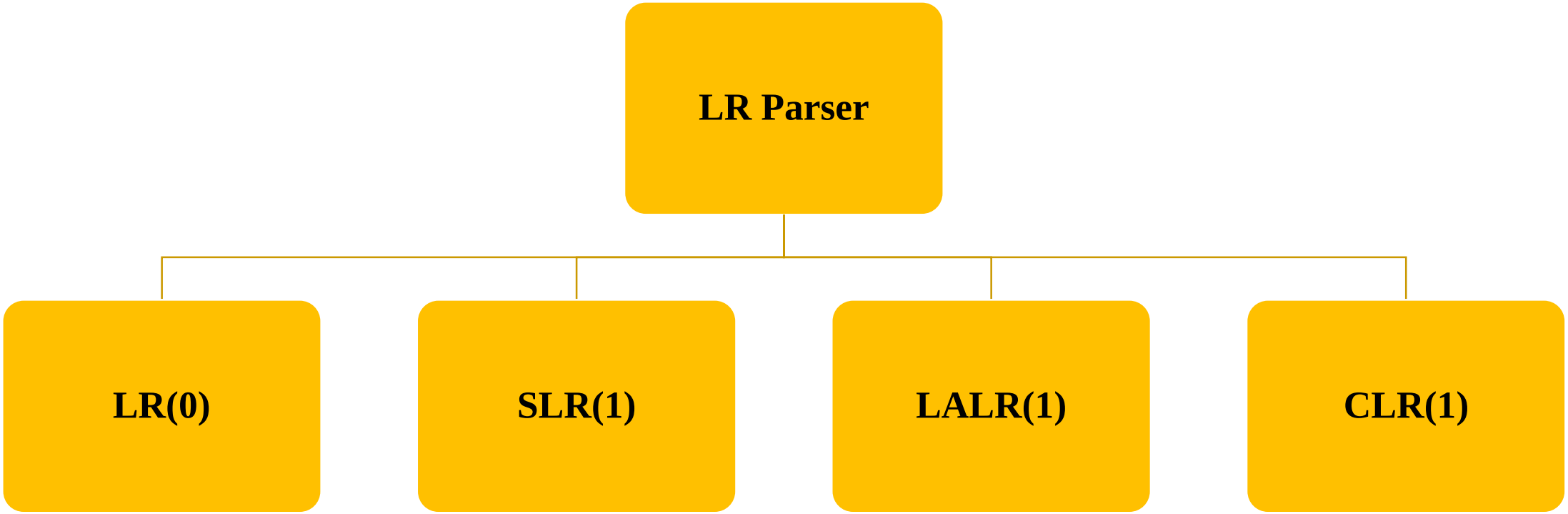
id * id

Id * id	**F * id** **id**	**T * id** **F** **id**	**T * F** **F id** **id**		

Bottoms up Parser



LR Parser



LR Parser

- LR parsing is a type of bottom-up parsing technique used to handle large and complex grammars.
- LR Means
  - L stands for left-to-right scanning of the input. It means the parser reads the input string one symbol at a time from left to right.
  - R stands for right most derivation in reverse.

LR(0) Parser

Augmented Grammar

- If P is a grammar with started symbol S, then G` augmented is new augmented grammar with the starting symbol S` and new production will be S` → .S.

LR(0) Parser

- In this we used LR(0) canonical item
- In LR(0) grammar, we do the following steps

1. Augment the given grammar

LR(0) Parser : Augmented Grammar

- To augment the grammar we used “.” symbol, which indicate that the value at the right side of . is not explore till now.

