

CSE 443
Compilers

Dr. Carl Alphonse
alphonse@buffalo.edu
343 Davis Hall

LR(k)

- LR(k) parser
 - L \Rightarrow left-to-right scanning of input
 - R \Rightarrow rightmost derivation in reverse
 - k \Rightarrow number of lookahead symbols
 - k is typically 0 or 1
 - LR \Rightarrow LR(1)

Lookahead here refers to how many input symbols can be consulted during parsing

LR(0) automaton and SLR

- SLR \Rightarrow Simple LR
- LR(0) automaton is constructed from G'
- "Suppose that the string γ of grammar symbols takes the LR(0) automaton from the start state 0 to some state j . Then, shift on next input symbol a if state j has a transition on a . Otherwise, we choose to reduce; the items in state j will tell us which production to use." [p 247]

LR(0) automaton and SLR

- SLR \Rightarrow Simple LR
- LR(0) automaton is constructed from G'

Lookahead here refers to how many input symbols can be consulted during automaton construction (i.e. in the items)

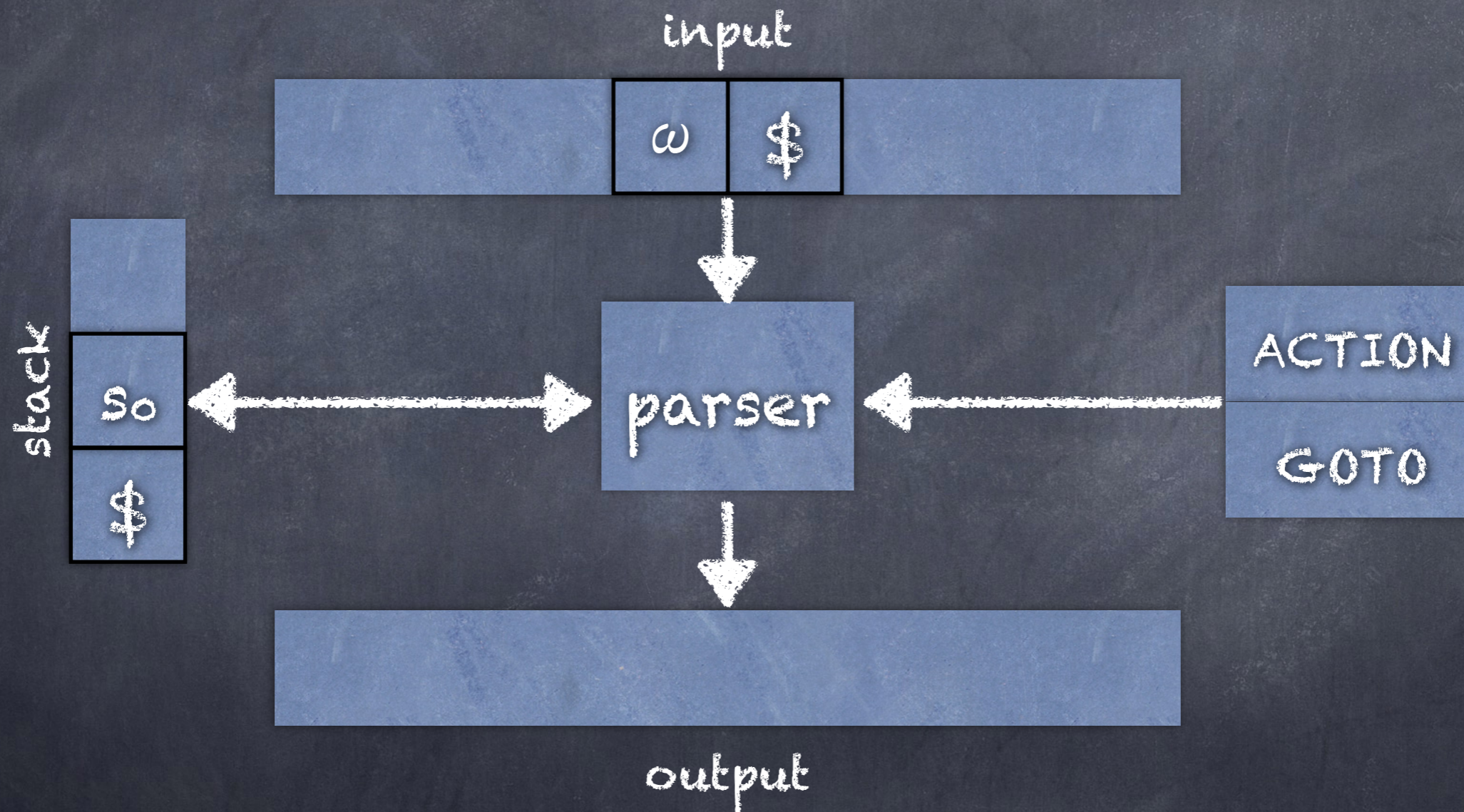
We'll see later how to incorporate lookahead in the items.

