SYNTAX ANALYSIS

- SLR PARSER

Introduction To LR Parser

- Most prevalent type of Bottom-Up Parser
- Known as LR (k) parser
  - L stands for Left to Right scanning of input
  - R stands for Rightmost Derivation in reverse
  - k used for number of input symbols of look ahead for making parsing decisions
- k = 0 or k = 1 is used for practical interest.
- When k is omitted then it is considered as 1
- Examples: SLR, Canonical LR and LALR

Why LR Parser??

- Table driven similar to Non recursive LL parser
- They can be constructed to recognize virtually all programming language construct for which context free grammar can be written
- Parsing method is the most General non-backtracking Shift Reduce
- Parsing Method can be implemented efficiently
- Can detect syntactic error at earliest from left to right scan
- The class of grammar that can be parsed using LR methods is the proper SUPERSET of the class of the grammar that can be parsed using LL Method

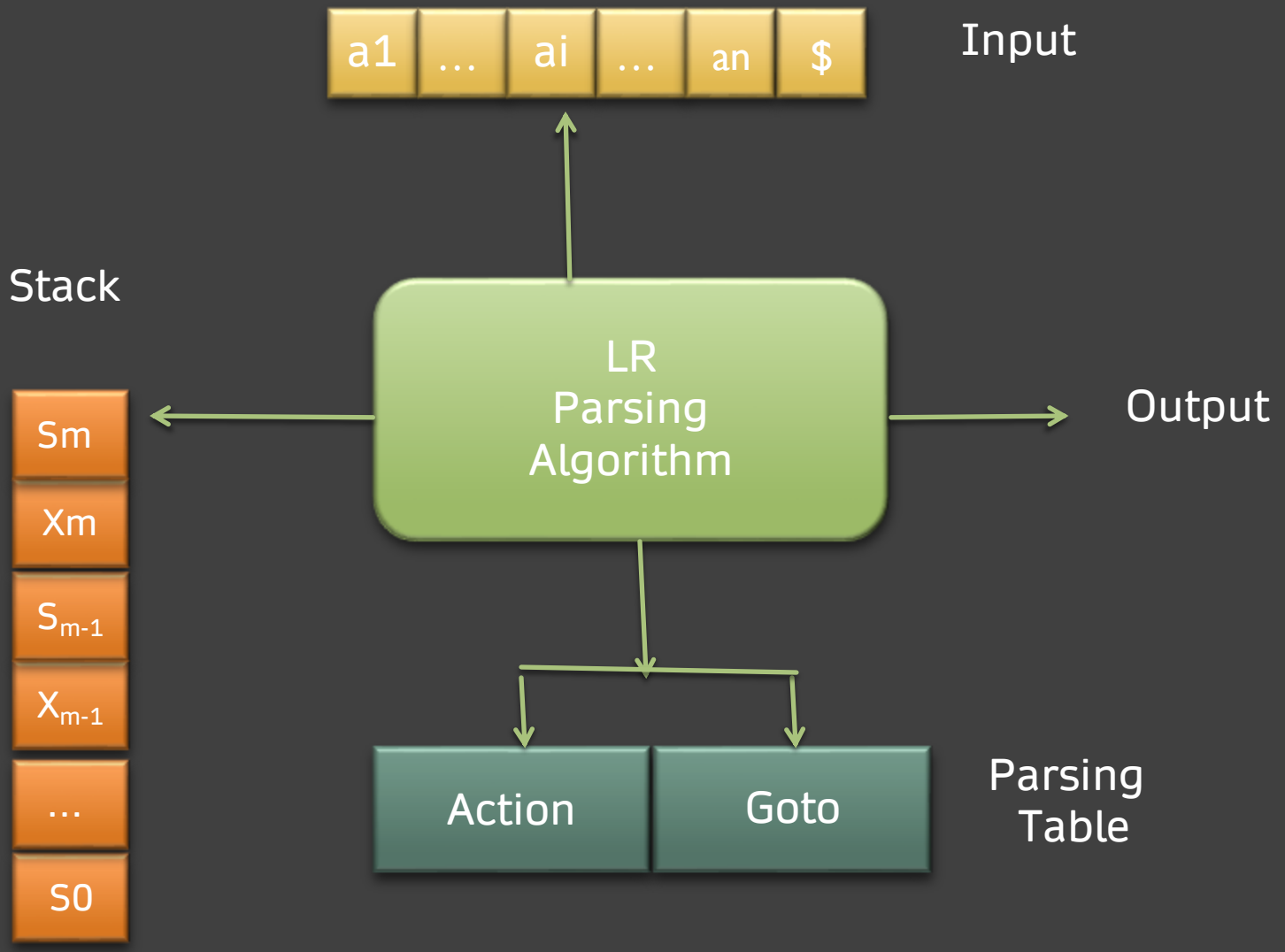
Why LR Parser??

Drawback:

- Too much work to construct LR Parser by hand for typical programming-language grammar

Solution: LR Parser generator is needed.

Example: YACC tool



On Stack:

Each Xi – Grammar Symbol

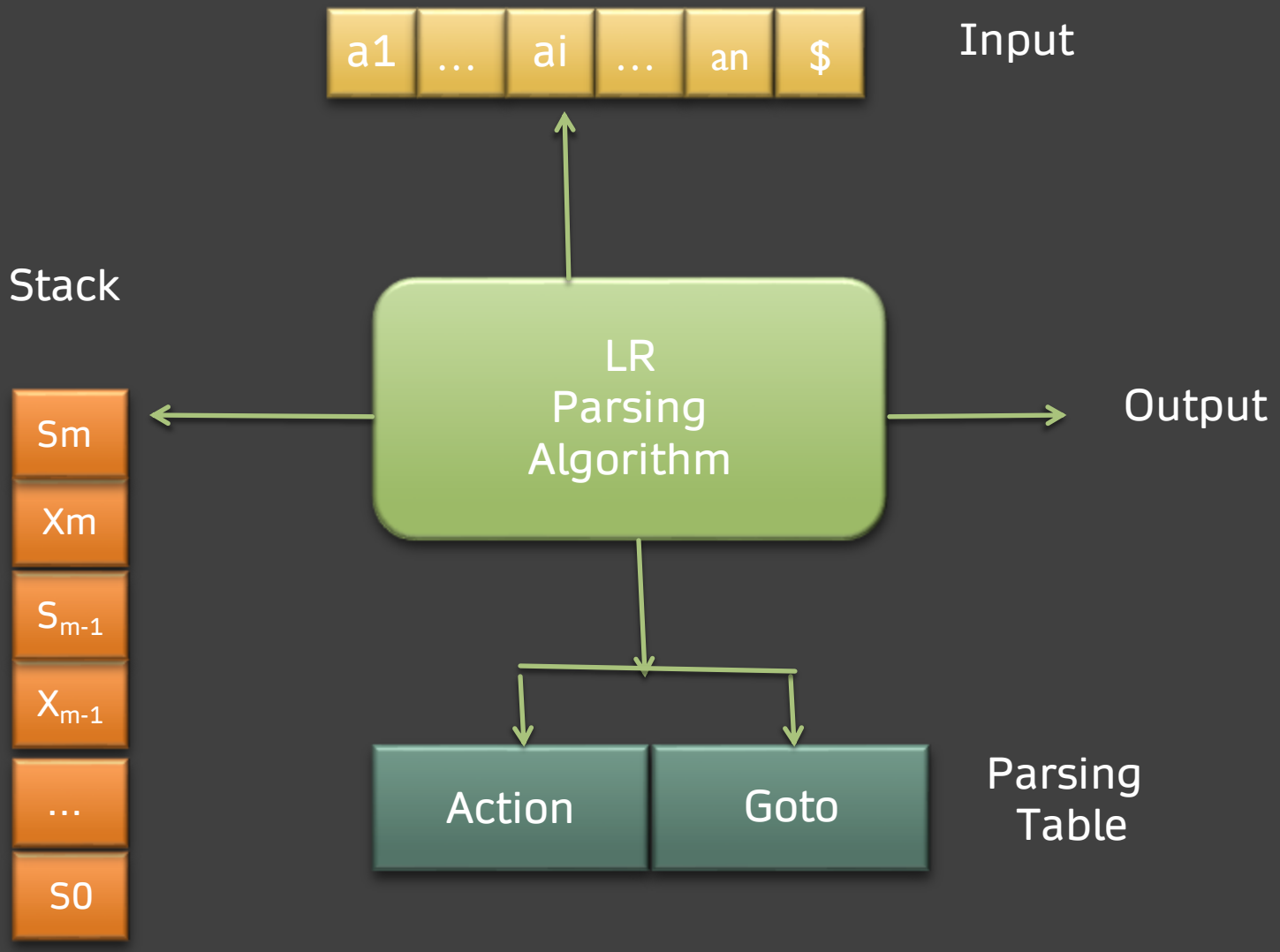
Si – State

The parsing Table consists of two parts

1. Action
2. Goto

MODEL OF SLR PARSER

MODEL OF SLR PARSER



The program reads [Sm , ai]

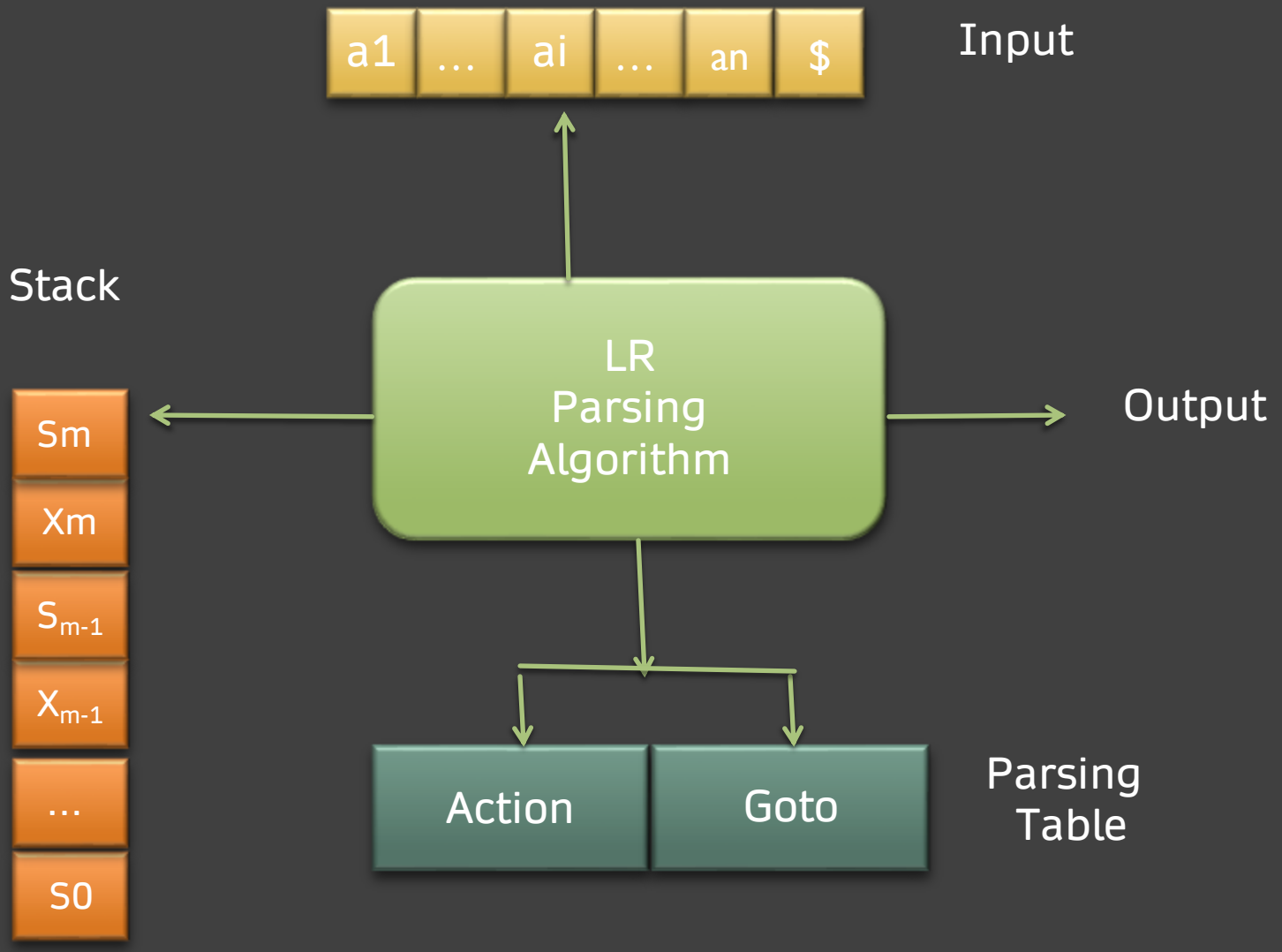
Where Sm : Top of the stack

ai : Current input symbol

Four Possible Actions are:

1. Shift S, where S is a state
2. Reduce by a production A → b
3. Accept
4. Error

MODEL OF SLR PARSER



The function Goto takes a state and grammar symbol as arguments and produce states

Construction Of SLR Parsing Table

Central Idea

To construct a DFA from the given grammar to recognize viable prefixes

Construction of LR(0) Items

LR (0) item of a grammar G is a production of G with a dot (.) at some position on the right side.

Example:

If A → XYZ then there are four possible LR(0) items as:

A → . XYZ

A → X . YZ

A → XY . Z

A → XYZ .

Augmented Grammar

Given: If a grammar G is with start symbol S

then G' is augmented Grammar for G

Two elements are added in G to get G'

1. New start symbol G'

2. New production $S' \rightarrow S$

Acceptance of string is announced only when parser is about to reduce $S' \rightarrow S$

Closure Operation

If I is a set of items for a grammar G then

The closure (I) is the set of items constructed from I by the two rules as:

1. Initially every item in I is added in closure (I)
2. If $A \rightarrow \alpha . B \beta$ is in closure (I) and $B \rightarrow r$ then add the item $B \rightarrow . r$ to closure(I) if it is not already there.

   Apply the rule until no more rules can be added

Closure Operation

Example:

Given Grammar as

E' → E

E → E + T | T

T → T * F | F

F → (E) | id

If I is the set of one item { [E' → . E] } then closure of I contains

E' → . E E → . E + T E → .T

T → . T * F T → . F F → . (E)

F → . id

GOTO Operation

If GOTO (I, X) where I is the set of items and X is a grammar symbol.