$$E \rightarrow T + E$$

- The augmented grammar will be

$$E` \rightarrow .E$$

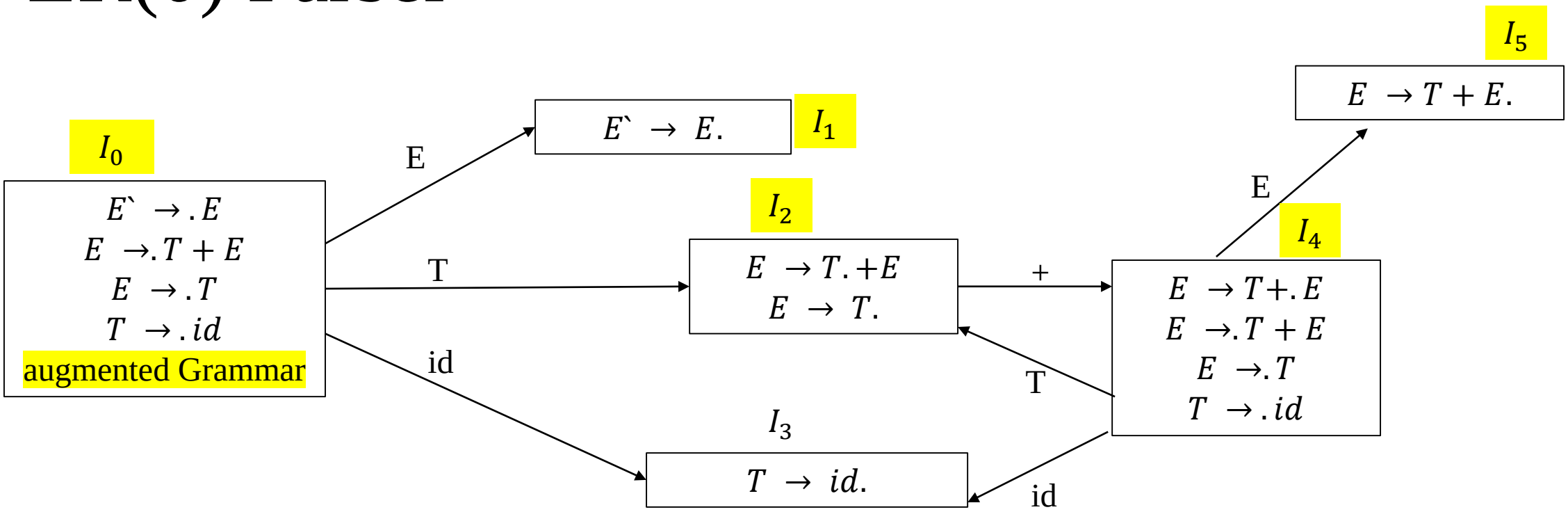
LR(0) Parser

- $E \rightarrow T + E \mid \mathrm{T}$
- $T \rightarrow id$

Augmented Grammar

- $E` \rightarrow .E$
- $E \rightarrow .T + E$
- $E \rightarrow .T$
- $T \rightarrow .id$

LR(0) Parser


I_0
$E` \rightarrow .E$
$E \rightarrow .T + E$
$E \rightarrow .T$
$T \rightarrow .id$
augmented Grammar
E
$E` \rightarrow E.$
I_1
T
I_2
$E \rightarrow T. + E$
$E \rightarrow T.$
id
I_3
$T \rightarrow id.$
+
I_4
$E \rightarrow T + .E$
$E \rightarrow .T + E$
$E \rightarrow .T$
$T \rightarrow .id$
T
id
E
I_5
$E \rightarrow T + E.$

LR(0) Parsing Table

	Action			GOTO	
	id	**+**	**$**	**E**	**T**
0	S_3			1	2
1					
2	r_2	S_4/r_2	r_2		
3	r_3	r_3	r_3		
4	S_3			5	2
5	r_1	r_1	r_1		

