

# Bottom-Up Syntax Analysis

Mooly Sagiv

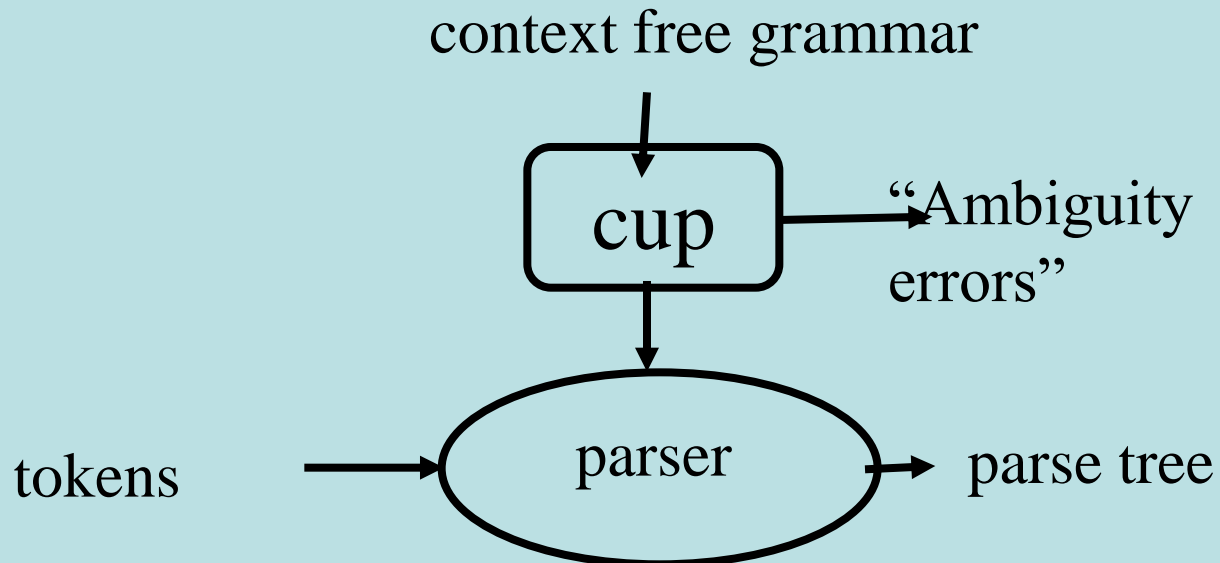
<http://www.cs.tau.ac.il/~msagiv/courses/wcc11-12.html>

Textbook: Modern Compiler Design

Chapter 2.2.5 (modified)

# Efficient Parsers

- Pushdown automata
- Deterministic
- Report an error as soon as the input is not a prefix of a valid program
- Not usable for all context free grammars



# Kinds of Parsers

- Top-Down (Predictive Parsing) LL
  - Construct parse tree in a top-down manner
  - Find the leftmost derivation
  - For every non-terminal and token **predict** the next production
- Bottom-Up LR
  - Construct parse tree in a bottom-up manner
  - Find the rightmost derivation in a reverse order
  - For every potential right hand side and token decide when a production is found

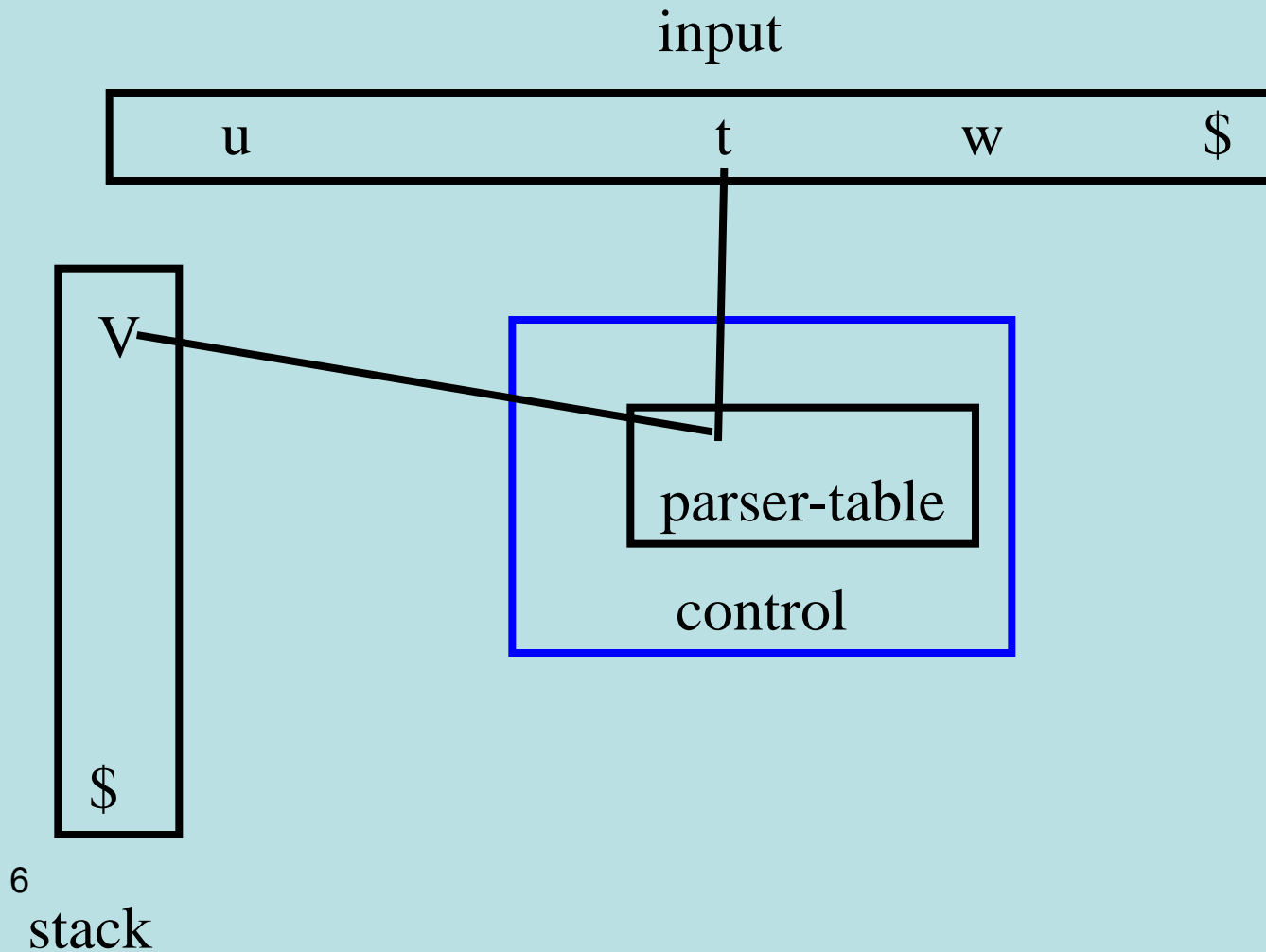
# Bottom-Up Syntax Analysis

- Input
  - A context free grammar
  - A stream of tokens
- Output
  - A syntax tree or error
- Method
  - Construct parse tree in a bottom-up manner
  - Find the rightmost derivation in (reversed order)
  - For every potential right hand side and token decide when a production is found
  - Report an error as soon as the input is not a prefix of valid program

# Plan

- Pushdown automata
- Bottom-up parsing (informal)
- Non-deterministic bottom-up parsing
- Deterministic bottom-up parsing
- Interesting non LR grammars

# Pushdown Automaton



# Informal Example(1)

$S \rightarrow E\$$     $E \rightarrow T \mid E + T$     $T \rightarrow i \mid ( E )$

stack

tree

input

\$

i + i \$

shift

stack

tree

input

i \$

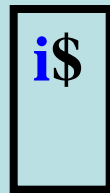
i

+ i \$

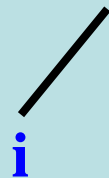
# Informal Example(2)

$S \rightarrow E\$$     $E \rightarrow T \mid E + T$     $T \rightarrow i \mid (E)$

stack



tree



input

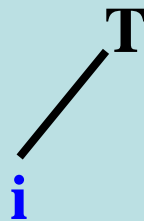


reduce  $T \rightarrow i$

stack



tree



input





# Informal Example(3)

$S \rightarrow E\$$     $E \rightarrow T \mid E + T$     $T \rightarrow i \mid ( E )$

stack

T  
\$

tree

T  
/   
i

input

+ i \$

reduce  $E \rightarrow T$

stack

E  
\$

tree

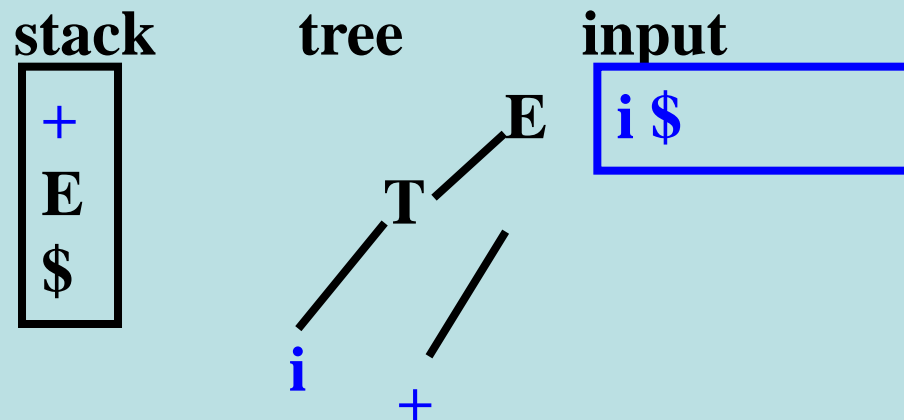
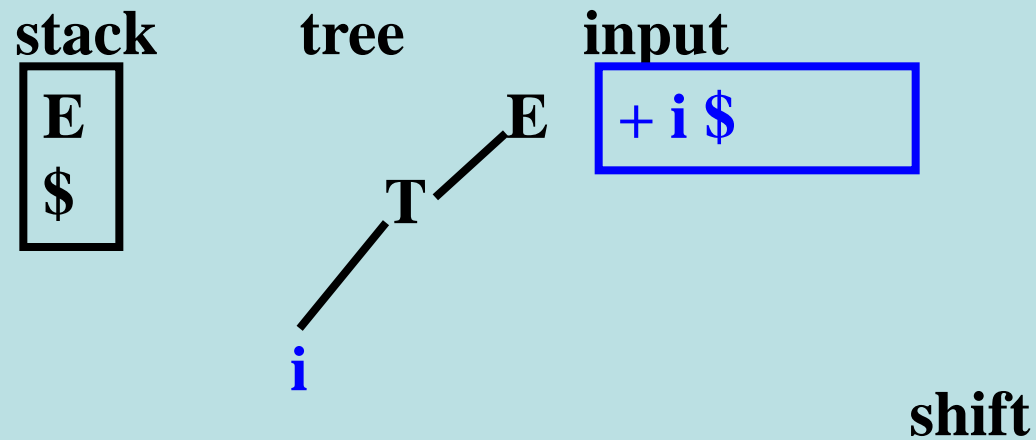
E  
/   
T  
/   
i

input

+ i \$

# Informal Example(4)

$S \rightarrow E\$$     $E \rightarrow T \mid E + T$     $T \rightarrow i \mid ( E )$