will tell us which production to use." [p 247]

Initial state of the parser

(top of stack is current state in LR(0) automata)

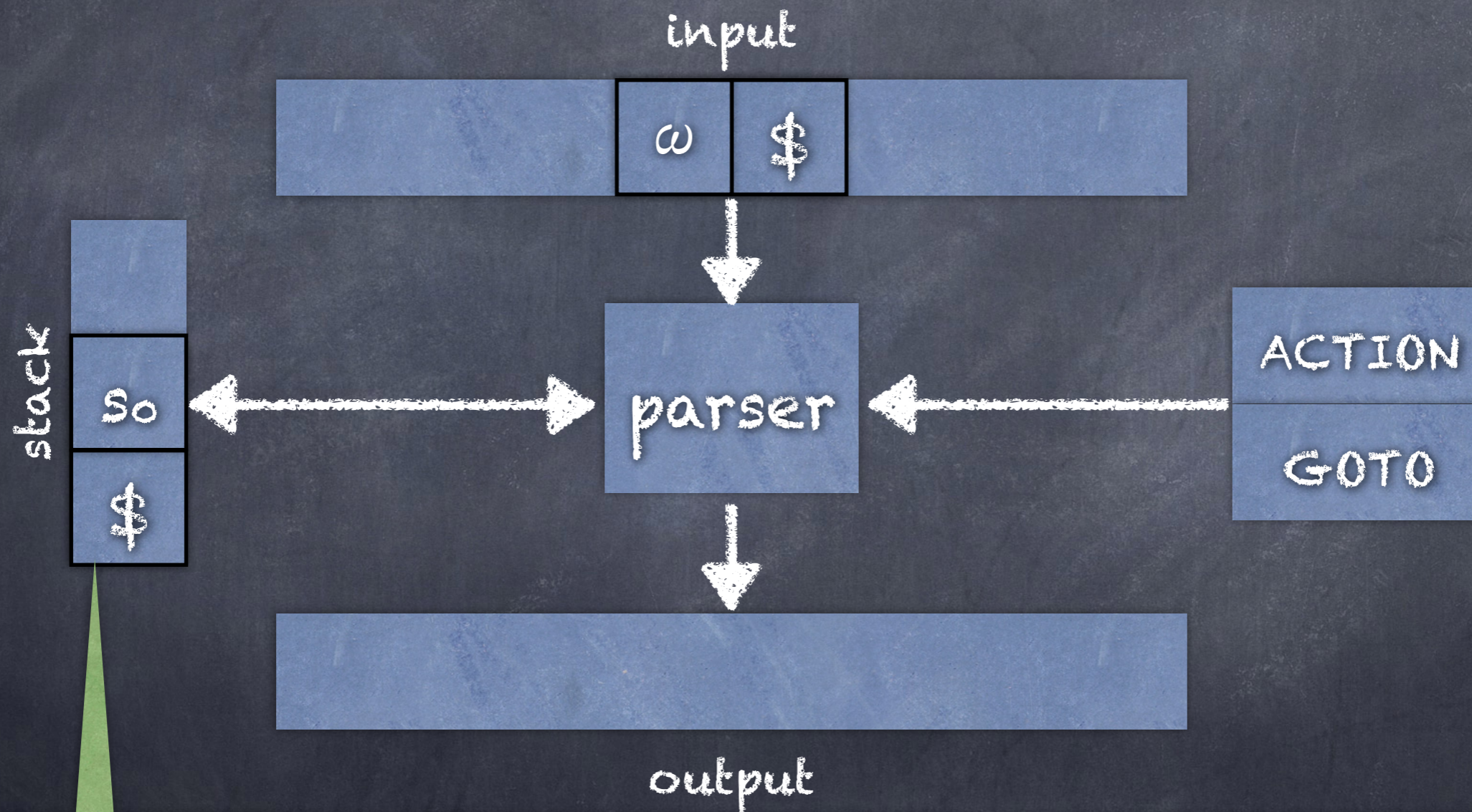
modified from figure 4.35 [p. 248]



Initial state of the parser

(top of stack is current state in LR(0) automata)

modified from figure 4.35 [p. 248]