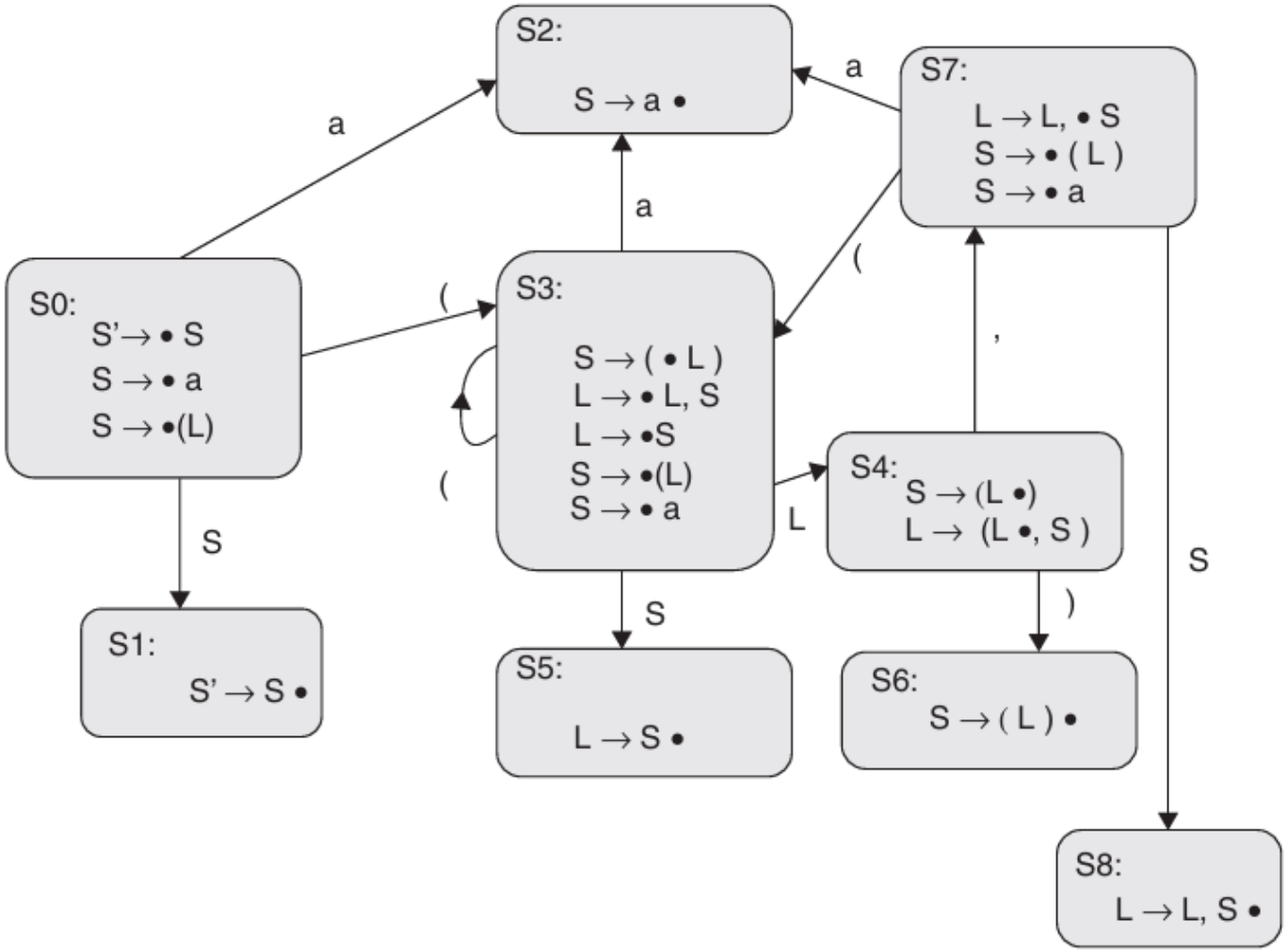
LR(0) Parser: Example 2

- $S \rightarrow (L)$
- $S \rightarrow a$
- $L \rightarrow S$
- $L \rightarrow L, S$

Augmented Grammar

- S” → .S
- S → .(L)
- S → .a
- L → .S
- L → .L, S

LR(0) Parsing Table



LR(0) Parsing Table

States	Action					Goto	
	(	)	a	,	$	S	L
0	s_3		s_2			1	
1	Accepting state						
2	r_2	r_2	r_2	r_2	r_2		
3	s_3		s_2			5	4
4		s_6		s_7			
5	r_3	r_3	r_3	r_3	r_3		
6	r_1	r_1	r_1	r_1	r_1		
7	s_3		s_2			8	
8	r_4	r_4	r_4	r_4	r_4		

LR(0) Parser :

- $E \rightarrow BB$
- $B \rightarrow cB \mid d$

Augmented Grammar

- $E` \rightarrow .E$
- $E \rightarrow .BB$
- $B \rightarrow .cB$
- $B \rightarrow .d$