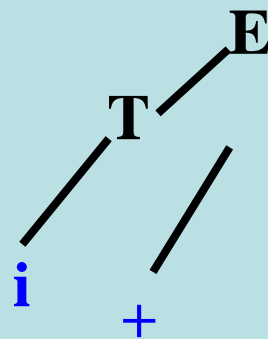
# Informal Example(5)

$S \rightarrow E\$$     $E \rightarrow T \mid E + T$     $T \rightarrow i \mid ( E )$

stack



tree



input

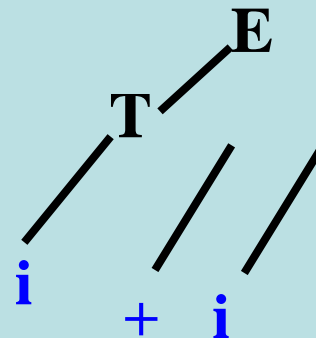


shift

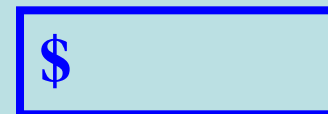
stack



tree

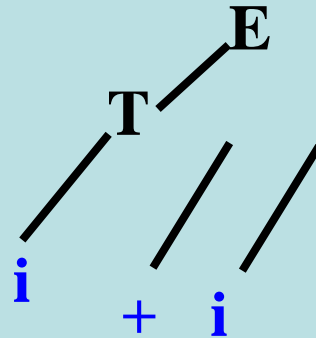


input



# Informal Example(6)

$S \rightarrow E\$$      $E \rightarrow T \mid E + T$      $T \rightarrow i \mid ( E )$   
 stack                      tree                      input

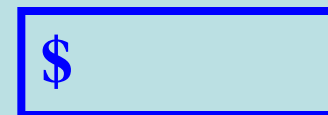
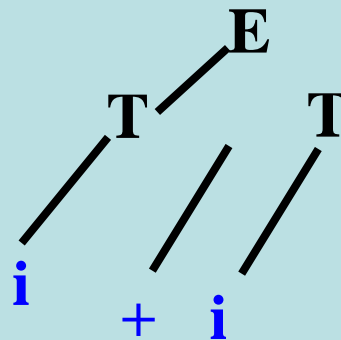


reduce  $T \rightarrow i$

stack

tree

input



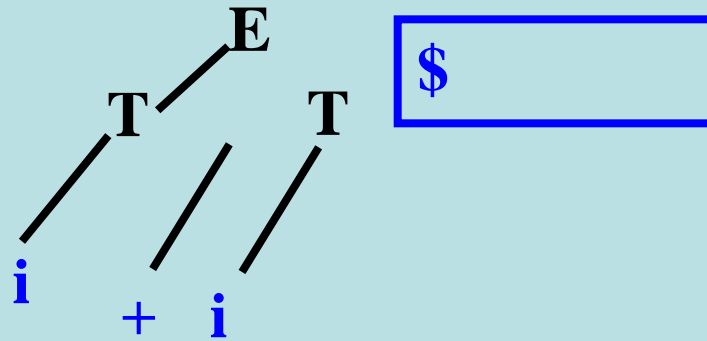
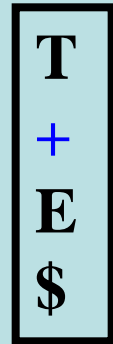
# Informal Example(7)

$S \rightarrow E\$$     $E \rightarrow T \mid E + T$     $T \rightarrow i \mid ( E )$

stack

tree

input

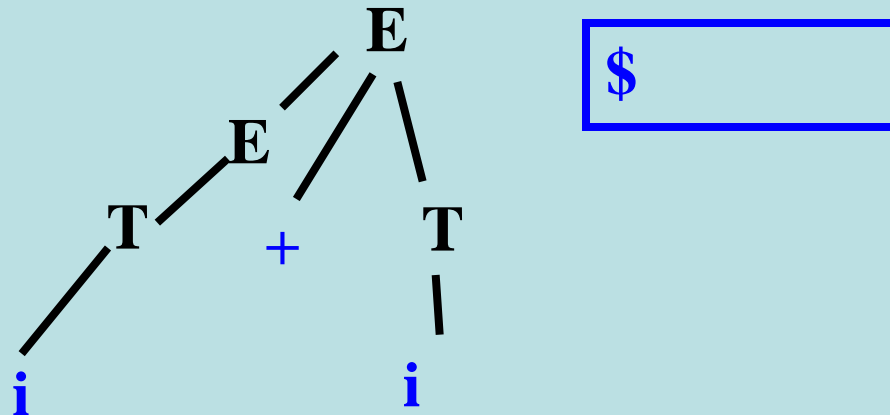


reduce  $E \rightarrow E + T$

stack

tree

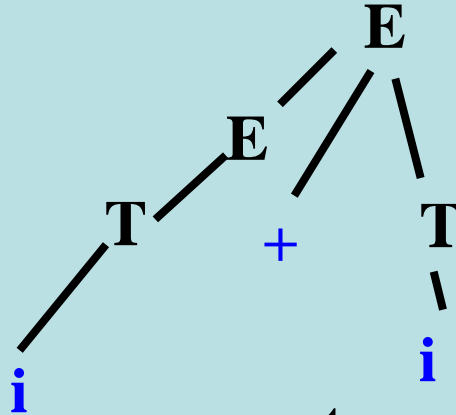
input



# Informal Example(8)

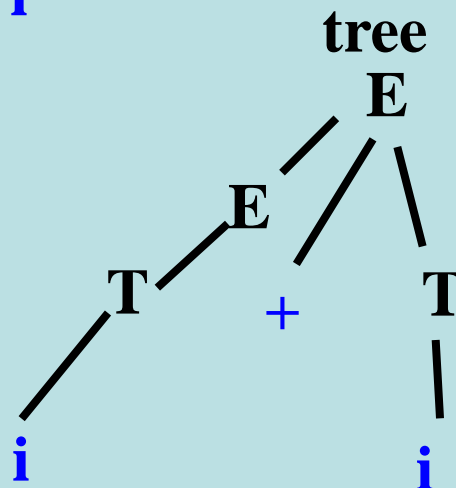
$S \rightarrow E\$$     $E \rightarrow T \mid E + T$     $T \rightarrow i \mid ( E )$   
 tree   input

stack



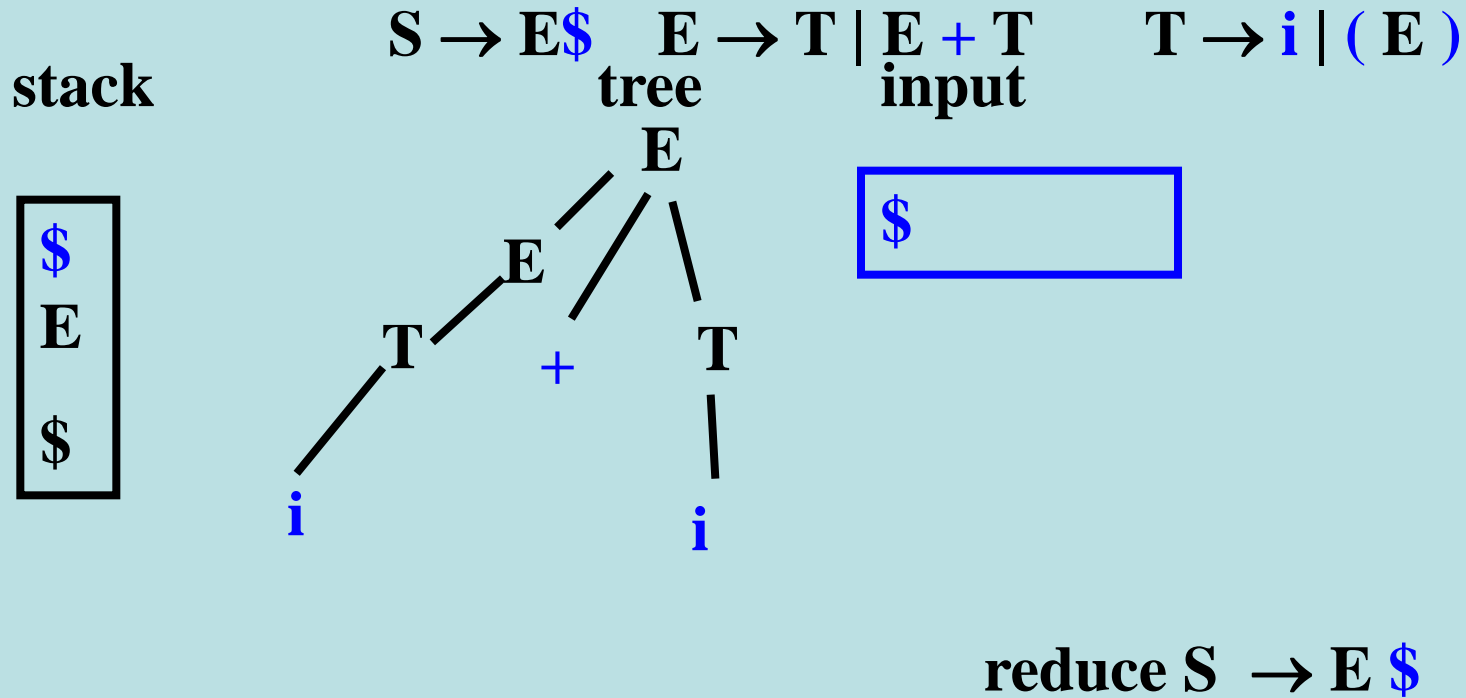
shift

stack



input

# Informal Example(9)



# Informal Example

reduce  $S \rightarrow E \$$

reduce  $E \rightarrow E + T$

reduce  $T \rightarrow i$

reduce  $E \rightarrow T$

reduce  $T \rightarrow i$



# The Problem

- Deciding between shift and reduce

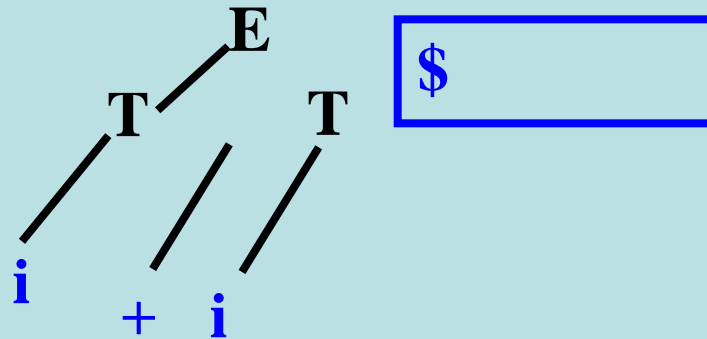
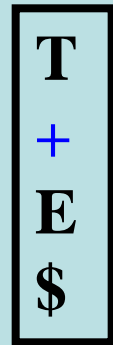
# Informal Example(7)