If I contains [A → α . X β] then

GOTO (I, X) - Closure of the set of all items [A → α X . β]

Example:

If I is set of two items as { [E' → E .] , [E → E . + T] }

Then GOTO (I, +) consists of

E → E + . T

T → . T * F

T → . F

F → . (E)

F → . id

Construction of LR (0) Automaton

Grammar

E' → E

E → E + T | T

T → T * F | F

F → (E) | id

State 1: I0

Consider Production with start symbol

E' → . E

Add to I0

E → . E + T

E → . T

T → . T * F

T → . F

F → . (E)

F → . id

Grammar

E' → E

E → E + T | T

T → T * F | F

F → (E) | id

State I0:

E' → . E

E → . E + T

E → . T

T → . T * F

T → . F

F → . (E)

F → . id

GOTO (I0 , E) :

E' → E .

E → E . + T [New State: I1]

GOTO (I0 , T) :

E → T .

T → T . * F [New State: I2]

Grammar

E' → E

E → E + T | T

T → T * F | F

F → (E) | id

State I0:

E' → . E

E → . E + T

E → . T

T → . T * F

T → . F

F → . (E)

F → . id

GOTO (I0 , F) :

T → F . [New State: I3]

GOTO (I0 , ( ) :

F → (. E)

E → . E + T

E → . T

T → . T * F

T → . F

F → . (E)

F → . id [New State: I4]

Grammar

E' → E

E → E + T | T

T → T * F | F

F → (E) | id

State I0:

E' → . E

E → . E + T

E → . T

T → . T * F

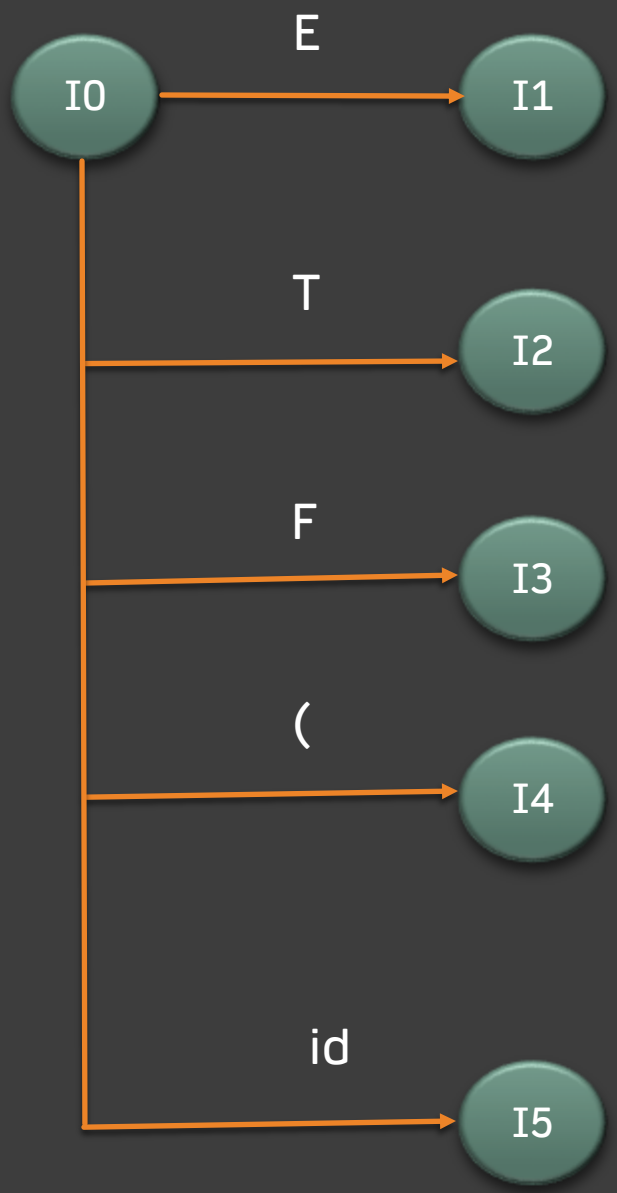
T → . F

F → . (E)

F → . id

GOTO (I0 , id) :

F → id . [New State: I5]


E
I0
I1
T
I2
F
I3
(
I4
id
I5

Grammar

E' → E

E → E + T | T

T → T * F | F

F → (E) | id

State I1:

E' → E .

E → E . + T

GOTO (I1 , +) :

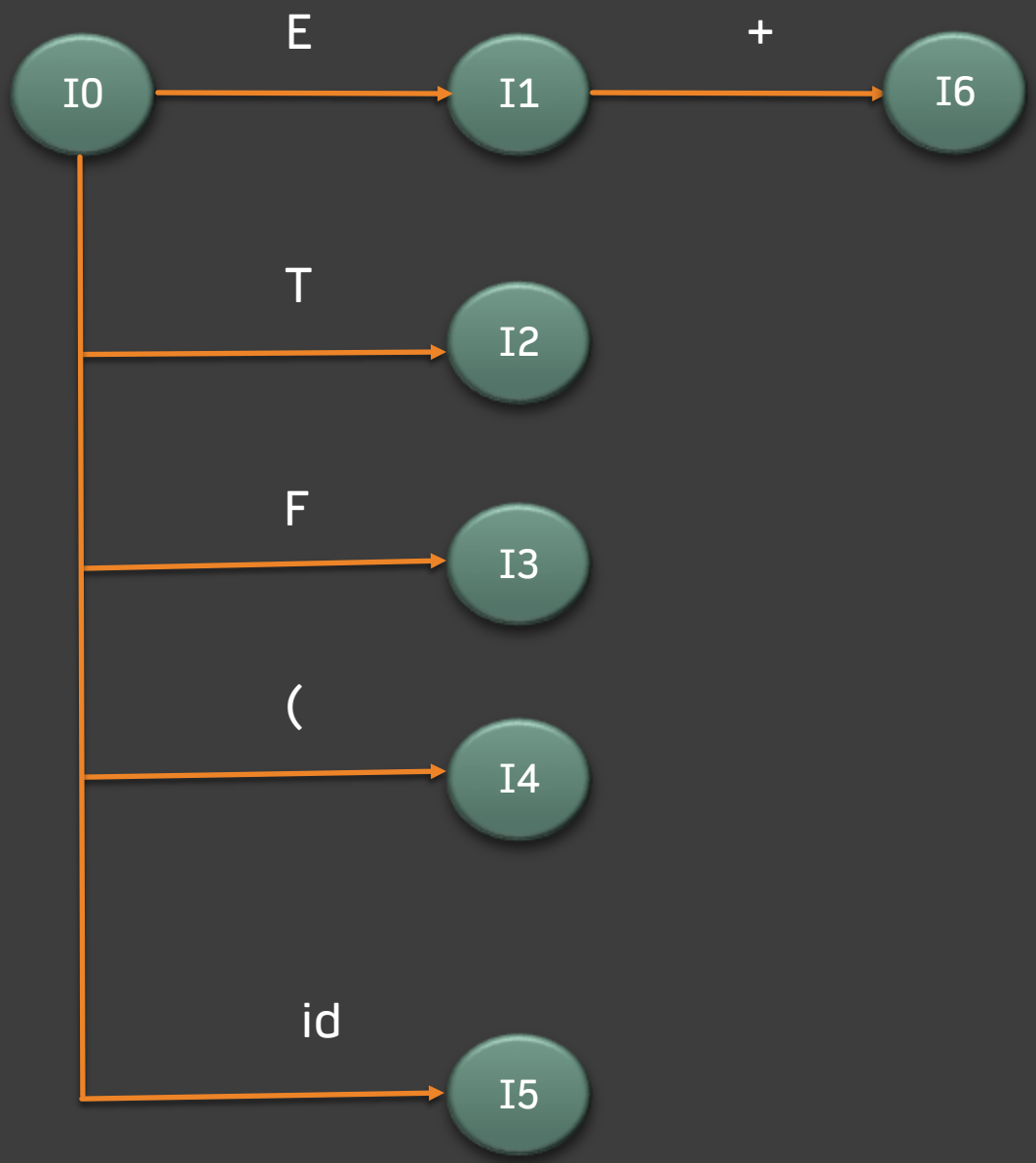
E → E + . T

T → . T * F

T → . F

F → . (E)

F → . id [New State: I6]


E
+
I0
I1
I6
T
I2
F
I3
(
I4
id
I5

Grammar

E' → E

E → E + T | T

T → T * F | F

F → (E) | id

State I2:

E → T .

