CS 4300: Compiler Theory

Chapter 4
Syntax Analysis

Dr. Xuejun Liang

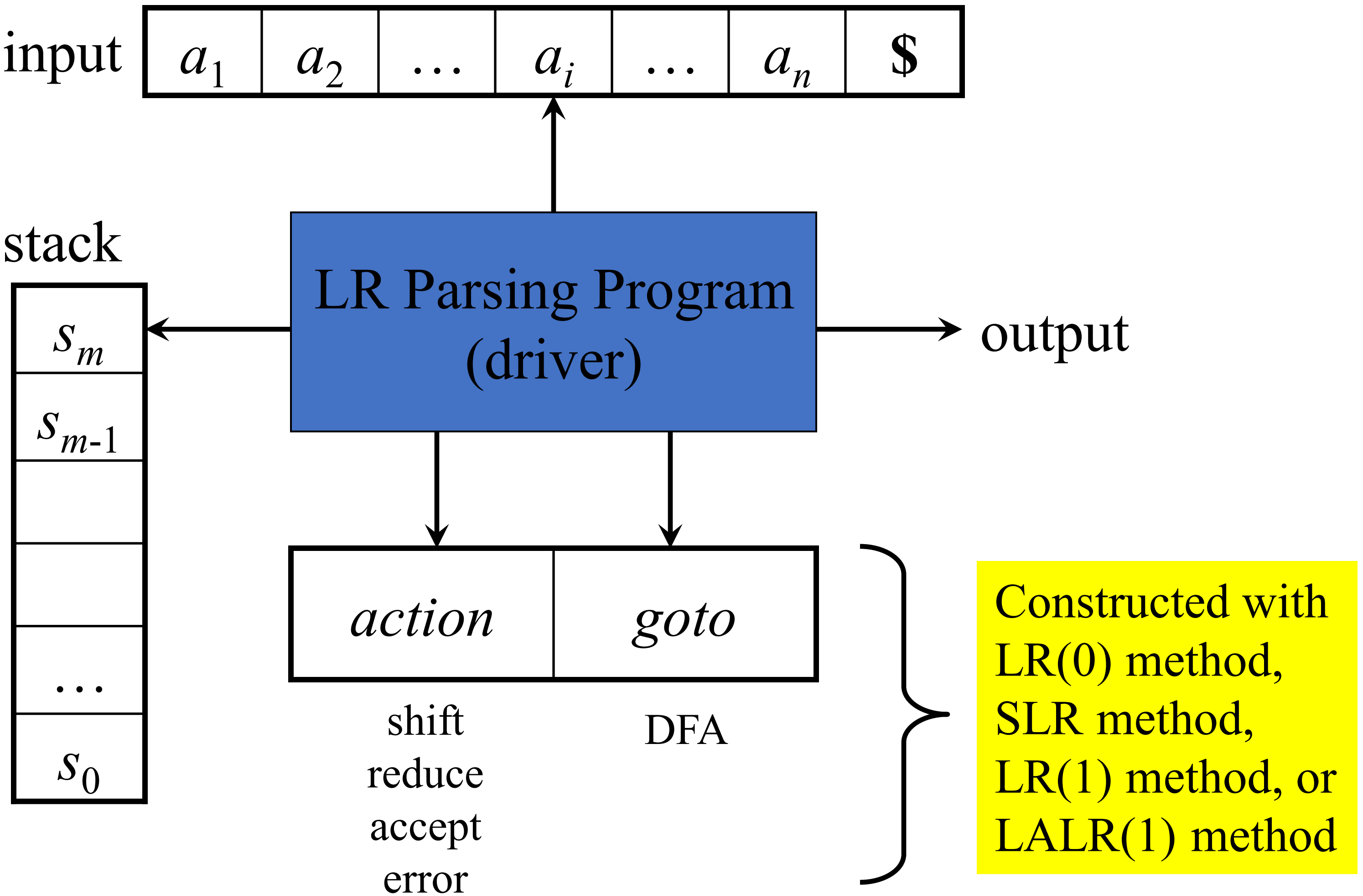
Outlines (Sections)

1. Introduction
2. Context-Free Grammars
3. Writing a Grammar
4. Top-Down Parsing
5. Bottom-Up Parsing
6. Introduction to LR Parsing: Simple LR
7. More Powerful LR Parsers
8. Using Ambiguous Grammars
9. Parser Generators