$S \rightarrow E\$$     $E \rightarrow T \mid E + T$     $T \rightarrow i \mid ( E )$

stack

tree

input

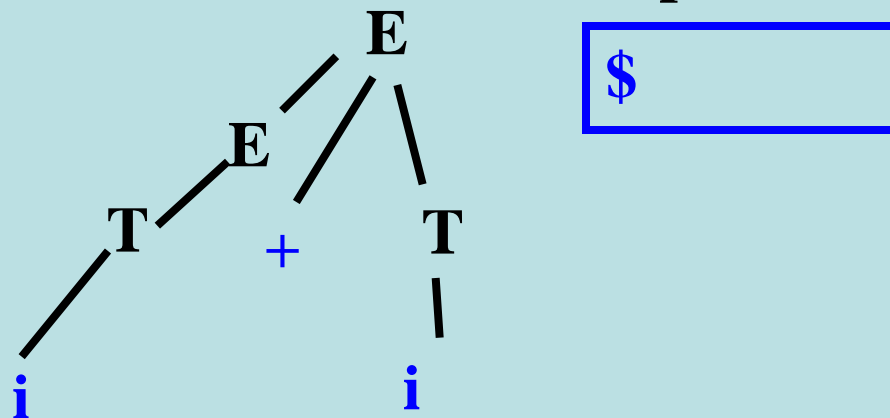


reduce  $E \rightarrow E + T$

stack

tree

input



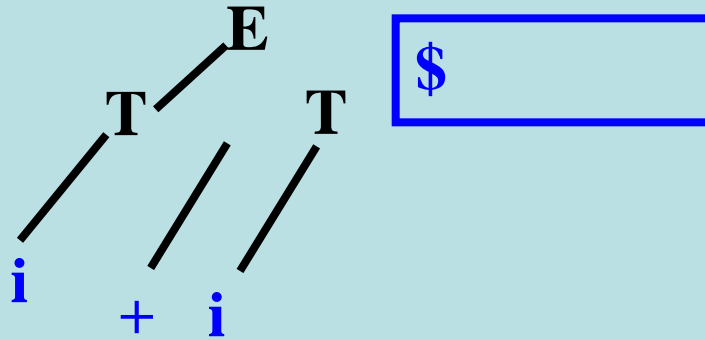
# Informal Example(7')

$S \rightarrow E\$$     $E \rightarrow T \mid E + T$     $T \rightarrow i \mid ( E )$

stack

tree

input

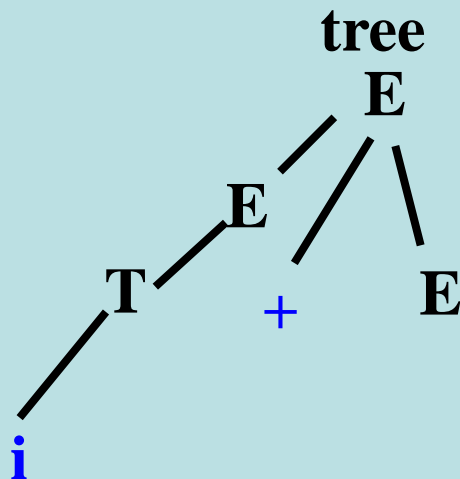
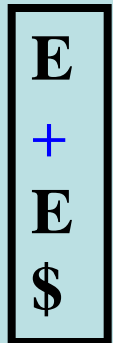


reduce  $E \rightarrow T$

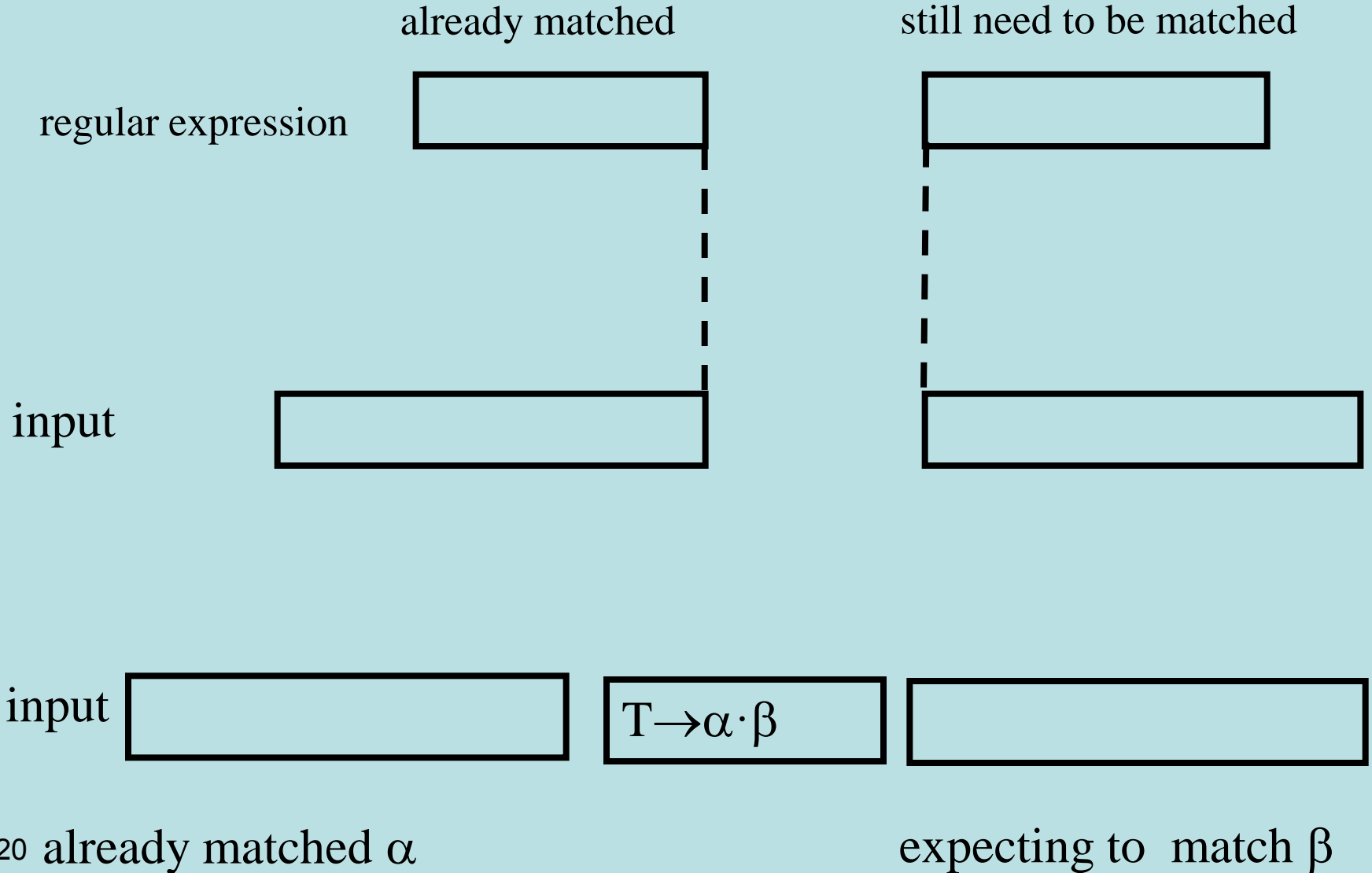
stack

tree

input



# Bottom-UP LR(0) Items



$S \rightarrow E\$$

$E \rightarrow T$

$E \rightarrow E + T$

$T \rightarrow i$

$T \rightarrow (E)$

LR(0) items	(	)	i	+	\$	T	E	$\epsilon$
1: $S \rightarrow \bullet E \$$							2	4, 6
2: $S \rightarrow E \bullet \$$					s3			
3: $S \rightarrow E \$ \bullet$								r
4: $E \rightarrow \bullet T$						5		10, 12
5: $E \rightarrow T \bullet$								r
6: $E \rightarrow \bullet E + T$							7	4, 6
7: $E \rightarrow E \bullet + T$				s8				
8: $E \rightarrow E + \bullet T$						9		10, 12
9: $E \rightarrow E + T \bullet$								r
10: $T \rightarrow \bullet i$			s11					
11: $T \rightarrow i \bullet$								r
12: $T \rightarrow \bullet (E)$	s13							
13: $T \rightarrow (\bullet E)$							14	4, 6
14: $T \rightarrow (E \bullet)$		s15						
15: $T \rightarrow (E) \bullet$								r

# Formal Example(1)

$S \rightarrow E\$$     $E \rightarrow T \mid E + T$     $T \rightarrow i \mid ( E )$

stack

1:  $S \rightarrow \bullet E\$$

input

$i + i \$$

$\epsilon$ -move 6

stack

6:  $E \rightarrow \bullet E + T$   
1:  $S \rightarrow \bullet E\$$

input

$i + i \$$

# Formal Example(2)

$S \rightarrow E\$$     $E \rightarrow T \mid E + T$     $T \rightarrow i \mid ( E )$

stack

input

6:  $E \rightarrow \bullet E + T$   
1:  $S \rightarrow \bullet E \$$

$i + i \$$

$\epsilon$ -move 4

stack

input

4:  $E \rightarrow \bullet T$   
6:  $E \rightarrow \bullet E + T$   
1:  $S \rightarrow \bullet E \$$