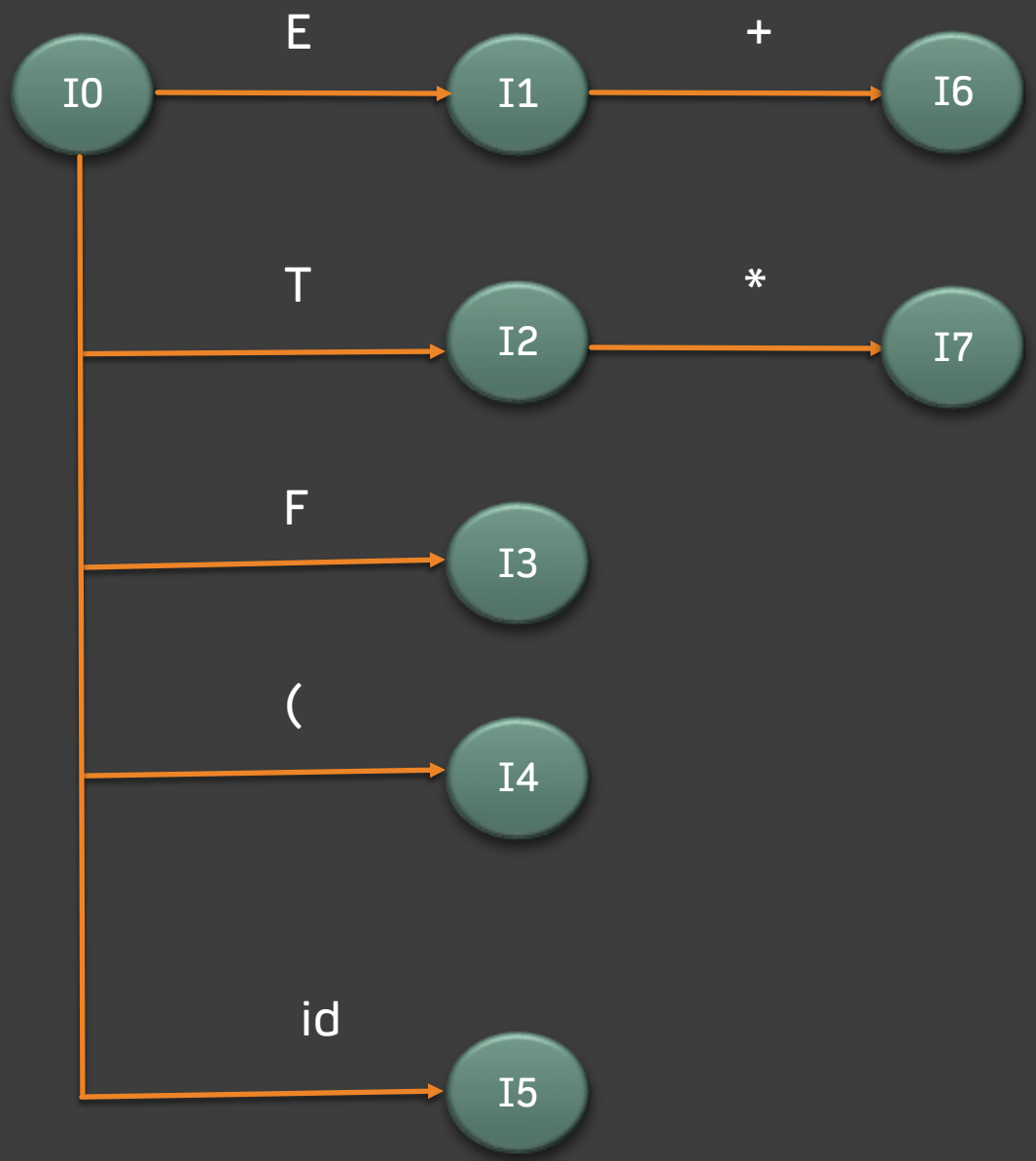
T → T . * F

GOTO (I2 , *) :

T → T * . F

F → . (E)

F → . id [New State: I7]