Quick Review of Last Lecture

- Bottom-Up Parsing
  - Stack Implementation of Shift-Reduce Parsing
  - Shift-reduce and reduce-reduce conflicts
- LR Parsing
  - LR(0) Items of a Grammar
  - The closure Operation for LR(0) Items
  - The goto Operation for LR(0) Items
  - Construct LR(0) Automaton of a Grammar
  - Use of the LR(0) Automaton
  - Examples

Model of an LR Parser



LR Parsing (Driver)

$X_1\, X_2\, \ldots\, X_m\, a_i\, a_{i+1}\, \ldots\, a_n$ ← right-sentential form

Configuration (= LR parser state):

$$(\underbrace{s_0\, s_1\, s_2\, \ldots\, s_m}_{\textit{stack}},\ \underbrace{a_i\, a_{i+1}\, \ldots\, a_n\, \$}_{\textit{input}})$$

If $action[s_m,a_i]$ = shift s **then** push s, and advance input:

$$(s_0\, s_1 s_2\, \ldots\, s_m\, s,\ \ a_{i+1}\, \ldots\, a_n\, \$)$$

If $action[s_m,a_i]$ = reduce $A \rightarrow \beta$ and $goto[s_{m-r},A] = s$ with $r=|\beta|$ **then** pop r symbols, and push s:

$$(s_0\, s_1\, s_2\, \ldots\, s_{m-r}\, s,\ \ a_i\, a_{i+1}\, \ldots\, a_n\, \$)$$

If $action[s_m,a_i]$ = accept **then** stop

If $action[s_m,a_i]$ = error **then** attempt recovery

Example LR(0) Parsing Table

State I_0:
$C' \to \bullet C$
$C \to \bullet A\ B$
$A \to \bullet \mathbf{a}$

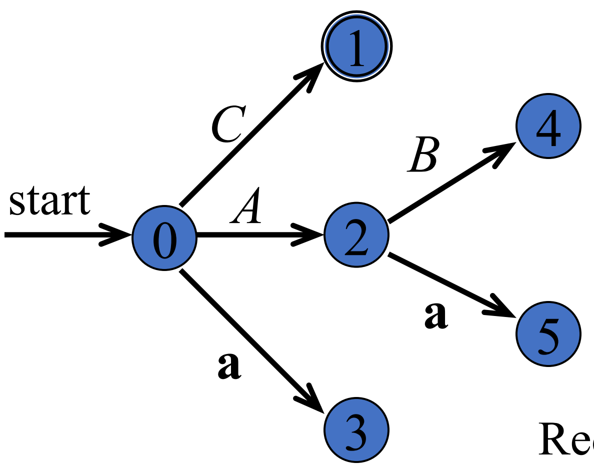
State I_1:
$C' \to C\bullet$

State I_2:
$C \to A \bullet B$
$B \to \bullet \mathbf{a}$

State I_3:
$A \to \mathbf{a}\bullet$

State I_4:
$C \to A\ B\bullet$

State I_5:
$B \to \mathbf{a}\bullet$



Shift & goto 3

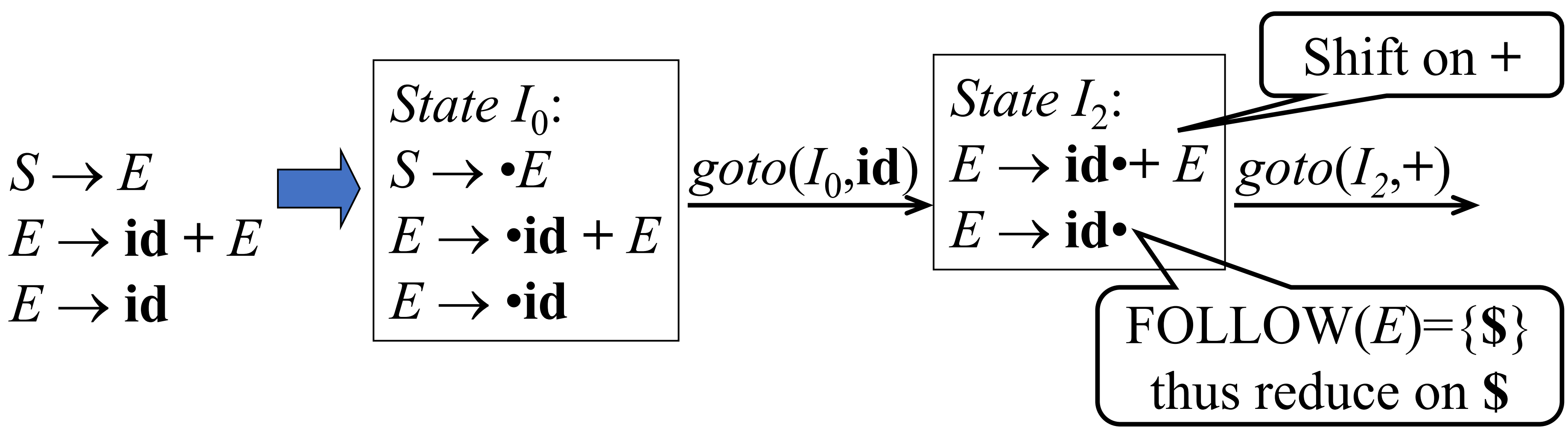
	action		*goto*		
state	**a**	**$**	*C*	*A*	*B*
0	s3		1	2	
1		acc			
2	s5				4
3	r3	r3			
4	r2	r2			
5	r4	r4			