$i + i \$$

# Formal Example(3)

$S \rightarrow E\$$     $E \rightarrow T \mid E + T$     $T \rightarrow i \mid ( E )$   
stack                                  input

4:  $E \rightarrow \bullet T$   
6:  $E \rightarrow \bullet E + T$   
1:  $S \rightarrow \bullet E \$$

$i + i \$$

stack

input

$\epsilon$ -move 10

10:  $T \rightarrow \bullet i$   
4:  $E \rightarrow \bullet T$   
6:  $E \rightarrow \bullet E + T$   
1:  $S \rightarrow \bullet E \$$

$i + i \$$



# Formal Example(4)

$S \rightarrow E\$$     $E \rightarrow T \mid E + T$     $T \rightarrow i \mid ( E )$

stack

10:  $T \rightarrow \bullet i$   
4:  $E \rightarrow \bullet T$   
6:  $E \rightarrow \bullet E + T$   
1:  $S \rightarrow \bullet E \$$

input

$i + i \$$

stack

11:  $T \rightarrow i \bullet$   
10:  $T \rightarrow \bullet i$   
4:  $E \rightarrow \bullet T$   
6:  $E \rightarrow \bullet E + T$   
1:  $S \rightarrow \bullet E \$$

input

$+ i \$$

shift 11

# Formal Example(5)

$S \rightarrow E\$$     $E \rightarrow T \mid E + T$     $T \rightarrow i \mid ( E )$

stack

input

11:  $T \rightarrow i \bullet$

10:  $T \rightarrow \bullet i$

4:  $E \rightarrow \bullet T$

6:  $E \rightarrow \bullet E + T$

1:  $S \rightarrow \bullet E \$$

+ i \$

stack

input

reduce  $T \rightarrow i$

5:  $E \rightarrow T \bullet$

4:  $E \rightarrow \bullet T$

6:  $E \rightarrow \bullet E + T$

1:  $S \rightarrow \bullet E \$$

+ i \$

# Formal Example(6)

$S \rightarrow E\$$     $E \rightarrow T \mid E + T$     $T \rightarrow i \mid ( E )$

stack

input

5:  $E \rightarrow T \bullet$

4:  $E \rightarrow \bullet T$

6:  $E \rightarrow \bullet E + T$

1:  $S \rightarrow \bullet E \$$

+ i \$

stack

input

reduce  $E \rightarrow T$

7:  $E \rightarrow E \bullet + T$

6:  $E \rightarrow \bullet E + T$

1:  $S \rightarrow \bullet E \$$

+ i \$

# Formal Example(7)

$S \rightarrow E\$$     $E \rightarrow T \mid E + T$     $T \rightarrow i \mid ( E )$

stack

input

7:  $E \rightarrow E \bullet + T$

6:  $E \rightarrow \bullet E + T$

1:  $S \rightarrow \bullet E \$$

+ i \$

stack

input

shift 8

8:  $E \rightarrow E + \bullet T$

7:  $E \rightarrow E \bullet + T$

6:  $E \rightarrow \bullet E + T$

1:  $S \rightarrow \bullet E \$$

i \$

# Formal Example(8)

$S \rightarrow E\$$     $E \rightarrow T \mid E + T$     $T \rightarrow i \mid ( E )$

stack

input

8:  $E \rightarrow E + \bullet T$   
7:  $E \rightarrow E \bullet + T$   
6:  $E \rightarrow \bullet E + T$   
1:  $S \rightarrow \bullet E \$$

i \$

stack

input

$\epsilon$ -move 10

10:  $T \rightarrow \bullet i$   
8:  $E \rightarrow E + \bullet T$   
7:  $E \rightarrow E \bullet + T$   
6:  $E \rightarrow \bullet E + T$   
1:  $S \rightarrow \bullet E \$$

i \$

# Formal Example(9)

$S \rightarrow E\$$     $E \rightarrow T \mid E + T$     $T \rightarrow i \mid ( E )$

stack

input

10:  $T \rightarrow \bullet i$   
8:  $E \rightarrow E + \bullet T$   
7:  $E \rightarrow E \bullet + T$   
6:  $E \rightarrow \bullet E + T$   
1:  $S \rightarrow \bullet E \$$

i \$

stack

input

shift 11

11:  $T \rightarrow i \bullet$   
10:  $T \rightarrow \bullet i$   
8:  $E \rightarrow E + \bullet T$   
7:  $E \rightarrow E \bullet + T$   
6:  $E \rightarrow \bullet E + T$   
1:  $S \rightarrow \bullet E \$$

\$

# Formal Example(10)

$S \rightarrow E\$$     $E \rightarrow T \mid E + T$     $T \rightarrow i \mid ( E )$

stack

11:  $T \rightarrow i \bullet$   
10:  $T \rightarrow \bullet i$   
8:  $E \rightarrow E + \bullet T$   
7:  $E \rightarrow E \bullet + T$   
6:  $E \rightarrow \bullet E + T$   
1:  $S \rightarrow \bullet E \$$

input

\$

stack

9:  $E \rightarrow E + T \bullet$   
8:  $E \rightarrow E + \bullet T$   
7:  $E \rightarrow E \bullet + T$   
6:  $E \rightarrow \bullet E + T$   
1:  $S \rightarrow \bullet E \$$

input

\$

reduce  $T \rightarrow i$

# Formal Example(11)

$S \rightarrow E\$$     $E \rightarrow T \mid E + T$     $T \rightarrow i \mid (E)$

stack

9:  $E \rightarrow E + T \bullet$   
8:  $E \rightarrow E + \bullet T$   
7:  $E \rightarrow E \bullet + T$   
6:  $E \rightarrow \bullet E + T$   
1:  $S \rightarrow \bullet E \$$

input

\$

reduce  $E \rightarrow E + T$

stack

2:  $S \rightarrow E \bullet \$$   
1:  $S \rightarrow \bullet E \$$

input

\$





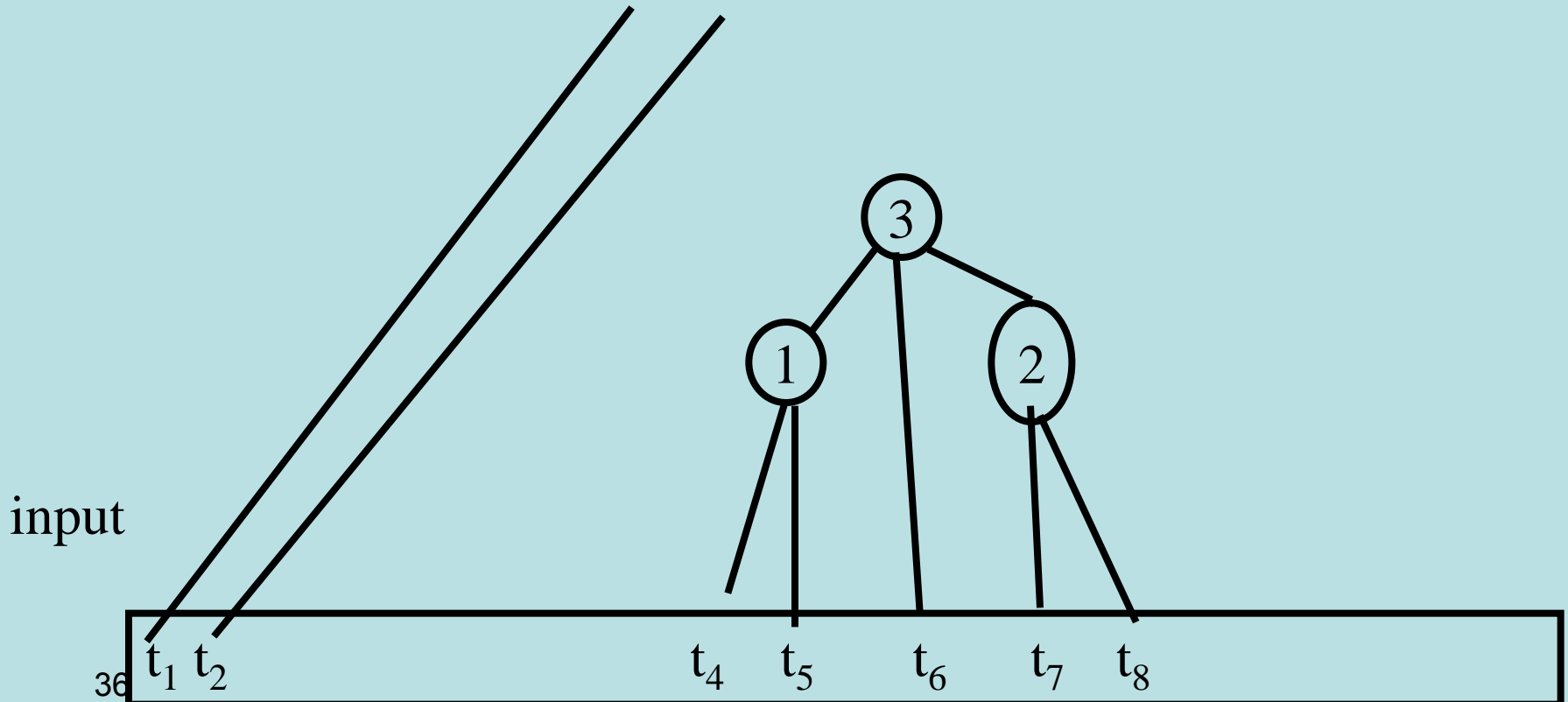


But how can this be done  
efficiently?