I0
E
I1
+
I6
T
I2
*
I7
F
I3
(
I4
id
I5

Grammar

E' → E

E → E + T | T

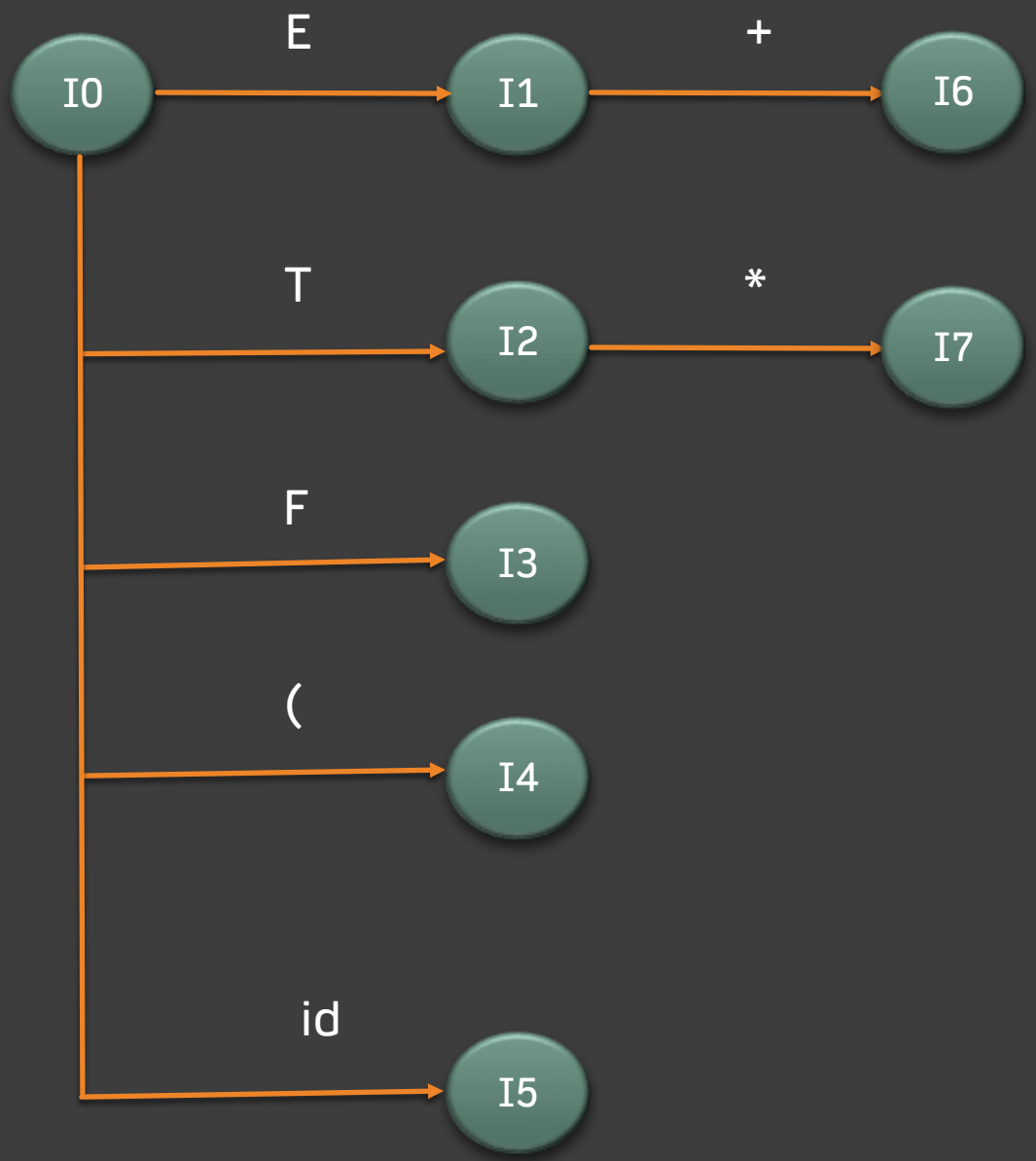
T → T * F | F

F → (E) | id

State I3:

T → F .

No possible GOTO operation


I0
E
I1
+
I6
T
I2
*
I7
F
I3
(
I4
id
I5

Grammar

E' → E

E → E + T | T

T → T * F | F

F → (E) | id

State I4:

F → (. E)

E → . E + T

E → . T

T → . T * F

T → . F

F → . (E)

F → . id

GOTO (I4 , E) :

F → (E .)

E → E . + T [New State: I8]

GOTO (I4 , T) :

E → T .

T → T . * F [Existing State: I2]

State I2:

E → T .

T → T . * F

GOTO (I4 , F) :

T → F . [Existing State: I3]

State I3:

T → F .

Grammar

E' → E

E → E + T | T

T → T * F | F

F → (E) | id

State I4:

F → (. E)

E → . E + T

E → . T

T → . T * F

T → . F

F → . (E)

F → . id

GOTO (I4 , ( ) :

F → (. E)

E → . E + T

E → . T

T → . T * F

T → . F

F → . (E)

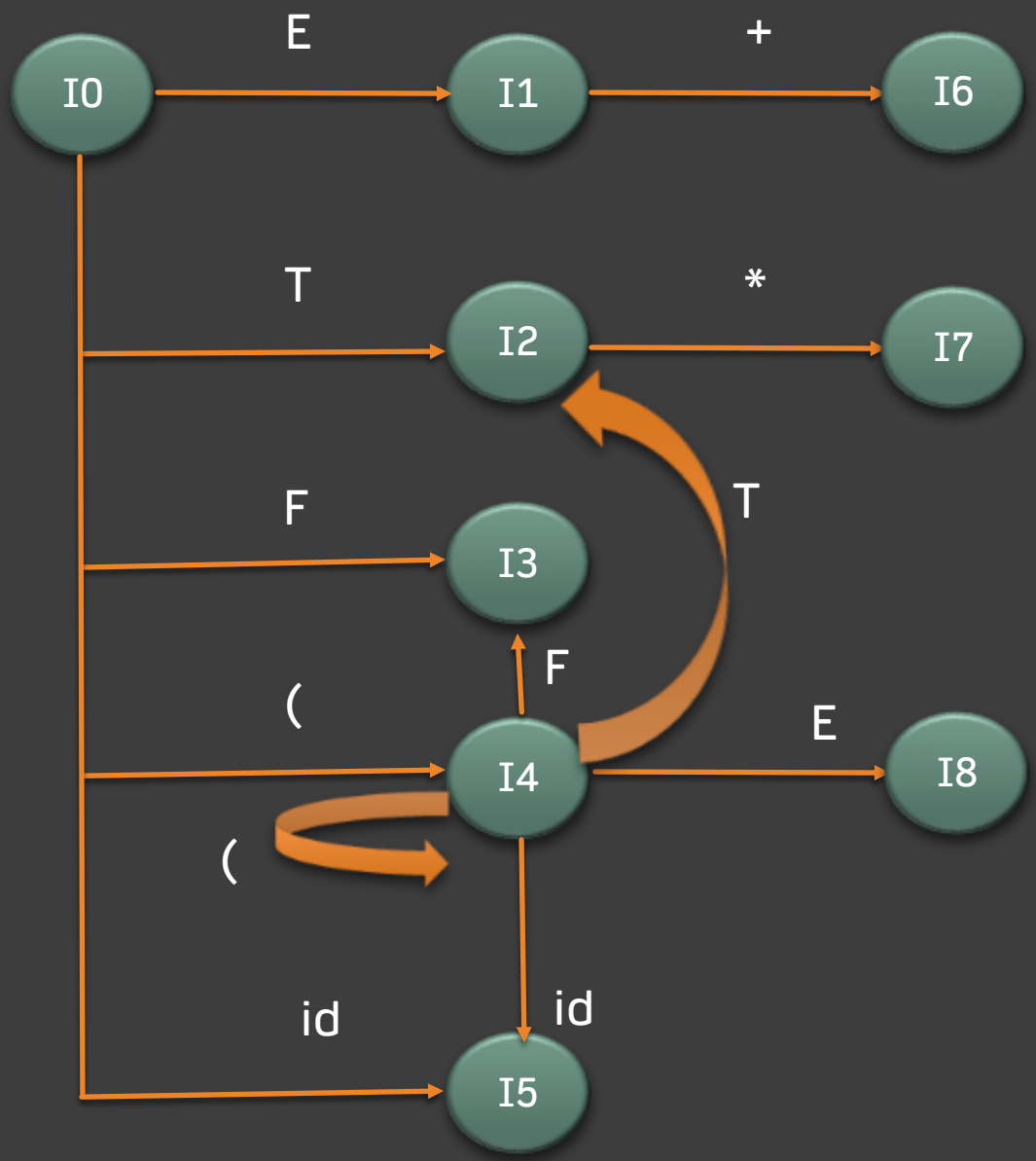
F → . Id [Existing State: I4]

GOTO (I4 , id) :

F → id . [Existing State: I5]

State I5:

F → id .