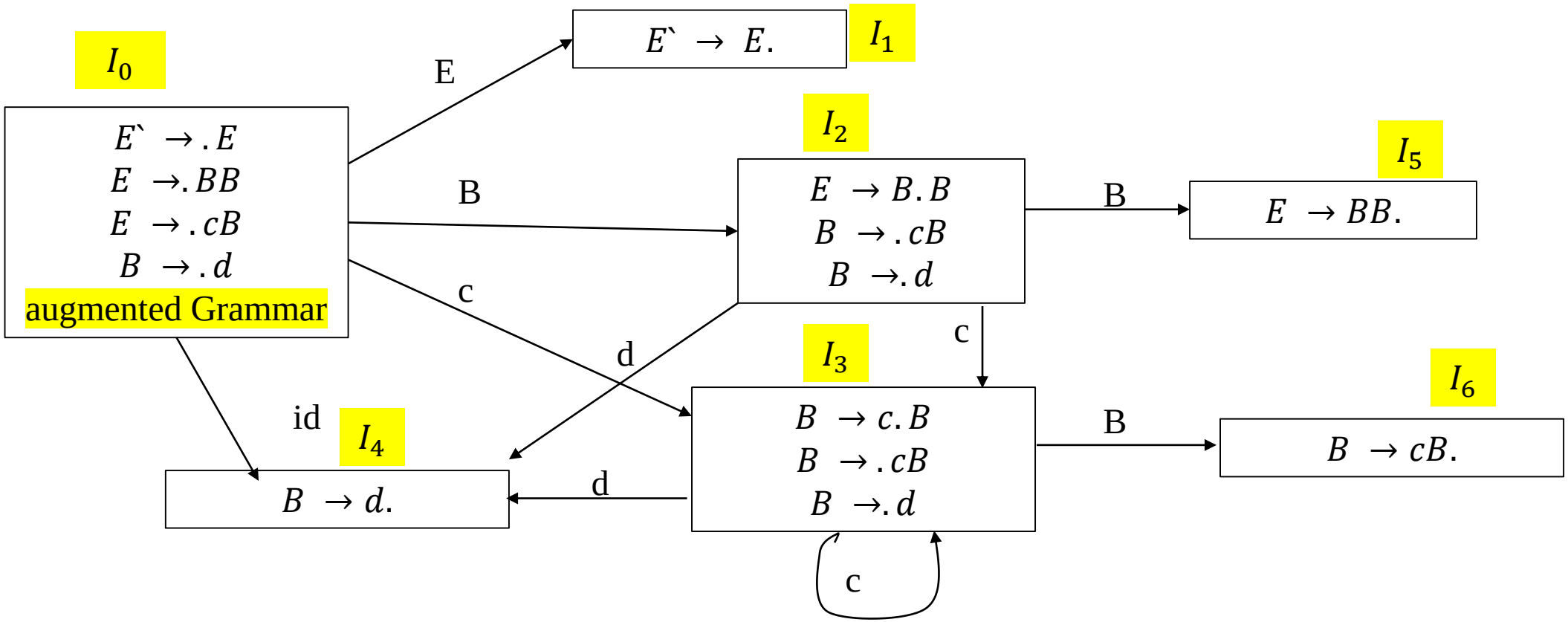
String : ccdd$

- $E \rightarrow BB$
- $B \rightarrow cB \mid d$

Augmented Grammar

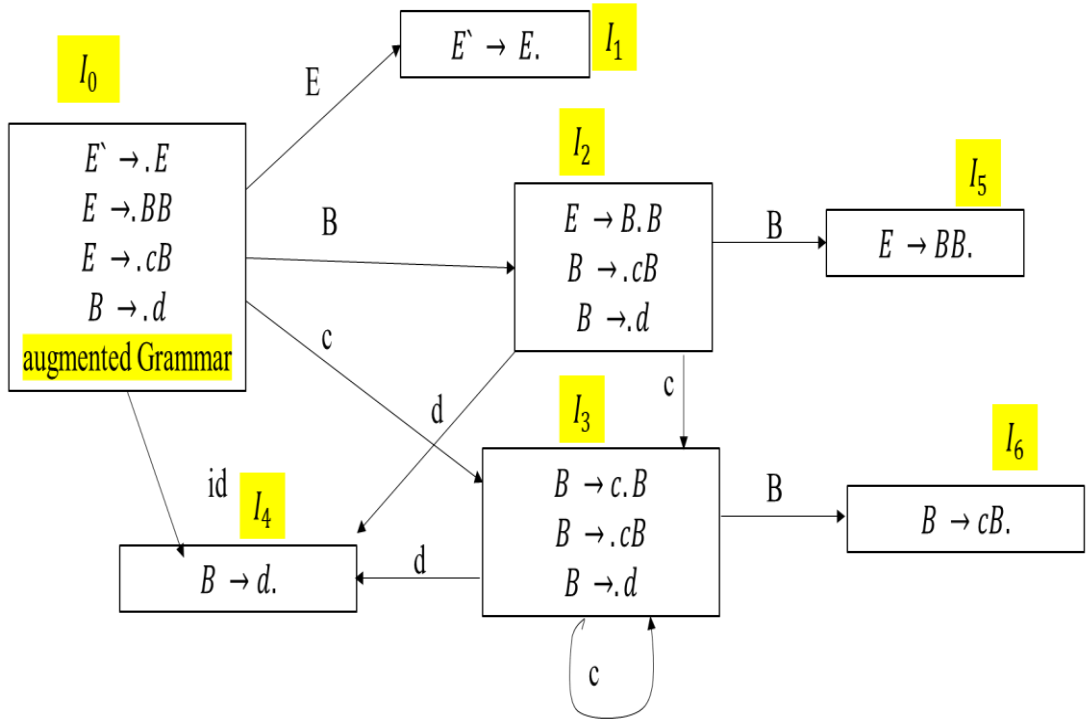
- $E` \rightarrow .E$
- $E \rightarrow .BB$
- $B \rightarrow .cB$
- $B \rightarrow .d$