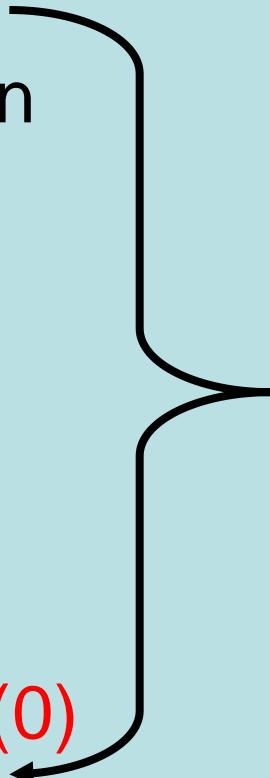
Deterministic Pushdown  
Automaton

# Handles

- Identify the leftmost node (nonterminal) that has not been constructed but all whose children have been constructed



# Identifying Handles

- Create a deterministic finite state automaton over grammar symbols
    - Sets of LR(0) items
    - Accepting states identify handles
  - Use automaton to build parser tables
    - **reduce** For items  $A \rightarrow \alpha \bullet$  on every token
    - **shift** For items  $A \rightarrow \alpha \bullet t \beta$  on token  $t$
  - **When conflicts occur the grammar is not LR(0)**
  - When no conflicts occur use a DPDA which pushes states on the stack
- 

# A Trivial Example

- $S \rightarrow A B \$$
- $A \rightarrow a$
- $B \rightarrow a$

$S \rightarrow E\$$

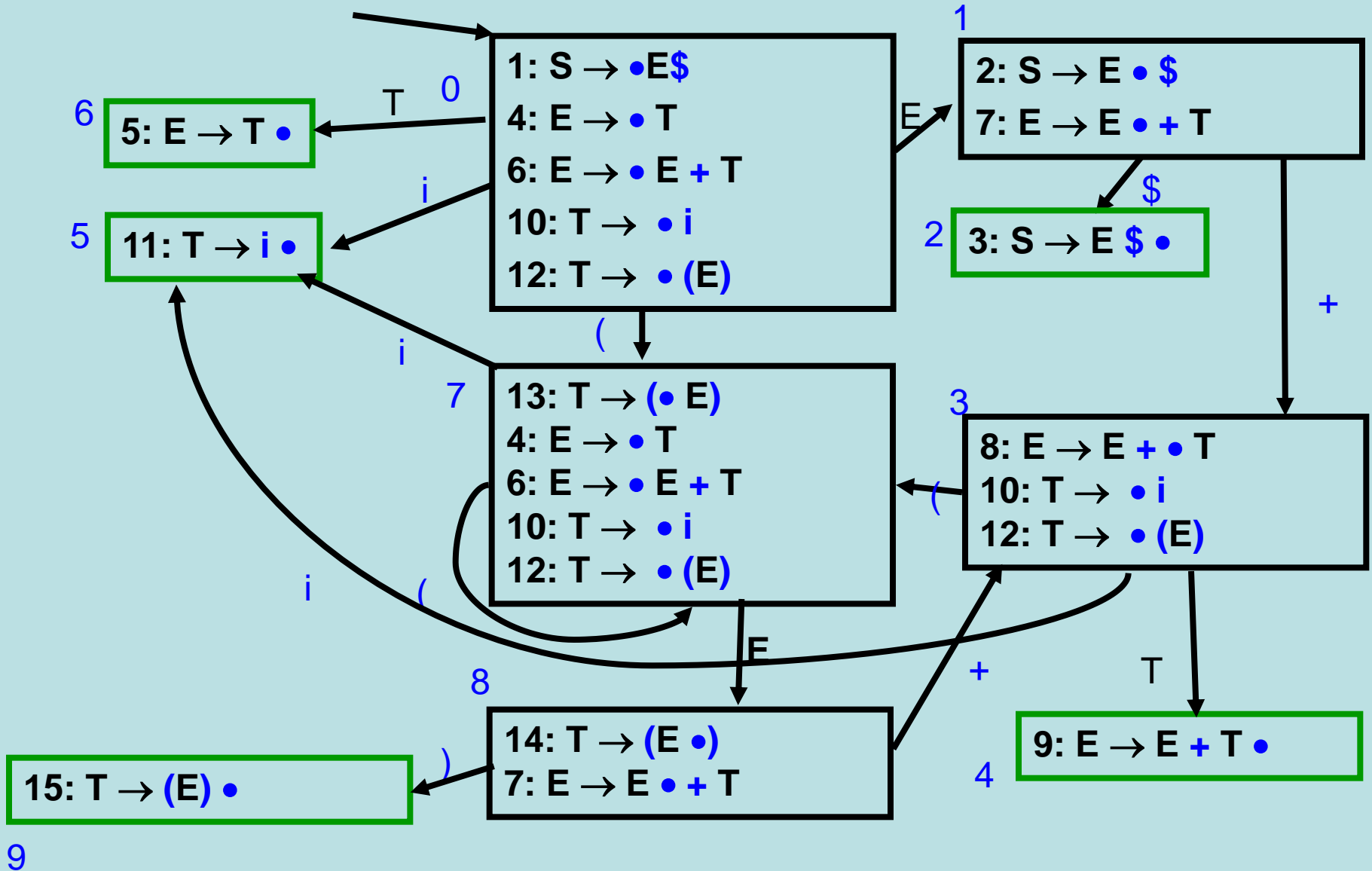
$E \rightarrow T$

$E \rightarrow E + T$

$T \rightarrow i$

$T \rightarrow (E)$

	(	)	i	+	\$	T	E	$\epsilon$
1: $S \rightarrow \bullet E\$$							2	4, 6
2: $S \rightarrow E \bullet \$$					s3			
3: $S \rightarrow E \$ \bullet$								r
4: $E \rightarrow \bullet T$						5		10, 12
5: $E \rightarrow T \bullet$								r
6: $E \rightarrow \bullet E + T$							7	4, 6
7: $E \rightarrow E \bullet + T$				s8				
8: $E \rightarrow E + \bullet T$						9		10, 12
9: $E \rightarrow E + T \bullet$								r
10: $T \rightarrow \bullet i$			s11					
11: $T \rightarrow i \bullet$								r
12: $T \rightarrow \bullet (E)$	s13							
13: $T \rightarrow (\bullet E)$							14	4, 6
14: $T \rightarrow (E \bullet)$		s15						
15: $T \rightarrow (E) \bullet$								r





# Example Control Table

	i	+	(	)	\$	E	T
0	s5	err	s7	err	err	1	6
1	err	s3	err	err	s2		
2	acc						
3	s5	err	s7	err	err		4
4	reduce $E \rightarrow E+T$						
5	reduce $T \rightarrow i$						
6	reduce $E \rightarrow T$						
7	s5	err	s7	err	err	8	6
8	err	s3	err	s9	err		
41 9	reduce $T \rightarrow (E)$						

	i	+	(	)	\$	E	T
0	s5	err	s7	err	err	1	6
1	err	s3	err	err	s2		
2	acc						
3	s5	err	s7	err	err		4
4	reduce $E \rightarrow E+T$						
5	reduce $T \rightarrow i$						
6	reduce $E \rightarrow T$						
7	s5	err	s7	err	err	8	6
8	err	s3	err	s9	err		
9	reduce $T \rightarrow (E)$						

**stack**

**0(\$)**

**input**

**i + i \$**

**shift 5**

	i	+	(	)	\$	E	T
0	s5	err	s7	err	err	1	6
1	err	s3	err	err	s2		
2	acc						
3	s5	err	s7	err	err		4
4	reduce $E \rightarrow E+T$						
5	reduce $T \rightarrow i$						
6	reduce $E \rightarrow T$						
7	s5	err	s7	err	err	8	6
8	err	s3	err	s9	err		
9	reduce $T \rightarrow (E)$						

**stack**

<b>5 (i)</b>
<b>0 (\$)</b>

**input**

<b>+ i \$</b>
---------------

**reduce  $T \rightarrow i$**

	i	+	(	)	\$	E	T
0	s5	err	s7	err	err	1	6
1	err	s3	err	err	s2		
2	acc						
3	s5	err	s7	err	err		4
4	reduce $E \rightarrow E+T$						
5	reduce $T \rightarrow i$						
6	reduce $E \rightarrow T$						
7	s5	err	s7	err	err	8	6
8	err	s3	err	s9	err		
9	reduce $T \rightarrow (E)$						

**stack**

6 (T)
0 (\$)

**input**

+ i \$
--------

**reduce  $E \rightarrow T$**

	i	+	(	)	\$	E	T
0	s5	err	s7	err	err	1	6
1	err	s3	err	err	s2		
2	acc						
3	s5	err	s7	err	err		4
4	reduce $E \rightarrow E+T$						
5	reduce $T \rightarrow i$						
6	reduce $E \rightarrow T$						
7	s5	err	s7	err	err	8	6
8	err	s3	err	s9	err		
9	reduce $T \rightarrow (E)$						

**stack**

**input**

1(E)
4(\$)

+ i \$
--------

**shift 3**

	i	+	(	)	\$	E	T
0	s5	err	s7	err	err	1	6
1	err	s3	err	err	s2		
2	acc						
3	s5	err	s7	err	err		4
4	reduce $E \rightarrow E+T$						
5	reduce $T \rightarrow i$						
6	reduce $E \rightarrow T$						
7	s5	err	s7	err	err	8	6
8	err	s3	err	s9	err		
9	reduce $T \rightarrow (E)$						

**stack**

**input**

**shift 5**

3 (+)  
1 (E)  
0 (\$)

i \$

	i	+	(	)	\$	E	T
0	s5	err	s7	err	err	1	6
1	err	s3	err	err	s2		
2	acc						
3	s5	err	s7	err	err		4
4	reduce $E \rightarrow E+T$						
5	reduce $T \rightarrow i$						
6	reduce $E \rightarrow T$						
7	s5	err	s7	err	err	8	6
8	err	s3	err	s9	err		
9	reduce $T \rightarrow (E)$						

stack

5 (i)

3 (+)

1(E)

0(\$)

47

input

\$

reduce  $T \rightarrow i$

	i	+	(	)	\$	E	T
0	s5	err	s7	err	err	1	6
1	err	s3	err	err	s2		
2	acc						
3	s5	err	s7	err	err		4
4	reduce $E \rightarrow E + T$						
5	reduce $T \rightarrow i$						
6	reduce $E \rightarrow T$						
7	s5	err	s7	err	err	8	6
8	err	s3	err	s9	err		
9	reduce $T \rightarrow (E)$						

stack

input

- 4 (T)
- 3 (+)
- 1(E)
- 0(\$)