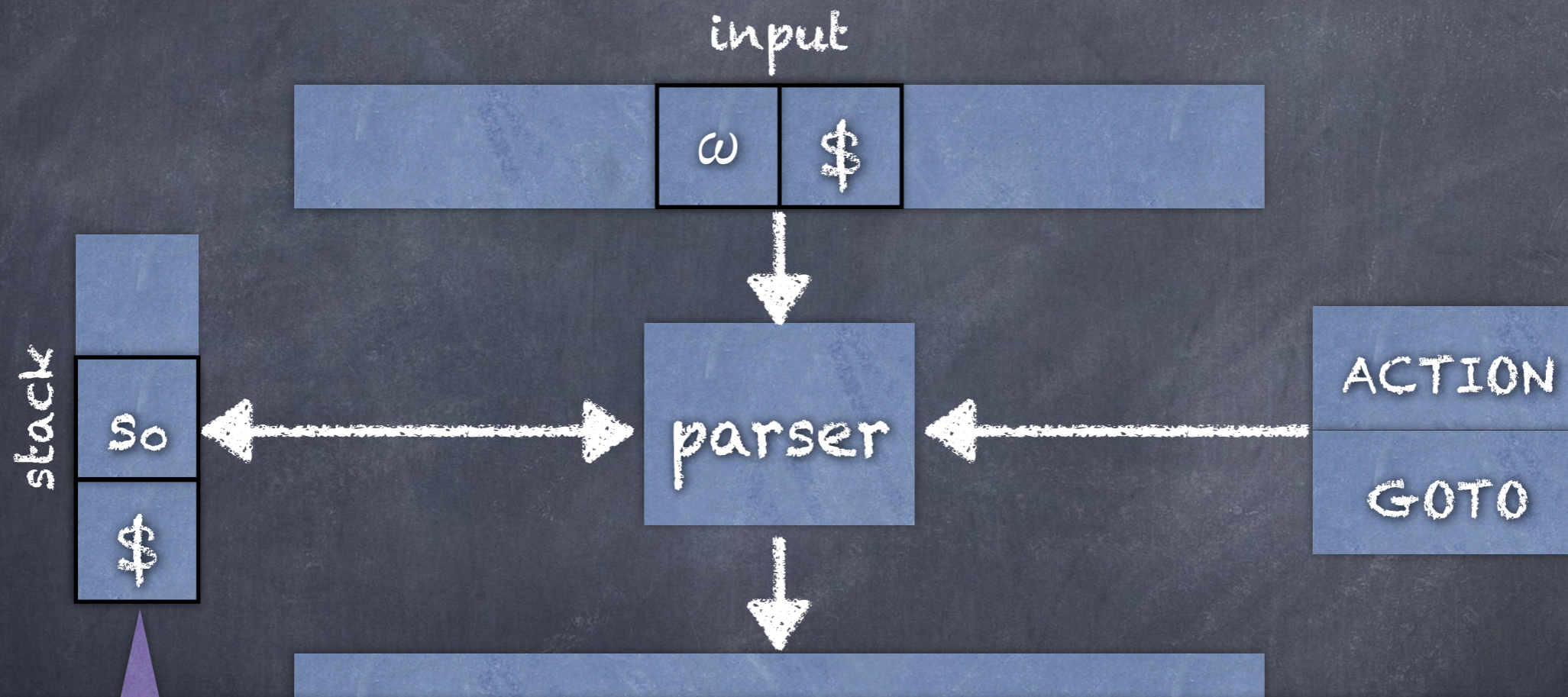


"In the SLR method, the stack holds states from the LR(0) automaton; the canonical LR and LALR methods are similar." [p. 248]

Initial state of the parser

(top of stack is current state in LR(0) automata)

modified from figure 4.35 [p. 248]



"By construction, each state has a corresponding grammar symbol. Recall that states correspond to sets of items, and that there is a transition from state i to state j if $GOTO(I_i, X) = I_j$. All transitions to state j must be for the same grammar symbol X . Thus, each state, except the start state 0 , has a unique grammar symbol associated with it." [p. 248]

written as s_0

LR parsing table

• ACTION function

- Inputs: state i and an input symbol a (terminal or \$)

- ACTION[i, a] is:

* Shift j - shift a onto stack, using state j to represent a

* Reduce $A \rightarrow \beta$

* Accept

* Error

• GOTO function - extend from sets of items to states.

- GOTO[I_i, A] = $I_j \Rightarrow$ GOTO[i, A] = j

LR parser configuration

- An LR parser configuration is a pair:

$(s_0 s_1 \dots s_m, a_i a_{i+1} \dots a_n \$)$

- $s_0 s_1 \dots s_m$ is the stack (bottom to top)

- $a_i a_{i+1} \dots a_n \$$ is the (remaining) input

- Represents the right-sentential form

$X_1 X_2 \dots X_m a_i a_{i+1} \dots a_n$

Algorithm 4.46 [p. 253]

Constructing an SLR-parsing table

INPUT: An augmented grammar G'

OUTPUT: The SLR-parsing table functions ACTION and GOTO for G'

METHOD:

1. Construct $C = \{I_0, I_1, \dots, I_n\}$, the collection of sets of LR(0) items for G'

2. State i is constructed from I_i . The parsing items for state i are determined as follows:

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

B. If $[A \rightarrow \alpha \circ]$ is in I_i , then set $\text{ACTION}[i, a]$ to "reduce $A \rightarrow \alpha$ " for all a in $\text{FOLLOW}(A)$; here A may not be S' .

C. If $[S' \rightarrow S \circ]$ is in I_i , then set $\text{ACTION}[i, \$]$ to "accept."

If conflicting actions result from the above rules, we say the grammar is not SLR(1). The algorithm fails to produce a parser in this case.