Reduce by production #2

Grammar:
1. $C' \to C$
2. $C \to A\ B$
3. $A \to \mathbf{a}$
4. $B \to \mathbf{a}$

SLR Grammars

- SLR (Simple LR): SLR is a simple extension of LR(0) shift-reduce parsing
- SLR eliminates some conflicts by populating the parsing table with reductions $A \rightarrow \alpha$ on symbols in FOLLOW(A)



SLR Parsing Table

- Reductions do not fill entire rows
- Otherwise the same as LR(0)

1. $S \to E$
2. $E \to \textbf{id} + E$
3. $E \to \textbf{id}$

	id	+	**$**	*E*
0	s2			1
1			acc	
2		s3	r3	
3	s2			4
4			r2	

Shift on +

FOLLOW(E)={**$**}
thus reduce on **$**

State I_0:
$S \to \bullet E$
$E \to \bullet\textbf{id} + E$
$E \to \bullet\textbf{id}$

State I_2:
$E \to \textbf{id}\bullet + E$
$E \to \textbf{id}\bullet$

State I_1:
$S \to E\ \bullet$

State I_3:
$E \to \textbf{id} + \bullet\ E$

State I_4:
$E \to \textbf{id} + E\ \bullet$

SLR Parsing

- An LR(0) state is a set of LR(0) items
- An LR(0) item is a production with a • (dot) in the right-hand side
- Build the LR(0) DFA by
  - *Closure operation* to construct LR(0) items
  - *Goto operation* to determine transitions
- Construct the SLR parsing table from the DFA
- LR parser program uses the SLR parsing table to determine shift/reduce operations

Constructing SLR Parsing Tables

1. Augment the grammar with $S' \rightarrow S$
2. Construct $C=\{I_0, I_1, \ldots, I_n\}$, the collection of sets of *LR*(0) *items.* State *i* is constructed from I_i.
3. If $[A \rightarrow \alpha \bullet a \beta] \in I_i$ and ***goto***$(I_i, a)=I_j$ then set ***action***$[i, a]$=shift *j,* where *a* is a terminal
4. If $[A \rightarrow \alpha \bullet] \in I_i$ then set ***action***$[i, a]$=reduce $A \rightarrow \alpha$ for all $a \in$ FOLLOW(A) (apply only if $A \neq S'$)
5. If $[S' \rightarrow S \bullet]$ is in I_i then set ***action***$[i,$ **$**$]$=accept
6. If ***goto***$(I_i, A)=I_j$ then set ***goto***$[i,A]=j$
7. Repeat 3-6 until no more entries added
8. The initial state *i* is the I_i holding item $[S' \rightarrow \bullet S]$