\$

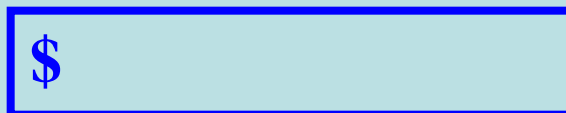
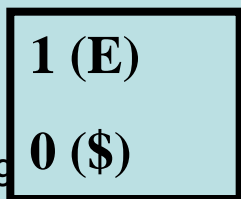
reduce  $E \rightarrow E + T$



	i	+	(	)	\$	E	T
0	s5	err	s7	err	err	1	6
1	err	s3	err	err	s2		
2	acc						
3	s5	err	s7	err	err		4
4	reduce $E \rightarrow E+T$						
5	reduce $T \rightarrow i$						
6	reduce $E \rightarrow T$						
7	s5	err	s7	err	err	8	6
8	err	s3	err	s9	err		
9	reduce $T \rightarrow (E)$						

**stack**

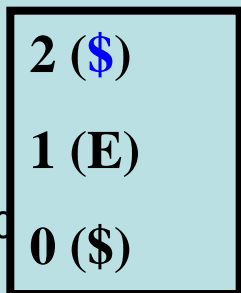
**input**



**shift 2**

	i	+	(	)	\$	E	T
0	s5	err	s7	err	err	1	6
1	err	s3	err	err	s2		
2	acc						
3	s5	err	s7	err	err		4
4	reduce $E \rightarrow E+T$						
5	reduce $T \rightarrow i$						
6	reduce $E \rightarrow T$						
7	s5	err	s7	err	err	8	6
8	err	s3	err	s9	err		
9	reduce $T \rightarrow (E)$						

**stack**



**input**



**accept**

	i	+	(	)	\$	E	T
0	s5	err	s7	err	err	1	6
1	err	s3	err	err	s2		
2	acc						
3	s5	err	s7	err	err		4
4	reduce $E \rightarrow E+T$						
5	reduce $T \rightarrow i$						
6	reduce $E \rightarrow T$						
7	s5	err	s7	err	err	8	6
8	err	s3	err	s9	err		
9	reduce $T \rightarrow (E)$						

**stack**

**0(\$)**

**input**

**((i) \$**

**shift 7**

	i	+	(	)	\$	E	T
0	s5	err	s7	err	err	1	6
1	err	s3	err	err	s2		
2	acc						
3	s5	err	s7	err	err		4
4	reduce $E \rightarrow E+T$						
5	reduce $T \rightarrow i$						
6	reduce $E \rightarrow T$						
7	s5	err	s7	err	err	8	6
8	err	s3	err	s9	err		
9	reduce $T \rightarrow (E)$						

**stack**

**input**

**shift 7**

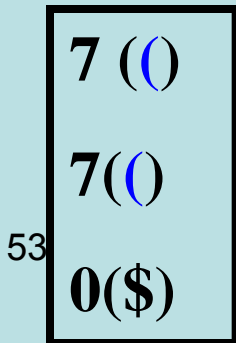
52

7( 0(\$)
-------------

(i) \$
--------

	i	+	(	)	\$	E	T
0	s5	err	s7	err	err	1	6
1	err	s3	err	err	s2		
2	acc						
3	s5	err	s7	err	err		4
4	reduce $E \rightarrow E+T$						
5	reduce $T \rightarrow i$						
6	reduce $E \rightarrow T$						
7	s5	err	s7	err	err	8	6
8	err	s3	err	s9	err		
9	reduce $T \rightarrow (E)$						

**stack**



**input**



**shift 5**

	i	+	(	)	\$	E	T
0	s5	err	s7	err	err	1	6
1	err	s3	err	err	s2		
2	acc						
3	s5	err	s7	err	err		4
4	reduce $E \rightarrow E+T$						
5	reduce $T \rightarrow i$						
6	reduce $E \rightarrow T$						
7	s5	err	s7	err	err	8	6
8	err	s3	err	s9	err		
9	reduce $T \rightarrow (E)$						

**stack**

5 (i)
7 (()
7 (()
0 (\$)

54

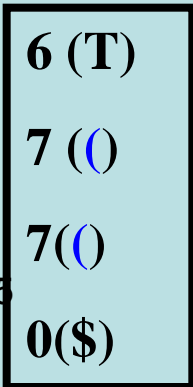
**input**

) \$
------

**reduce  $T \rightarrow i$**

	i	+	(	)	\$	E	T
0	s5	err	s7	err	err	1	6
1	err	s3	err	err	s2		
2	acc						
3	s5	err	s7	err	err		4
4	reduce $E \rightarrow E+T$						
5	reduce $T \rightarrow i$						
6	reduce $E \rightarrow T$						
7	s5	err	s7	err	err	8	6
8	err	s3	err	s9	err		
9	reduce $T \rightarrow (E)$						

**stack**



53

**input**



**reduce  $E \rightarrow T$**

	i	+	(	)	\$	E	T
0	s5	err	s7	err	err	1	6
1	err	s3	err	err	s2		
2	acc						
3	s5	err	s7	err	err		4
4	reduce $E \rightarrow E+T$						
5	reduce $T \rightarrow i$						
6	reduce $E \rightarrow T$						
7	s5	err	s7	err	err	8	6
8	err	s3	err	s9	err		
9	reduce $T \rightarrow (E)$						

**stack**

8 (E)  
7 (()  
7 (()  
0 (\$)

56

**input**

) \$

**shift 9**



	i	+	(	)	\$	E	T
0	s5	err	s7	err	err	1	6
1	err	s3	err	err	s2		
2	acc						
3	s5	err	s7	err	err		4
4	reduce $E \rightarrow E+T$						
5	reduce $T \rightarrow i$						
6	reduce $E \rightarrow T$						
7	s5	err	s7	err	err	8	6
8	err	s3	err	s9	err		
9	reduce $T \rightarrow (E)$						

**stack**

9 ( )

8 (E)

7 ( )

7 ( )

0 (\$)

**input**

\$

**reduce  $T \rightarrow (E)$**

	i	+	(	)	\$	E	T
0	s5	err	s7	err	err	1	6
1	err	s3	err	err	s2		
2	acc						
3	s5	err	s7	err	err		4
4	reduce $E \rightarrow E+T$						
5	reduce $T \rightarrow i$						
6	reduce $E \rightarrow T$						
7	s5	err	s7	err	err	8	6
8	err	s3	err	s9	err		
9	reduce $T \rightarrow (E)$						

**stack**

6 (T)  
7()  
0(\$)

**input**

\$

**reduce  $E \rightarrow T$**

	i	+	(	)	\$	E	T
0	s5	err	s7	err	err	1	6
1	err	s3	err	err	s2		
2	acc						
3	s5	err	s7	err	err		4
4	reduce $E \rightarrow E+T$						
5	reduce $T \rightarrow i$						
6	reduce $E \rightarrow T$						
7	s5	err	s7	err	err	8	6
8	err	s3	err	s9	err		
9	reduce $T \rightarrow (E)$						

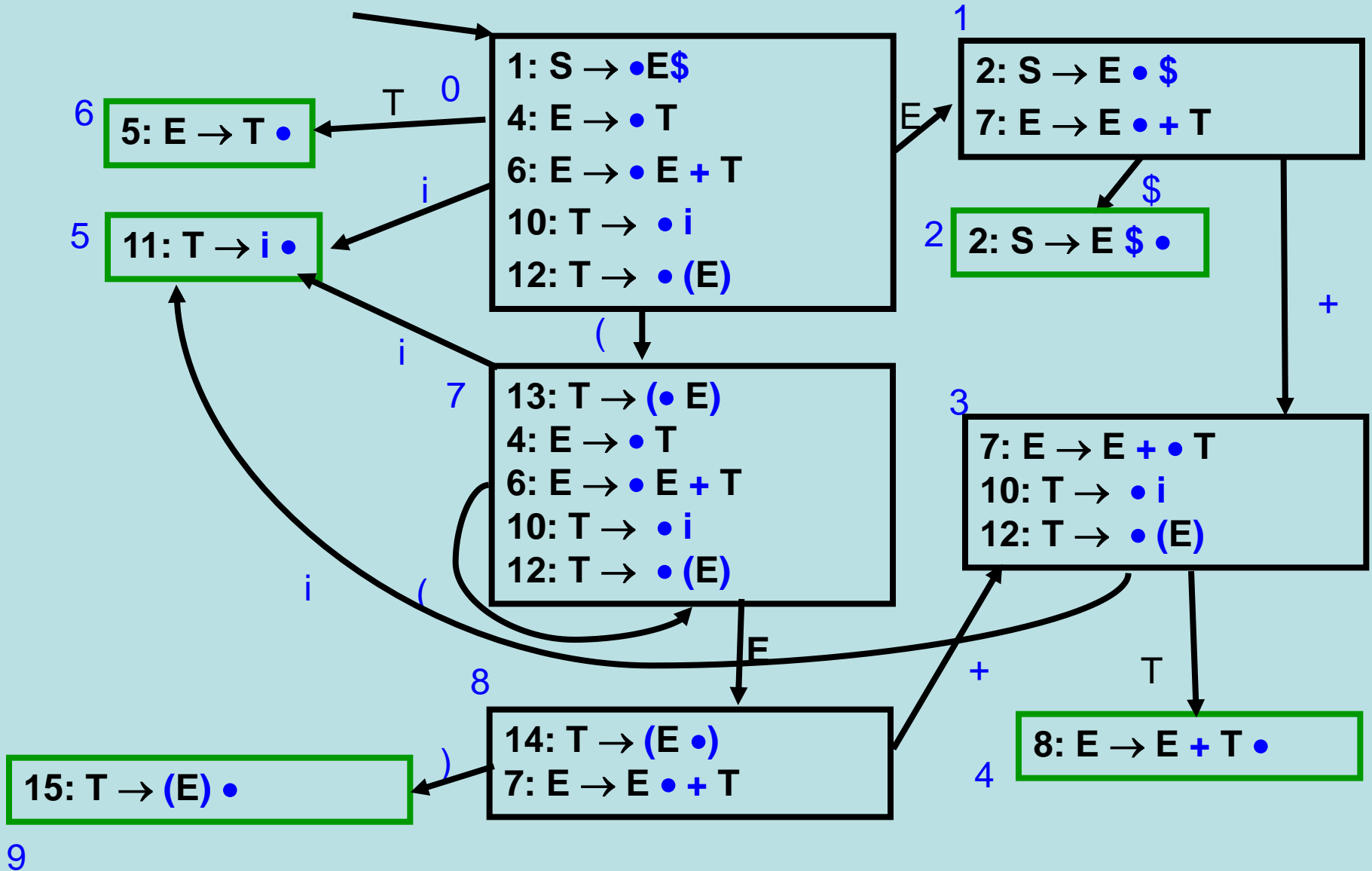
**stack**

8 (E)
7()
0(\$)