3. The goto transitions for state I are constructed for all nonterminals A using the rule: If $\text{GOTO}(I_i, A) = I_j$, then $\text{GOTO}[i, A] = j$.

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 $[S' \rightarrow \circ S]$

FIRST(X)

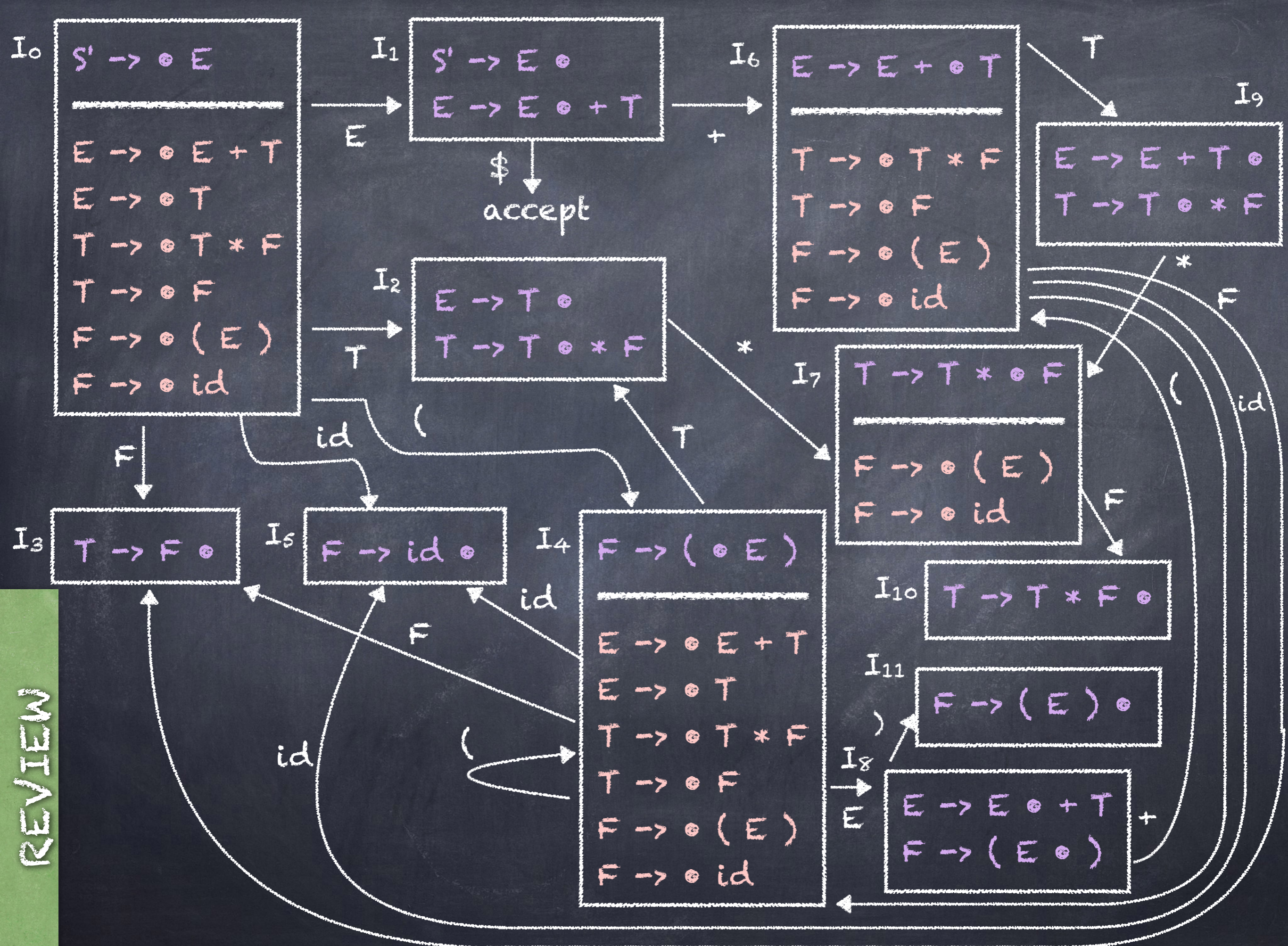
- if $X \in T$ then $\text{FIRST}(X) = \{X\}$
- if $X \in N$ and $X \rightarrow Y_1 Y_2 \dots Y_k \in P$ for $k \geq 1$, then
 - add $a \in T$ to $\text{FIRST}(X)$ if $\exists i$ s.t. $a \in \text{FIRST}(Y_i)$ and $\epsilon \in \text{FIRST}(Y_j) \forall j < i$ (i.e. $Y_1 Y_2 \dots Y_k \Rightarrow^* \epsilon$)
 - if $\epsilon \in \text{FIRST}(Y_j) \forall j < k$ add ϵ to $\text{FIRST}(X)$

FOLLOW(X)