String : Diagram$



String Traversal : ccdd$

	Action			GOTO	
	c	**d**	**$**	**B**	**E**
0	S_3	S_4		2	1
1					
2	S_3	S_4		5	
3	S_3	S_4		6	
4	r_3	r_3	r_3		
5	r_1	r_1	r_1		
6	r_2	r_2	r_2		



String Traversal : ccdd$

stack	input	Action
$0	ccdd$	Shift c → s3
$0c3	cdd$	Shift c → s3
$0c3c3	dd$	Shift c → s4
$0c3c3c3	d$	Reduce 3 $B \rightarrow d$
$0c3c3B6	d$	Reduce 2 $B \rightarrow .cB$
$0c3B6	d$	Reduce 2 $B \rightarrow .cB$
$0B2	d$	Shift d → s4
$0B2d4	$	Reduce r3 $B \rightarrow d$
$0B2B5	$	Reduce r1 $E \rightarrow .BB$
$0E1	$	Accept

$$E` \rightarrow .E$$
$$E \rightarrow .BB$$
$$B \rightarrow .cB$$
$$B \rightarrow .d$$

	Action			GOTO	
	c	d	$	B	E
0	S_3	S_4		2	1
1					
2	S_3	S_4		5	
3	S_3	S_4		6	
4	r_3	r_3	r_3		
5	r_1	r_1	r_1		
6	r_2	r_2	r_2		

Advantages of LR(0) Parser

- It is very simple to construct the LR(0) Parser compared to other LR parsers.
- Each row in the table defines unique action, that is, either shift action or reduce action or accept.

Disadvantages of LR(0) Parser

- The LR(0) Parser can be used to parse a small class of grammars.
- Look ahead is not used in making parsing decisions

Canonical LR(1) Parsers CLR(1)/LR(1)

- The CLR parser stands for canonical LR parser. It is a more powerful LR parser.
- It makes use of lookahead symbols.
- This method uses a large set of items called LR(1) items.
- The main difference between LR(0) and LR(1) items is that, in LR(1) items, it is possible to carry more information in a state, which will rule out useless reduction states.
- This extra information is incorporated into the state by the lookahead symbol