Example Grammar and LR(0) Items

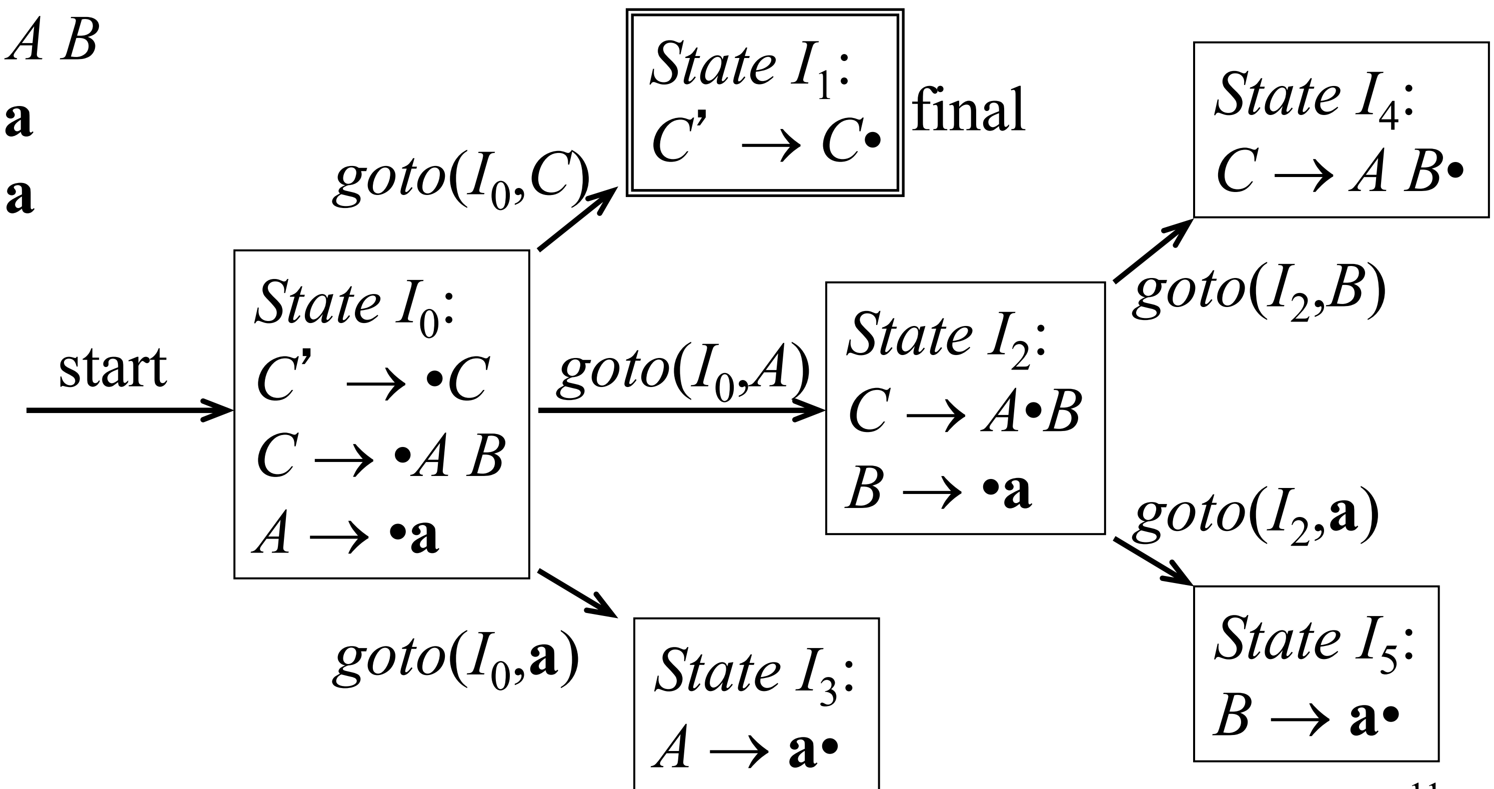
Augmented grammar:

1. $C' \to C$
2. $C \to A\ B$
3. $A \to \mathbf{a}$
4. $B \to \mathbf{a}$

$I_0 = closure(\{[C' \to \bullet C]\})$

$I_1 = \text{goto}(I_0,C) = closure(\{[C' \to C\bullet]\})$

…



Example SLR Parsing Table

State I_0:
$C' \to \bullet C$
$C \to \bullet A\ B$
$A \to \bullet \mathbf{a}$

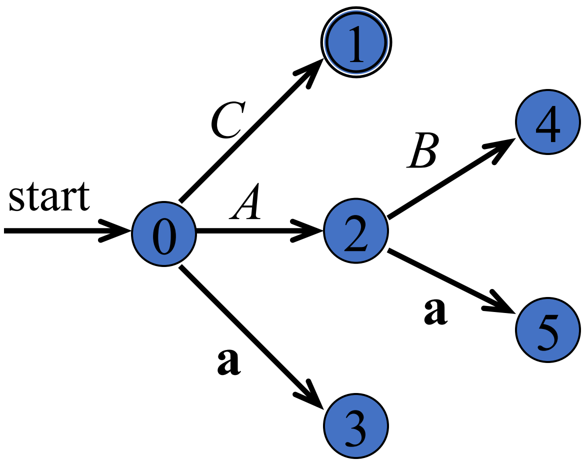
State I_1:
$C' \to C \bullet$

State I_2:
$C \to A \bullet B$
$B \to \bullet \mathbf{a}$

State I_3:
$A \to \mathbf{a} \bullet$

State I_4:
$C \to A\ B \bullet$

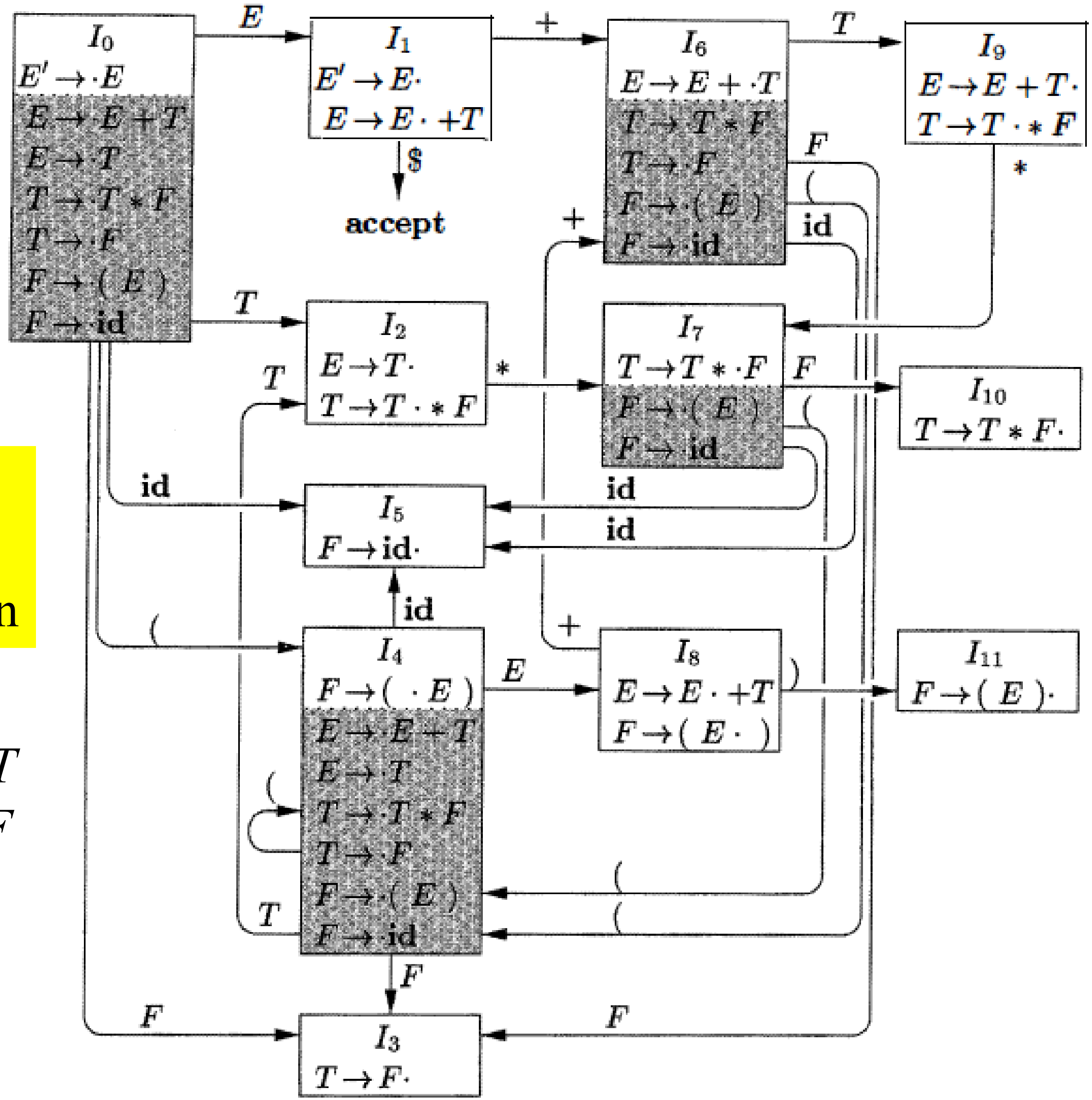
State I_5:
$B \to \mathbf{a} \bullet$



state	*action* **a**	*action* **$**	*goto* *C*	*goto* *A*	*goto* *B*
0	s3		1	2	
1		acc			
2	s5				4
3	r3				
4		r2			
5		r4			