- Place $\$$ in FOLLOW(S), where S is the start symbol ($\$$ is an end marker)
- if $A \rightarrow \alpha B \beta \in P$, then $\text{FIRST}(\beta) - \{\epsilon\}$ is in FOLLOW(B)
- if $A \rightarrow \alpha B \in P$ or $A \rightarrow \alpha B \beta \in P$ where $\epsilon \in \text{FIRST}(\beta)$, then everything in FOLLOW(A) is in FOLLOW(B)

FIRST(X) and FOLLOW(X)

X	FIRST(X)	FOLLOW(X)
S'	id, (\$
E	id, (+,), \$
T	id, (*, +,), \$
F	id, (*, +,), \$
id	id	*, +,), \$
((id, (
))	*, +,), \$
+	+	id, (
*	*	id, (



REVIEW

2A. If $[A \rightarrow \alpha \bullet a \beta]$ 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. This will be written as 'sj' where j is a state number.

2B. If $[A \rightarrow \alpha \bullet]$ is in I_i , then set $ACTION[i, a]$ to "reduce $A \rightarrow \alpha$ " for all a in $FOLLOW(A)$; here A may not be S' . This will be written as 'rp' where p is a production number (see below).

2C. If $[S' \rightarrow S \bullet]$ is in I_i , then set $ACTION[i, \$]$ to "accept."

3. The goto transitions for state I are constructed for all nonterminals A using the rule: If $GOTO(I_i, A) = I_j$, then $GOTO[i, A] = j$. This will be written as 'j' where j is a state number.

Production numbers:

1. $E \rightarrow E + T$

2. $E \rightarrow T$

3. $T \rightarrow T * F$

4. $T \rightarrow F$

5. $F \rightarrow (E)$

6. $F \rightarrow id$

Figure 4.37 [p. 252]

Parsing table for expression grammar

Production numbers:

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

STATE	ACTION						GOTO		
	id	+	*	()	\$	E	T	F
0	ss			s4			1	2	3
1	<div style="border: 1px dashed white; padding: 10px; margin: 10px;"> <p>I_0</p> <p>$S' \rightarrow \bullet E$</p> <hr style="border: 1px solid white;"/> <p>$E \rightarrow \bullet E + T$</p> <p>$E \rightarrow \bullet T$</p> <p>$T \rightarrow \bullet T * F$</p> <p>$T \rightarrow \bullet F$</p> <p>$F \rightarrow \bullet (E)$</p> <p>$F \rightarrow \bullet id$</p> </div>								
2									
3									
4									
5									
6									
7									
8									
9									
10									
11									

REVIEW

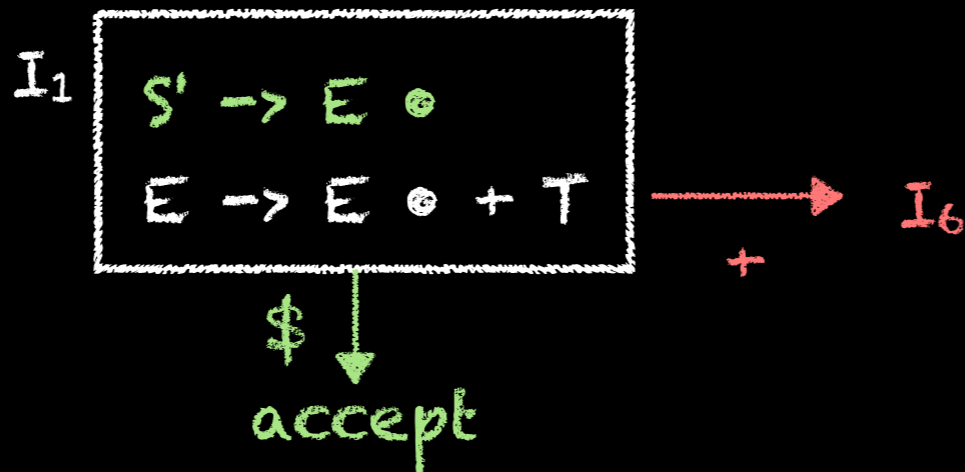
Figure 4.37 [p. 252]

Parsing table for expression grammar

Production numbers:

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

STATE	ACTION						GOTO		
	id	+	*	()	\$	E	T	F
0	s5			s4			1	2	3
1		s6				accept			
2									
3									
4									
5									
6									
7									
8									
9									
10									
11									



REVIEW

Figure 4.37 [p. 252]

Parsing table for expression grammar

Production numbers:

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

STATE	ACTION						GOTO		
	id	+	*	()	\$	E	T	F
0	s5			s4			1	2	3
1		s6				accept			
2		r2	s7		r2	r2			



REVIEW

- 3
- 4
- 5
- 6
- 7
- 8
- 9
- 10
- 11

Figure 4.37 [p. 252]

Parsing table for expression grammar

Production numbers:

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

STATE	ACTION						GOTO		
	id	+	*	()	\$	E	T	F
0	s5			s4			1	2	3
1		s6				accept			
2		r2	s7		r2	r2			
3		r4	r4		r4	r4			
4	<div style="border: 1px dashed black; padding: 5px; display: inline-block;"> $I_3 \quad T \rightarrow F \circ$ </div>								
5									
6									
7									
8									
9									
10									
11									

REVIEW

Figure 4.37 [p. 252]

Parsing table for expression grammar

Production numbers:

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

STATE	ACTION						GOTO		
	id	+	*	()	\$	E	T	F
0	s5			s4			1	2	3
1		s6				accept			
2		r2	s7		r2	r2			
3		r4	r4		r4	r4			
4	... and so on ...								
5									
6									
7									
8									
9									
10									
11									

REVIEW

Figure 4.37 [p. 252]

Parsing table for expression grammar

Production numbers:

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

STATE	ACTION						GOTO		
	id	+	*	()	\$	E	T	F
0	s5			s4			1	2	3
1		s6				accept			
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			

Textbook Typo:

- On page 254, line-4:
- "Fig. 4.31" should be "Fig. 4.37".

Phases of a compiler

Syntactic
structure

Symbol Table

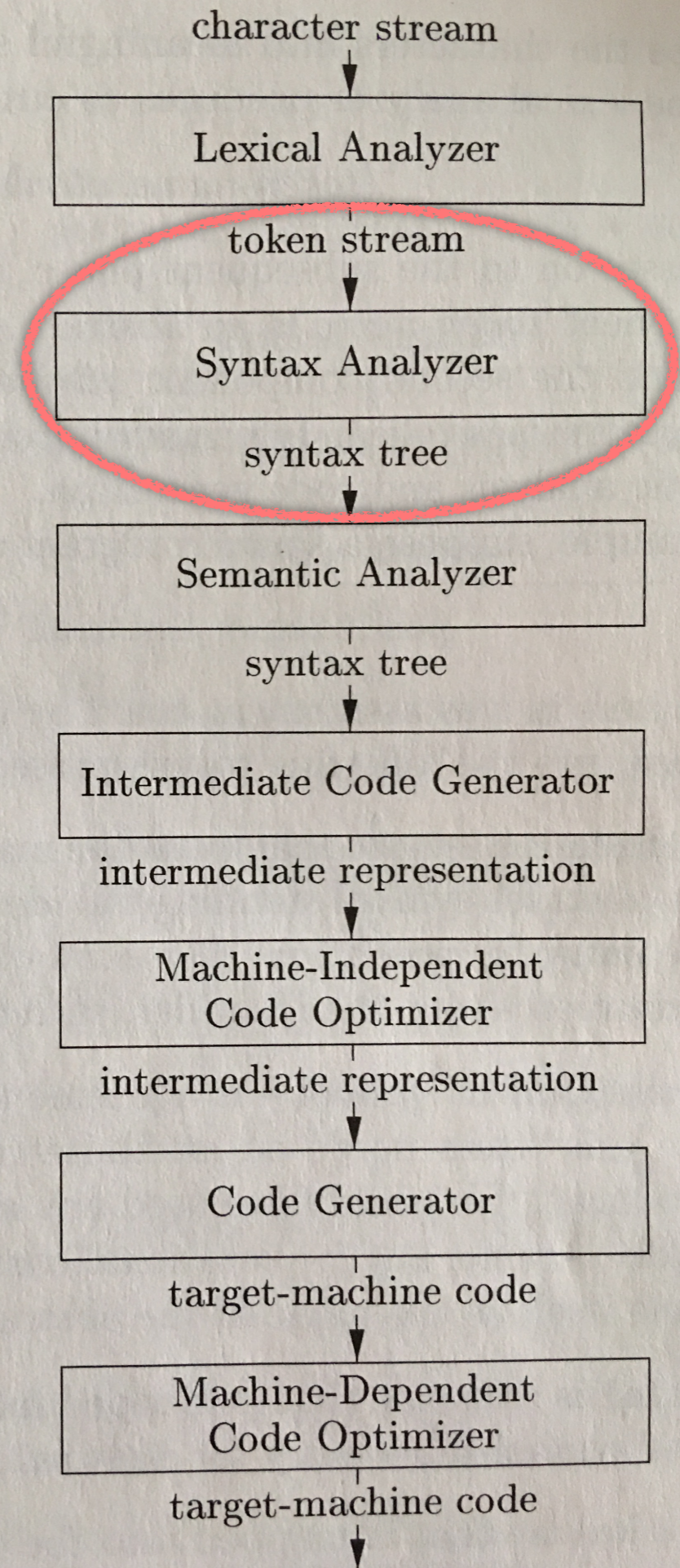


Figure 1.7,
page 5 of text

Algorithm 4.44 [p. 250-251]

The LR-parsing algorithm

- INPUT: An input string w and an LR-parsing table with functions ACTION and GOTO for a grammar G
- OUTPUT: If w is in $L(G)$, the reduction steps of a bottom-up parse for w ; otherwise, an error indication
- METHOD: Initially, the parser has s_0 on its stack, where s_0 is the initial state. The parser then executes the program in Fig. 4.36.