CLR (1) Parser

- Consider the following grammar

$S \rightarrow CC$

$C \rightarrow cC$

$C \rightarrow d$

CLR (1) Parser

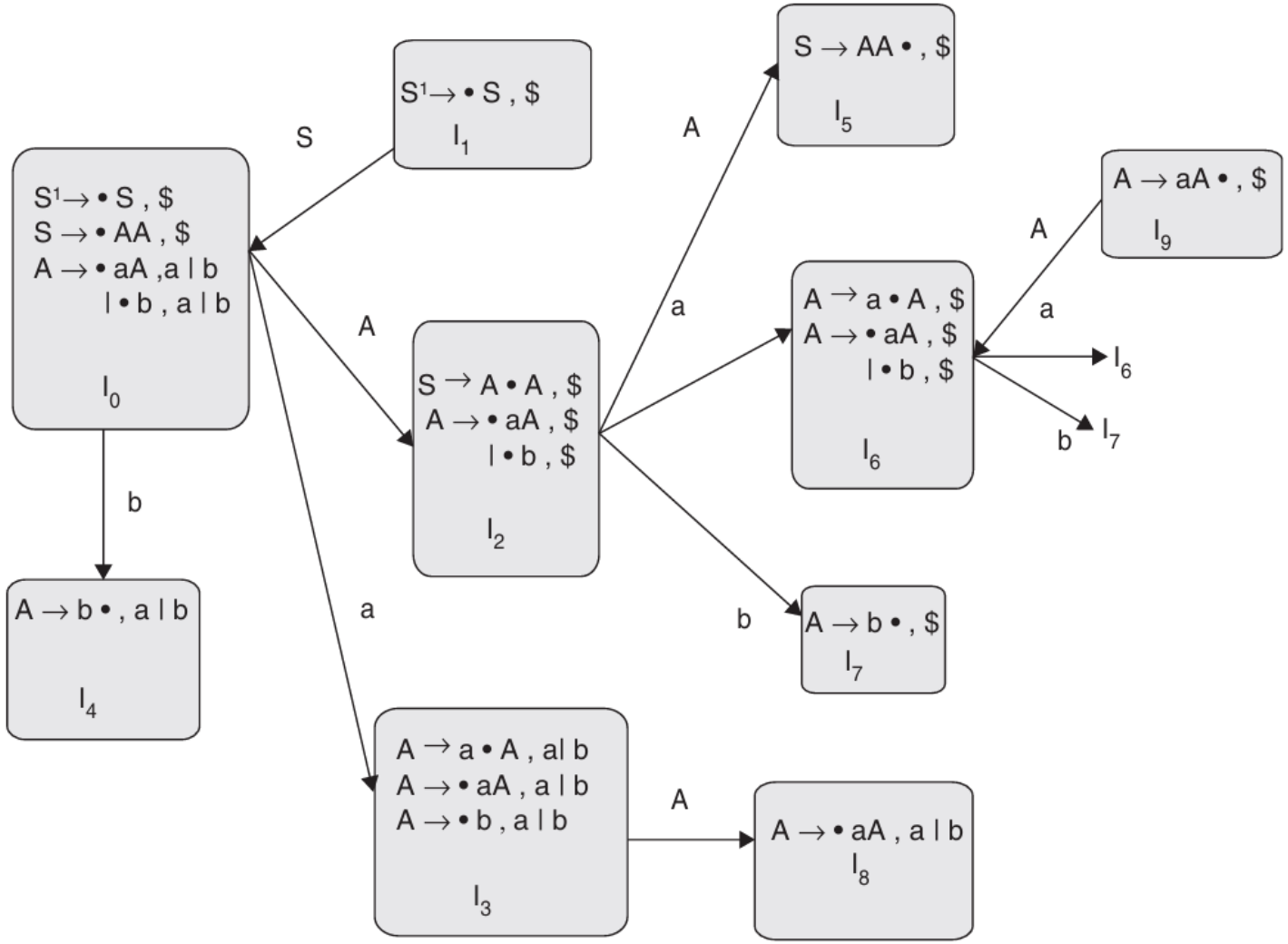
- Step 1: generate the augmented grammar

S’ → .S

S → .AA

A → .aA |.b

CLR (1) Parser


S¹→ • S , $
S → • AA , $
A → • aA ,a | b
| • b , a | b
I0
S¹→ • S , $
I1
S → AA • , $
I5
A → aA • , $
I9
A → a • A , $
A → • aA , $
| • b , $
I6
S → A • A , $
A → • aA , $
| • b , $
I2
A → b • , a | b
I4
A → b • , $
I7
A → a • A , a| b
A → • aA , a | b
A → • b , a | b
I3
A → • aA , a | b
I8
S
A
a
b
A
a
b
A
a
b
A

CLR (1) Parsing Table

States	Action			Goto	
	a	b	$	S	A
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		

CLR(1) Grammar Example:

- Given Grammar
- $S \rightarrow AA$
- $A \rightarrow aA$
- $A \rightarrow b$

CLR(1) Grammar Example:

- S → AA
- A → aA
- A → b

Augmented Grammar

- S` → •S, $
- S → •AA, $
- A → •aA, a/b
- A → •b, a/b