FOLLOW(A) = {**a**}
FOLLOW(C) = {**$**}
FOLLOW(B) = {**$**}

Grammar:
1. $C' \to C$
2. $C \to A\ B$
3. $A \to \mathbf{a}$
4. $B \to \mathbf{a}$



LR(0) Automaton for expression

Grammar:

$E \rightarrow E + T \mid T$

$T \rightarrow T * F \mid F$

$F \rightarrow (E)$

$F \rightarrow \textbf{id}$

SLR Parse Table for Expression Grammar

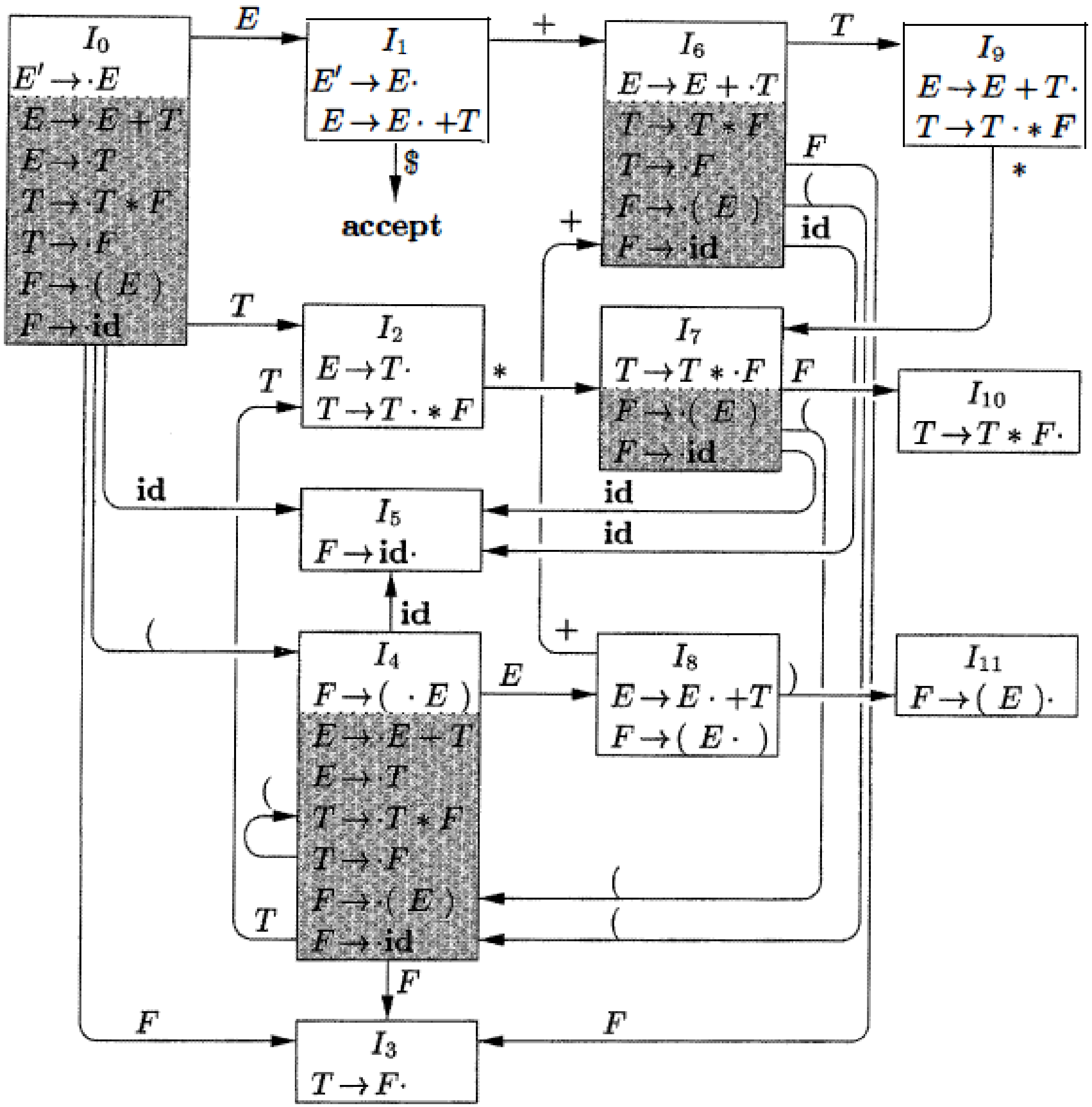
Grammar:

1. $E \rightarrow E + T$
2. $E \rightarrow T$
3. $T \rightarrow T * F$
4. $T \rightarrow F$
5. $F \rightarrow (E)$
6. $F \rightarrow$ **id**

	action						*goto*		
state	**id**	**+**	*****	**(**	**)**	**$**	*E*	*T*	*F*
0	s5			s4			1	2	3
1		s6				acc			
2		r2	s7		r2	r2			
3		r4	r4		r4	r4			
4	s5			s4			8	2	3
5		r6	r6		r6	r6			
6	s5			s4				9	3
7	s5			s4					10
8		s6			s11				
9		r1	s7		r1	r1			
10		r3	r3		r3	r3			
11		r5	r5		r5	r5			

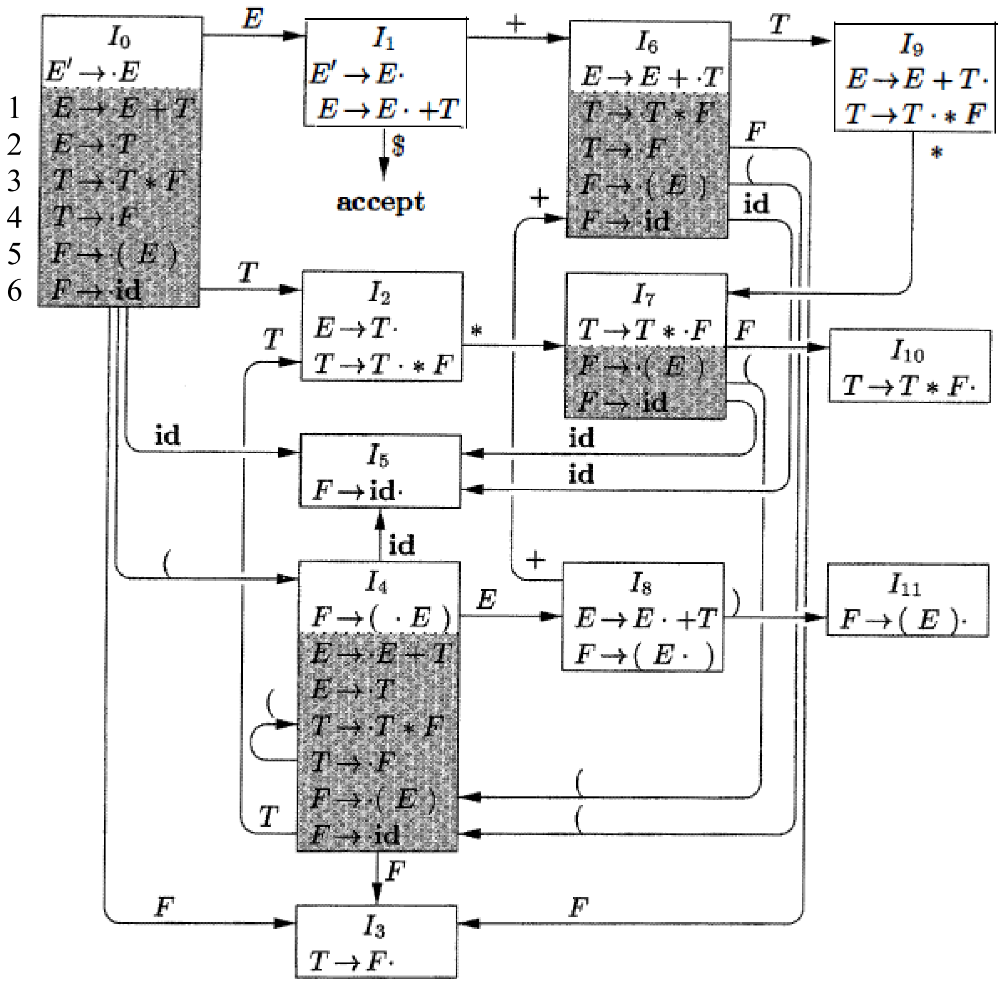
Shift & goto 5

Reduce by production #1



state	*goto* E	T	F
0	1	2	3
1			
2			
3			
4	8	2	3
5			
6		9	3
7			10
8			
9			
10			
11			