**input**

\$
----

**err**



# Constructing LR(0) parsing table

- Add a production  $S' \rightarrow S\$$
- Construct a deterministic finite automaton accepting “valid stack symbols”
- States are set of items  $A \rightarrow \alpha \bullet \beta$ 
  - The states of the automaton becomes the states of parsing-table
  - Determine **shift** operations
  - Determine **goto** operations
  - Determine **reduce** operations

# Filling Parsing Table

- A state  $s_i$
- reduce  $A \rightarrow \alpha$ 
  - $A \rightarrow \alpha \bullet \in s_i$
- Shift on  $t$ 
  - $A \rightarrow \alpha \bullet t \beta \in s_i$
- $\text{Goto}(s_i, X) = s_j$ 
  - $A \rightarrow \alpha \bullet X \beta \in s_i$
  - $\delta(s_i, X) = s_j$
- When conflicts occurs the grammar is not LR(0)

# Example Control Table

	i	+	(	)	\$	E	T
0	s5	err	s7	err	err	1	6
1	err	s3	err	err	s2		
2	acc						
3	s5	err	s7	err	err		4
4	reduce $E \rightarrow E + T$						
5	reduce $T \rightarrow i$						
6	reduce $E \rightarrow T$						
7	s5	err	s7	err	err	8	6
8	err	s3	err	s9	err		
63 9	reduce $T \rightarrow (E)$						

# Example Non LR(0) Grammar

$S \rightarrow E\$$

$E \rightarrow E+E$

$E \rightarrow i$

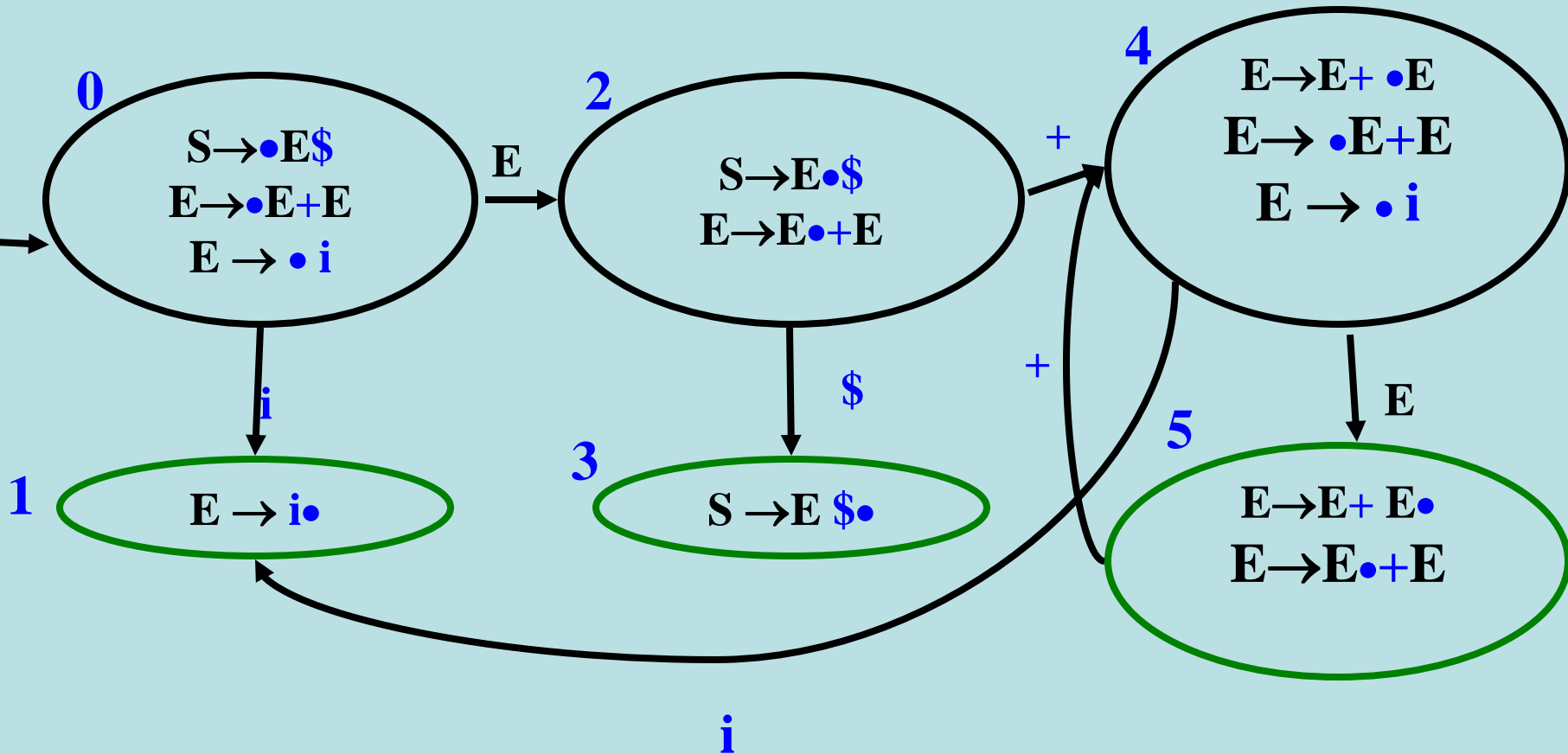
LR(0) items	i	+	\$	E	$\epsilon$
1: $S \rightarrow \bullet E \$$				2	4, 8
2: $S \rightarrow E \bullet \$$			s3		
3: $S \rightarrow E \$ \bullet$					r $S \rightarrow E \$$
4: $E \rightarrow \bullet E + E$				5	4, 8
5: $E \rightarrow E \bullet + E$		s6			
6: $E \rightarrow E + \bullet E$				7	
7: $E \rightarrow E + E \bullet$					r $E \rightarrow E + E$
8: $E \rightarrow \bullet i$	s9				
9: $E \rightarrow i \bullet$					r $E \rightarrow i$

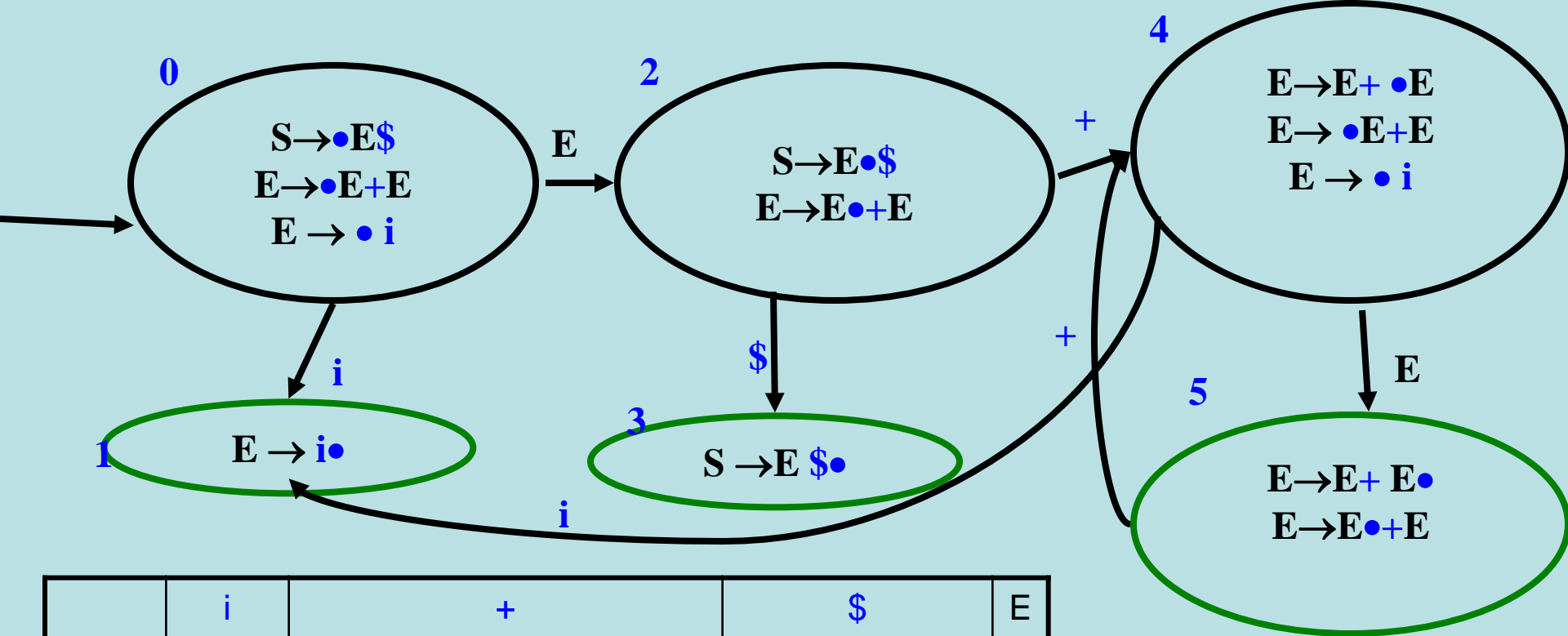


# Example Non LR(0)

## DFA

$S \rightarrow E \$$     $E \rightarrow E + E \mid i$





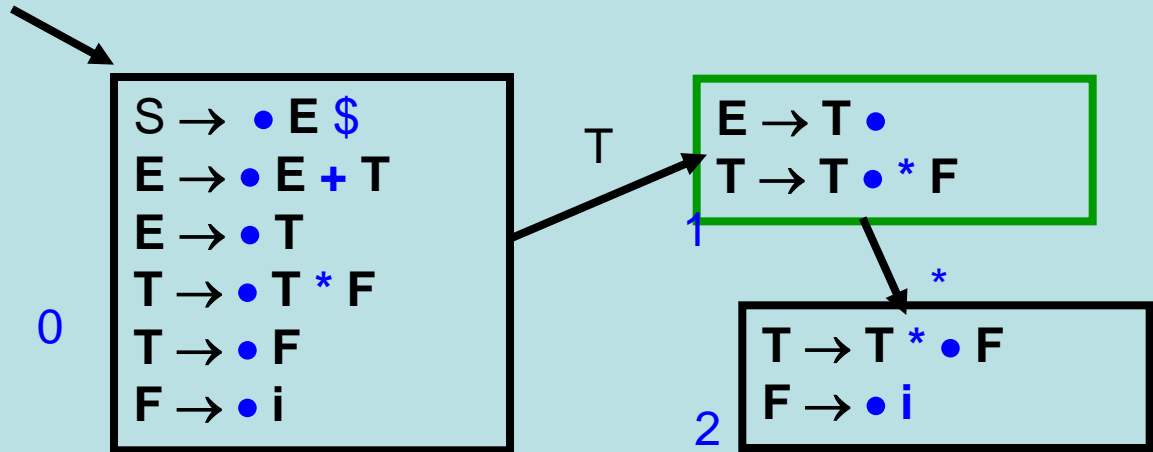
	i	+	\$	E
0	s1	err	err	2
1	red $E \rightarrow i$			
2	err	s4	s3	
3	accept			
4	s1			5
5	red $E \rightarrow E + E$	s4 red $E \rightarrow E + E$	red $E \rightarrow E + E$	