Figure 4.36 [p. 251]

Let a be the first symbol of $w\$$

while (true) {

 let s be the state on top of the stack

 if ($\text{ACTION}[s,a] = \text{shift } t$) {

 push t onto the stack

 let a be the next input symbol

 } else if ($\text{ACTION}[s,a] = \text{reduce } A \rightarrow \beta$) {

 pop $|\beta|$ symbols off the stack

 let state t now be on top of the stack

 push $\text{GOTO}[t,A]$ onto the stack

 output the production $A \rightarrow \beta$

 } else if ($\text{ACTION}[s,a] = \text{accept}$) break

 else call error-recovery routine

}

Figure 4.37 [p. 252]

Parsing table for expression grammar

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

STATE	ACTION						GOTO		
	id	+	*	()	\$	E	T	F
0	s5			s4			1	2	3
1		s6				accept			
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			

REVIEW

Parsing: $id + id * id$

Stack		Input		Action		Output			
\$ 0		$id + id * id$ \$							
STATE	ACTION						GOTO		
	id	+	*	()	\$	E	T	F
0	s5			s4			1	2	3
1		s6				accept			
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		r	r		r	r			

Parsing: $id + id * id$

Stack		Input		Action		Output			
\$ 0		$id + id * id$ \$		s5					
STATE	ACTION						GOTO		
	id	+	*	()	\$	E	T	F
0	s5			s4			1	2	3
1		s6				accept			
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		r	r		r	r			

Parsing: $id + id * id$

Stack	Input	Action	Output
\$ 0	id + id * id \$	s5	
\$ 0 s	+ id * id \$		

STATE	ACTION						GOTO		
	id	+	*	()	\$	E	T	F
0	s5			s4			1	2	3
1		s6				accept			
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		r2	s7		r2	r2			

Parsing: $id + id * id$

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

Stack	Input	Action	
\$ 0	id + id * id \$	s5	
\$ 0 5	+ id * id \$	r6	$F \rightarrow id$
\$ 0	+ id * id \$		

STATE	ACTION						GOTO		
	id	+	*	()	\$	E	T	F
0	s5			s4			1	2	3
1		s6				accept			
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				

Parsing: $id + id * id$

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

Stack	Input	Action	
\$ 0	id + id * id \$	s5	
\$ 0 5	+ id * id \$	r6	$F \rightarrow id$
\$ 0	+ id * id \$		

STATE	ACTION						GOTO		
	id	+	*	()	\$	E	T	F
0	s5			s4			1	2	3
1		s6				accept			
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				

Parsing: $id + id * id$

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

Stack	Input	Action	
\$ 0	id + id * id \$	s5	
\$ 0 5	+ id * id \$	r6	$F \rightarrow id$
\$ 0 3	+ id * id \$		

STATE	ACTION						GOTO		
	id	+	*	()	\$	E	T	F
0	s5			s4			1	2	3
1		s6				accept			
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				

Parsing: $id + id * id$

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

Stack	Input	Action	
\$ 0	id + id * id \$	s5	
\$ 0 5	+ id * id \$	r6	$F \rightarrow id$
\$ 0 3	+ id * id \$		

STATE	ACTION						GOTO		
	id	+	*	()	\$	E	T	F
0	s5			s4			1	2	3
1		s6				accept			
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				

Parsing: $id + id * id$

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

Stack	Input	Action	
\$ 0	$id + id * id \$$	s5	
\$ 0 5	$+ id * id \$$	r6	$F \rightarrow id$
\$ 0 3	$+ id * id \$$	r4	$T \rightarrow F$

STATE	ACTION						GOTO		
	id	+	*	()	\$	E	T	F
0	s5			s4			1	2	3
1		s6				accept			
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				

Parsing: $id + id * id$

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

Stack	Input	Action	
\$ 0	$id + id * id \$$	s5	
\$ 0 5	$+ id * id \$$	r6	$F \rightarrow id$
\$ 0 3	$+ id * id \$$	r4	$T \rightarrow F$
\$ 0 2	$+ id * id \$$	r2	$E \rightarrow T$

STATE	ACTION						GOTO		
	id	+	*	()	\$	E	T	F
0	s5			s4			1	2	3
1		s6				accept			
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

Parsing: $id + id * id$

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

Stack	Input	Action	
\$ 0	$id + id * id \$$	s5	
\$ 0 5	$+ id * id \$$	r6	$F \rightarrow id$
\$ 0 3	$+ id * id \$$	r4	$T \rightarrow F$
\$ 0 2	$+ id * id \$$	r2	$E \rightarrow T$

Try to complete the rest on your own!