I0
E
I1
+
I6
T
I2
*
I7
F
I3
T
F
(
I4
E
I8
(
id
id
I5

Grammar

E' → E

E → E + T | T

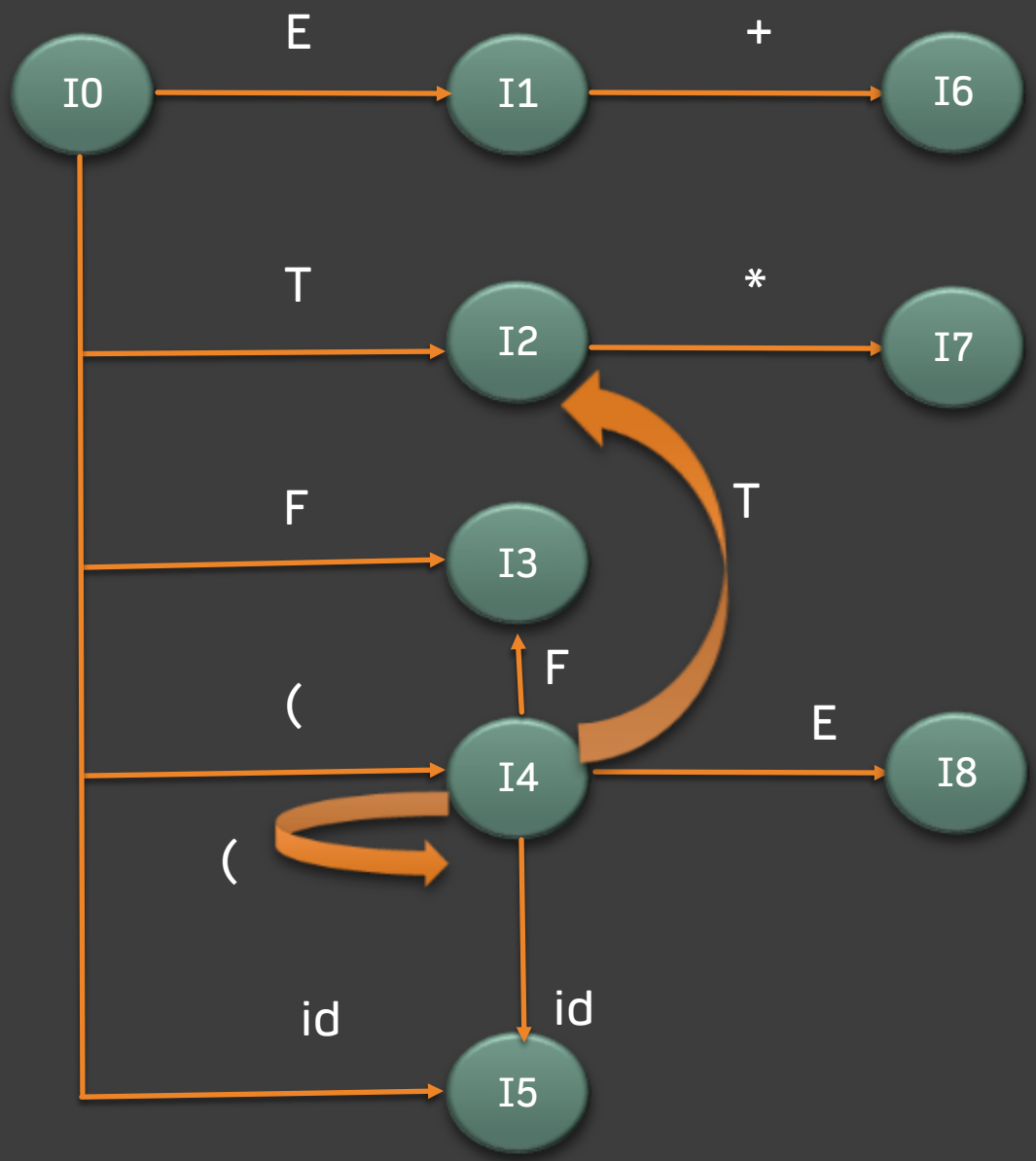
T → T * F | F

F → (E) | id

No possible GOTO operation

State I5:

F → id .


I0
E
I1
+
I6
T
I2
*
I7
F
I3
T
F
(
I4
E
I8
(
id
id
I5

Grammar

E' → E

E → E + T | T

T → T * F | F

F → (E) | id

State I6:

E → E + . T

T → . T * F

T → . F

F → . (E)

F → . id

GOTO (I6 , T) :

E → E + T .

T → T . * F [New State: I9]

GOTO (I6 , F) :

T → F . [Existing State: I3]

State I3:

T → F .

Grammar

E' → E

E → E + T | T

T → T * F | F

F → (E) | id

State I6:

E → E + . T

T → . T * F

T → . F

F → . (E)

F → . id

GOTO (I6 , ( ) :

F → (. E)

E → . E + T

E → . T

T → . T * F

T → . F

F → . (E)

F → . Id [Existing State: I4]

State I4:

F → (. E)

E → . E + T

E → . T

T → . T * F

T → . F

F → . (E)

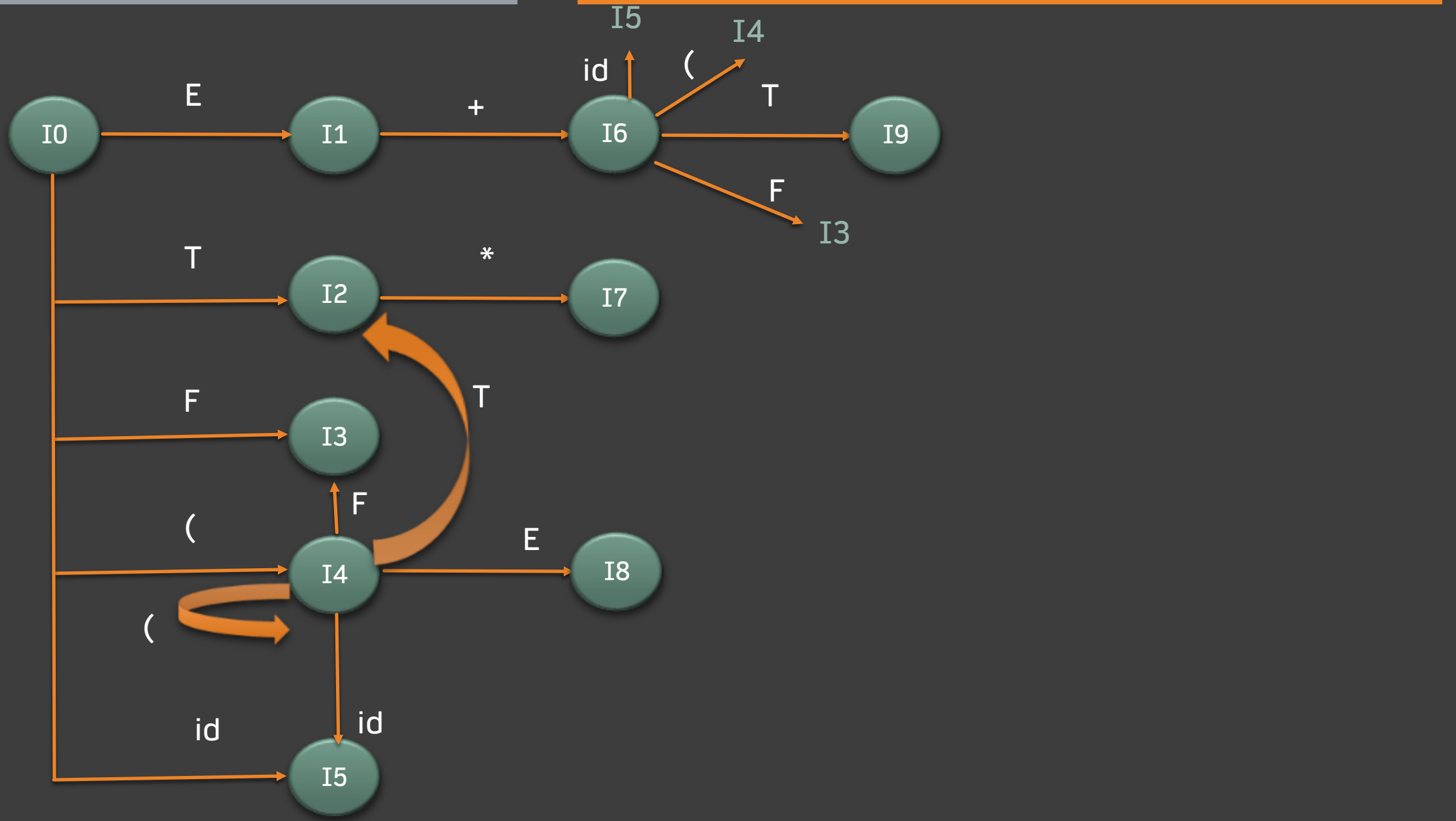
F → . id

GOTO (I6 , id) :

F → id . [Existing State: I5]

State I5:

F → id .