# Dangling Else

S → if cond s else s  
| if cond s  
| assign

# Non-Ambiguous Non LR(0) Grammar

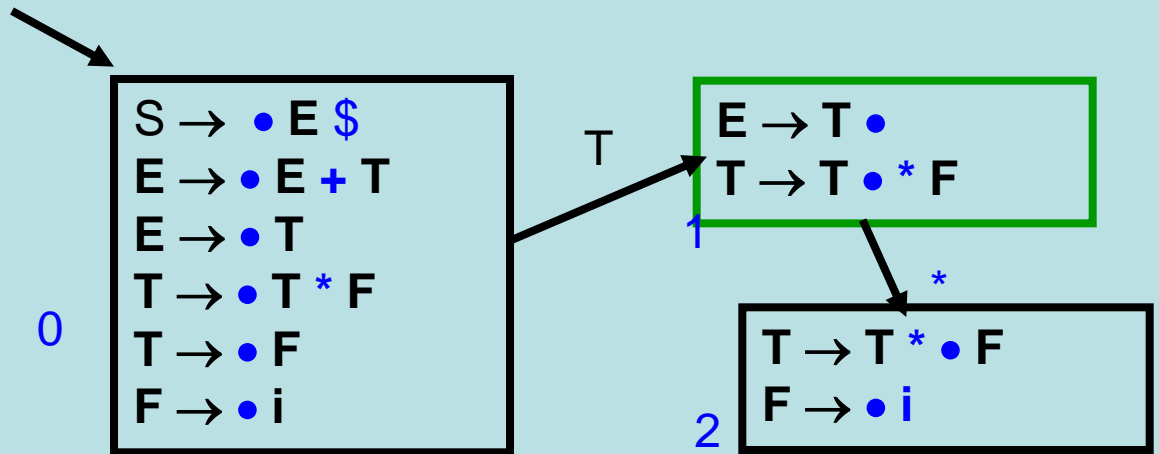
$S \rightarrow E \$$   
 $E \rightarrow E + T \mid T$   
 $T \rightarrow T * F \mid F$   
 $F \rightarrow i$



	i	+	*	
0				
1		?	?	
2				

# Non-Ambiguous SLR(1) Grammar

$S \rightarrow E \$$   
 $E \rightarrow E + T \mid T$   
 $T \rightarrow T * F \mid F$   
 $F \rightarrow i$

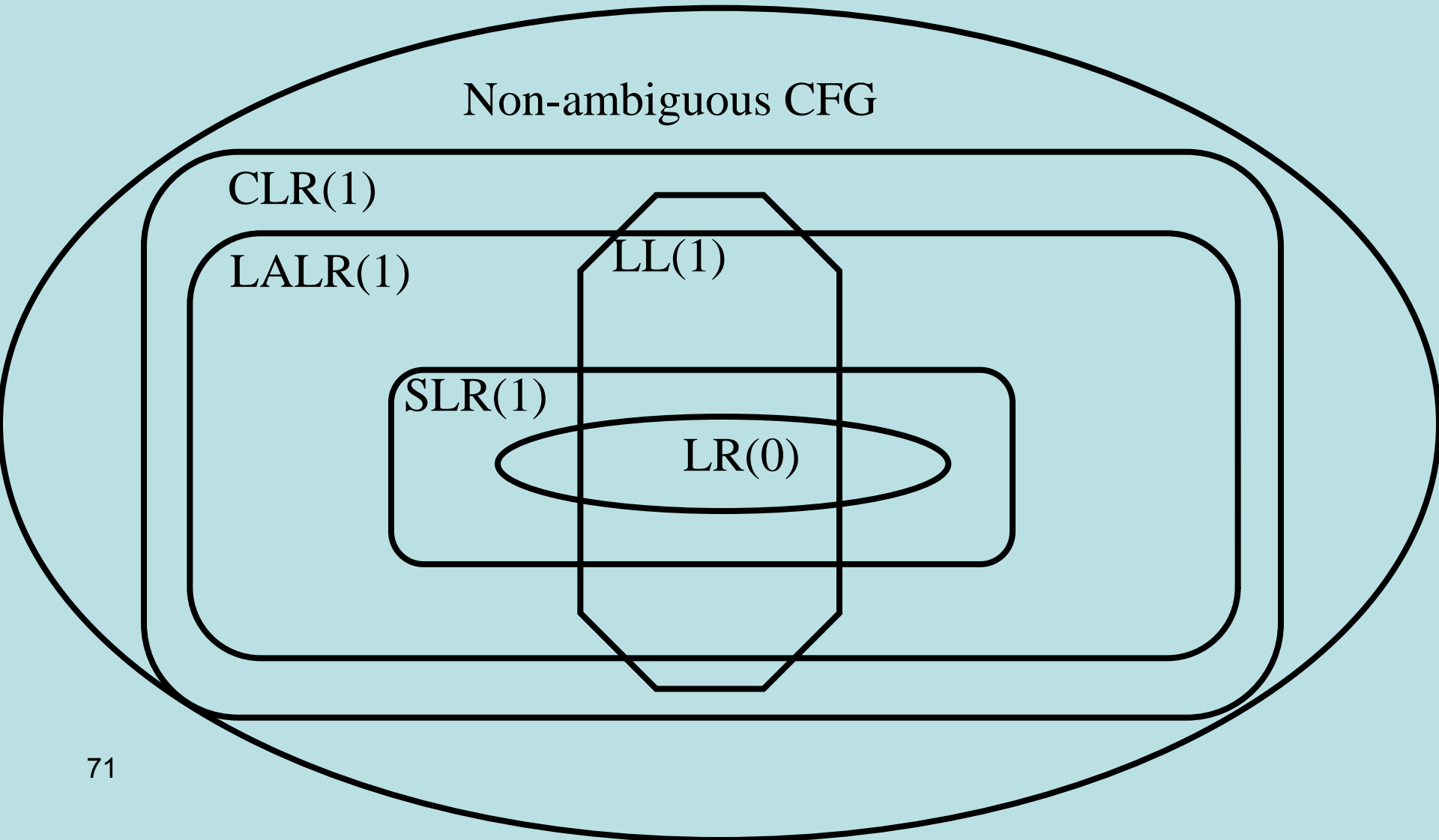


	i	+	*	
0				
1		r $E \rightarrow T$	s2	
2				

# LR(1) Parser

- LR(1) Items  $A \rightarrow \alpha \bullet \beta, t$ 
  - $\alpha$  is at the top of the stack and we are expecting  $\beta t$
- LR(1) State
  - Sets of items
- LALR(1) State
  - Merge items with the same look-ahead

# Grammar Hierarchy



# Interesting Non LR(1) Grammars

- Ambiguous
  - Arithmetic expressions
  - Dangling-else
- Common derived prefix
  - $A \rightarrow B_1 a b \mid B_2 a c$
  - $B_1 \rightarrow \varepsilon$
  - $B_2 \rightarrow \varepsilon$
- Optional non-terminals
  - $St \rightarrow OptLab Ass$
  - $OptLab \rightarrow id : \mid \varepsilon$
  - $Ass \rightarrow id := Exp$



# A motivating example

- Create a desk calculator
- Challenges
  - Non trivial syntax
  - Recursive expressions (semantics)
    - Operator precedence

# Solution (lexical analysis)

```
import java_cup.runtime.*;
%%
%cup
%eofval{
    return sym.EOF;
%eofval}
NUMBER=[0-9]+
%%
"+" { return new Symbol(sym.PLUS); }
"-" { return new Symbol(sym.MINUS); }
"*" { return new Symbol(sym.MULT); }
"/" { return new Symbol(sym.DIV); }
"(" { return new Symbol(sym.LPAREN); }
")" { return new Symbol(sym.RPAREN); }
{NUMBER} {
    return new Symbol(sym.NUMBER, new Integer(yytext()));
}
\n { }
. { }
```

- 74 Parser gets terminals from the Lexer

```

terminal Integer NUMBER;
terminal PLUS,MINUS,MULT,DIV;
terminal LPAREN, RPAREN;
terminal UMINUS;
nonterminal Integer expr;
precedence left PLUS, MINUS;
precedence left DIV, MULT;
Precedence left UMINUS;
%%
expr ::= expr:e1 PLUS expr:e2
      | expr:e1 MINUS expr:e2
      | expr:e1 MULT expr:e2
      | expr:e1 DIV expr:e2
      | MINUS expr:e1 %prec UMINUS
      | LPAREN expr:e1 RPAREN
      | NUMBER:n
      | RESULT = new Integer(e1.intValue() + e2.intValue()); :}
      | RESULT = new Integer(e1.intValue() - e2.intValue()); :}
      | RESULT = new Integer(e1.intValue() * e2.intValue()); :}
      | RESULT = new Integer(e1.intValue() / e2.intValue()); :}
      | RESULT = new Integer(0 - e1.intValue()); :}
      | RESULT = e1; :}
      | RESULT = n; :}

```

# Summary

- LR is a powerful technique
- Generates efficient parsers
- Generation tools exist LALR(1)
  - Bison, yacc, CUP
- But some grammars need to be tuned
  - Shift/Reduce conflicts
  - Reduce/Reduce conflicts
  - Efficiency of the generated parser
- There exist more general methods
  - GLR
  - Arbitrary grammars in  $n^3$ 
    - Early parsers
    - CYK algorithms