STATE	ACTION						GOTO		
	id	+	*	()	\$	E	T	F
0	s5			s4			1	2	3
1		s6				accept			
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			

Stack	Input	Action	Output
\$ 0	id + id * id \$	s5	
\$ 0 5	+ id * id \$	r6	F → id
\$ 0 3	+ id * id \$	r4	T → F
\$ 0 2	+ id * id \$	r2	E → T
\$ 0 1	+ id * id \$	s6	
\$ 0 1 6	id * id \$	s5	
\$ 0 1 6 5	* id \$	r6	F → id
\$ 0 1 6 3	* id \$	r4	T → F
\$ 0 1 6 9	* id \$	s7	
\$ 0 1 6 9 7	id \$	s5	
\$ 0 1 6 9 7 5	\$	r6	F → id
\$ 0 1 6 9 7 10	\$	r3	T → T * F
\$ 0 1 6 9	\$	r1	E → E + T
\$ 0 1	\$	accept	

This is the output

$F \rightarrow id$

$T \rightarrow F$

$E \rightarrow T$

$F \rightarrow id$

$T \rightarrow F$

$F \rightarrow id$

$T \rightarrow T * F$

$E \rightarrow E + T$

It is a rightmost derivation in reverse

$$E \rightarrow E + T$$

$$T \rightarrow T * F$$

$$F \rightarrow id$$

$$T \rightarrow F$$

$$F \rightarrow id$$

$$E \rightarrow T$$

$$T \rightarrow F$$

$$F \rightarrow id$$

Here's the derivation:

$$E \rightarrow E + T$$

$$T \rightarrow T * F$$

$$F \rightarrow id$$

$$T \rightarrow F$$

$$F \rightarrow id$$

$$E \rightarrow T$$

$$T \rightarrow F$$

$$F \rightarrow id$$

$$E \rightarrow E + T$$

$$\rightarrow E + T * F$$

$$\rightarrow E + T * id$$

$$\rightarrow E + F * id$$

$$\rightarrow E + id * id$$

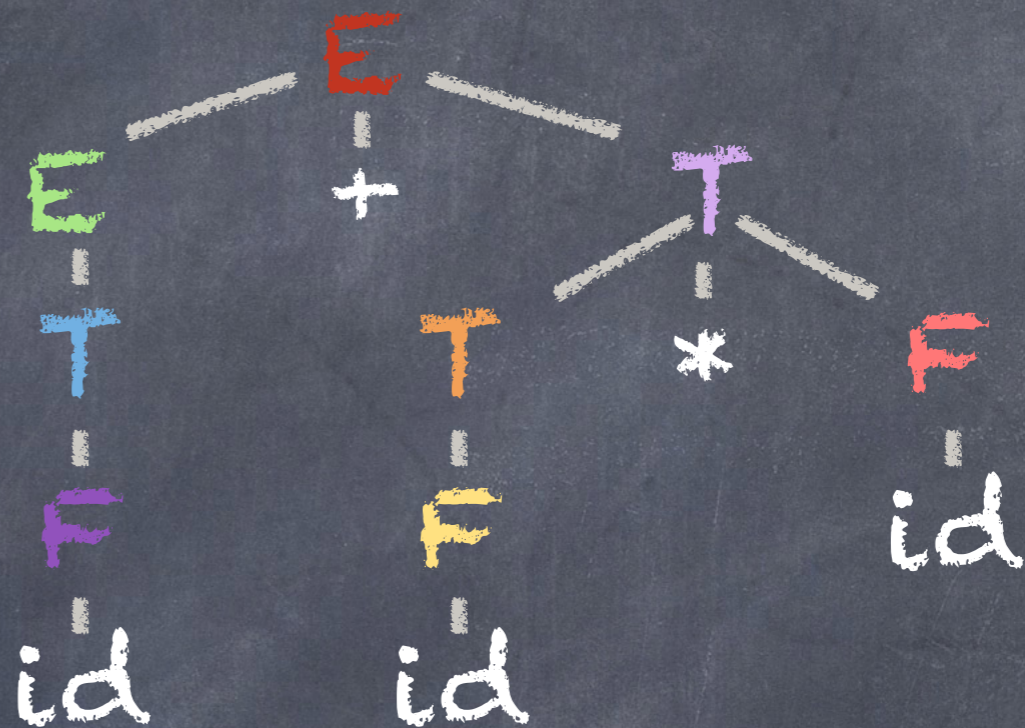
$$\rightarrow T + id * id$$

$$\rightarrow F + id * id$$

$$\rightarrow id + id * id$$

and the corresponding parse tree:

$E \rightarrow E + T$
 $\rightarrow E + T * F$
 $\rightarrow E + T * id$
 $\rightarrow E + F * id$
 $\rightarrow E + id * id$
 $\rightarrow T + id * id$
 $\rightarrow F + id * id$
 $\rightarrow id + id * id$



For Wednesday

- Class will focus on Sprint 2:
 - ▶ structure of Bison's .y file
 - ▶ yylex and yyparse
 - ▶ the union
 - ▶ symbol tables (read esp. section 2.7.1)
 - ▶ general advice