FOLLOW(E) = { +) $ }
FOLLOW(T) = { + *) $ }
FOLLOW(F) = { + *) $ }



action

state	**id**	+	*	(	)	$
0	s5			s4		
1		s6				acc
2		r2	s7		r2	r2
3		r4	r4		r4	r4
4	s5			s4		
5		r6	r6		r6	r6
6	s5			s4		
7	s5			s4		
8		s6			s11	
9		r1	s7		r1	r1
10		r3	r3		r3	r3
11		r5	r5		r5	r5

Moves of an SLR parser on id * id + id
Using the SLR Parse Table on Previous Slide

	STACK	SYMBOLS	INPUT	ACTION
(1)	0		$\mathbf{id} * \mathbf{id} + \mathbf{id}\,\$$	shift
(2)	0 5	$\mathbf{id}$	$* \mathbf{id} + \mathbf{id}\,\$$	reduce by $F \to \mathbf{id}$
(3)	0 3	F	$* \mathbf{id} + \mathbf{id}\,\$$	reduce by $T \to F$
(4)	0 2	T	$* \mathbf{id} + \mathbf{id}\,\$$	shift
(5)	0 2 7	$T *$	$\mathbf{id} + \mathbf{id}\,\$$	shift
(6)	0 2 7 5	$T * \mathbf{id}$	$+ \mathbf{id}\,\$$	reduce by $F \to \mathbf{id}$
(7)	0 2 7 10	$T * F$	$+ \mathbf{id}\,\$$	reduce by $T \to T * F$
(8)	0 2	T	$+ \mathbf{id}\,\$$	reduce by $E \to T$
(9)	0 1	E	$+ \mathbf{id}\,\$$	shift
(10)	0 1 6	$E +$	$\mathbf{id}\,\$$	shift
(11)	0 1 6 5	$E + \mathbf{id}$	$\$$	reduce by $F \to \mathbf{id}$
(12)	0 1 6 3	$E + F$	$\$$	reduce by $T \to F$
(13)	0 1 6 9	$E + T$	$\$$	reduce by $E \to E + T$
(14)	0 1	E	$\$$	accept

Moves of an SLR parser on id * id + id Using the SLR Parse Table on Previous Slide

	action						goto		
state	**id**	+	*	(	)	$	E	T	F
0	s5			s4			1	2	3
1		s6				acc			
2		r2	s7		r2	r2			
3		r4	r4		r4	r4			
4	s5			s4			8	2	3
5		r6	r6		r6	r6			
6	s5			s4				9	3
7	s5			s4					10
8		s6			s11				
9		r1	s7		r1	r1			
10		r3	r3		r3	r3			
11		r5	r5		r5	r5			

Grammar:

1. $E \to E + T$
2. $E \to T$
3. $T \to T * F$
4. $T \to F$
5. $F \to (E)$
6. $F \to \mathbf{id}$

	STACK	SYMBOLS	INPUT	ACTION
(1)	0		$\mathbf{id} * \mathbf{id} + \mathbf{id}\,\$$	shift
(2)	0 5	**id**	$* \mathbf{id} + \mathbf{id}\,\$$	reduce by $F \to \mathbf{id}$
(3)	0 3	F	$* \mathbf{id} + \mathbf{id}\,\$$	reduce by $T \to F$
(4)	0 2	T	$* \mathbf{id} + \mathbf{id}\,\$$	shift
(5)	0 2 7	$T *$	$\mathbf{id} + \mathbf{id}\,\$$	shift
(6)	0 2 7 5	$T * \mathbf{id}$	$+ \mathbf{id}\,\$$	reduce by $F \to \mathbf{id}$
(7)	0 2 7 10	$T * F$	$+ \mathbf{id}\,\$$	reduce by $T \to T * F$
(8)	0 2	T	$+ \mathbf{id}\,\$$	reduce by $E \to T$
(9)	0 1	E	$+ \mathbf{id}\,\$$	shift
(10)	0 1 6	$E +$	$\mathbf{id}\,\$$	shift
(11)	0 1 6 5	$E + \mathbf{id}$	$\$$	reduce by $F \to \mathbf{id}$
(12)	0 1 6 3	$E + F$	$\$$	reduce by $T \to F$
(13)	0 1 6 9	$E + T$	$\$$	reduce by $E \to E + T$
(14)	0 1	E	$\$$	accept