I0
E
I1
+
I6
id
I5
(
I4
T
I9
F
I3
T
I2
*
I7
F
I3
T
F
(
I4
E
I8
(
id
I5
id

Grammar

E' → E

E → E + T | T

T → T * F | F

F → (E) | id

State I7:

T → T * . F

F → . (E)

F → . id

GOTO (I7 , F) :

T → T * F . [New State: I10]

GOTO (I7 , ( ) :

F → (. E)

E → . E + T

E → . T

T → . T * F

T → . F

F → . (E)

F → . Id [Existing State: I4]

GOTO (I7 , id) :

F → id . [Existing State: I5]

State I4:

F → (. E)

E → . E + T

E → . T

T → . T * F

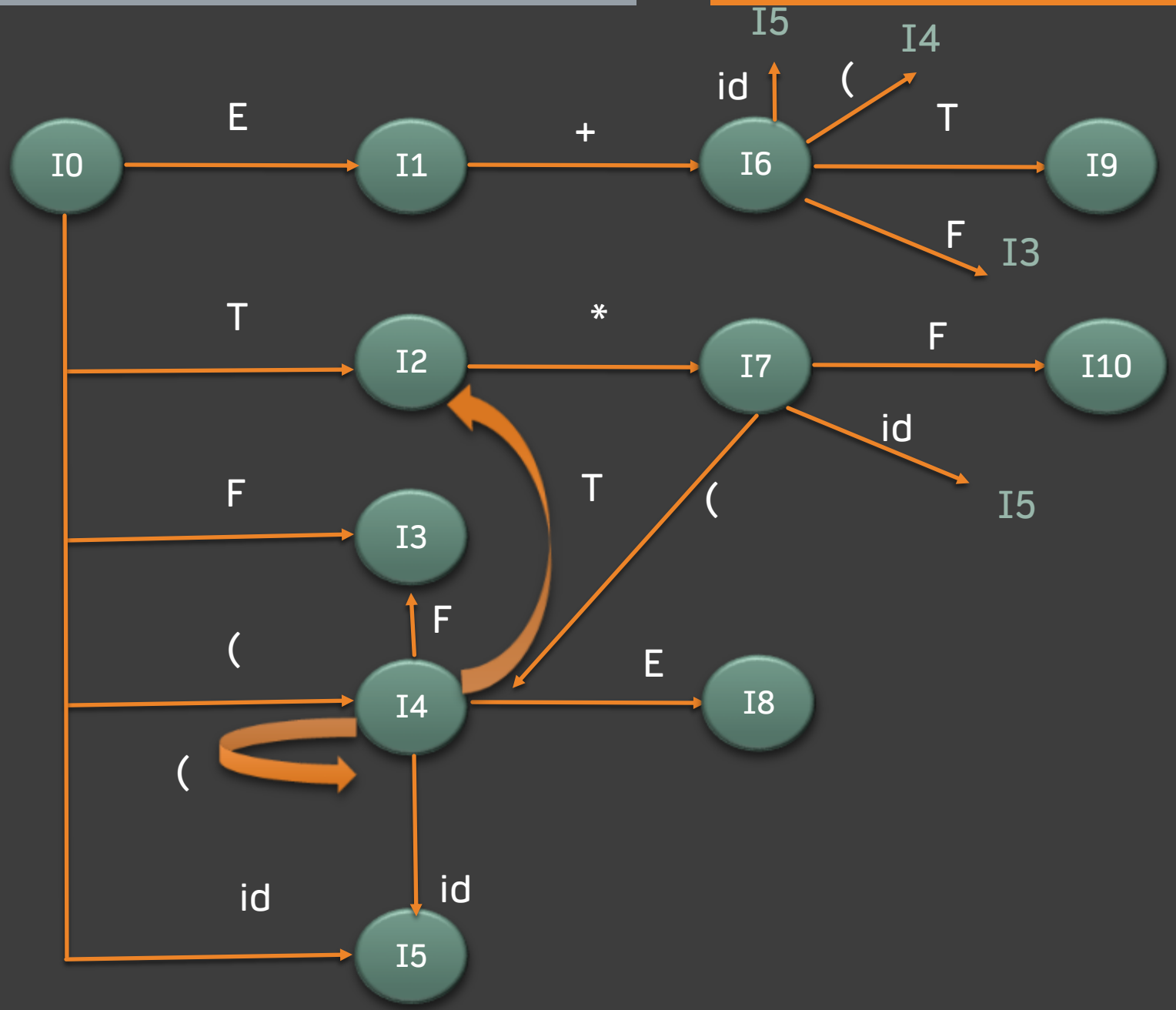
T → . F

F → . (E)

F → . id

State I5:

F → id .


I0
E
I1
+
I6
id
I5
(
I4
T
I9
F
I3
T
I2
*
I7
F
I10
id
I5
F
I3
T
(
F
(
I4
E
I8
(
id
id
I5

Grammar

E' → E

E → E + T | T

T → T * F | F

F → (E) | id

State I8:

F → (E .)

E → E . + T

GOTO (I8 ,)) :

F → (E) . [New State: I11]

GOTO (I8 , +) :

E → E + . T

T → . T * F

T → . F

F → . (E)

F → . id [Existing State: I6]

State I6:

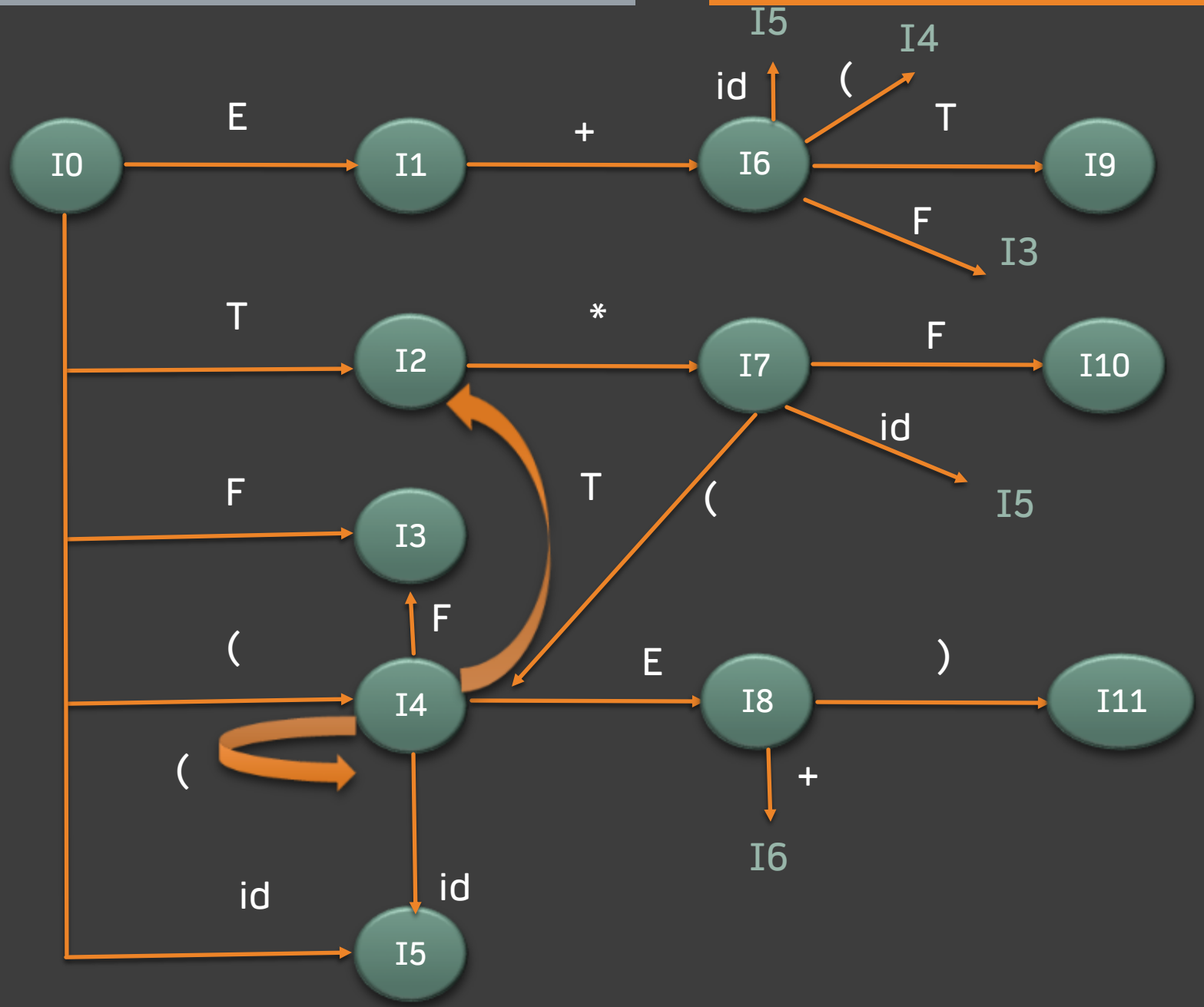
E → E + . T

T → . T * F

T → . F

F → . (E)

F → . id


I0
E
I1
+
I6
id
I5
(
I4
T
I9
F
I3
T
I2
*
I7
F
I10
id
I5
F
I3
T
(
F
(
I4
E
I8
)
I11
(
+
I6
id
id
I5

Grammar

E' → E

E → E + T | T

T → T * F | F

F → (E) | id

State I9:

F → E + T .

T → T . * F

GOTO (I9 , *) :

T → T * . F

F → . (E)

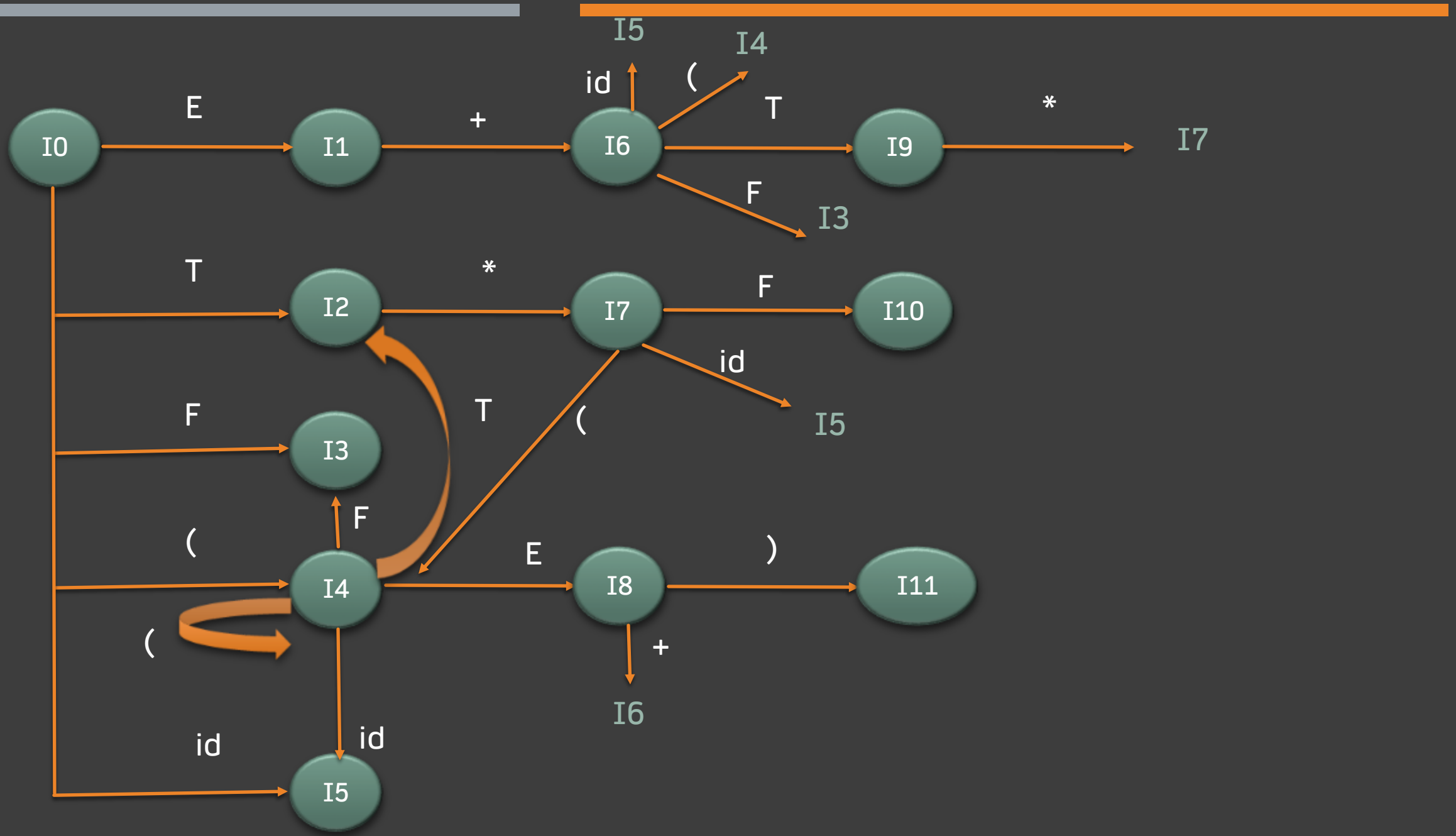
F → . id [Existing State: I7]

State I7:

T → T * . F

F → . (E)

F → . id


I0
E
I1
+
I6
id
I5
(
I4
T
I9
*
I7
F
I3
T
I2
*
I7
F
I10
id
I5
(
T
F
I3
F
(
I4
E
I8
)
I11
(
+
I6
id
id
I5

Grammar

E' → E

E → E + T | T

T → T * F | F

F → (E) | id

State I10:

T → T * F .

No possible GOTO operation

Grammar

E' → E

E → E + T | T

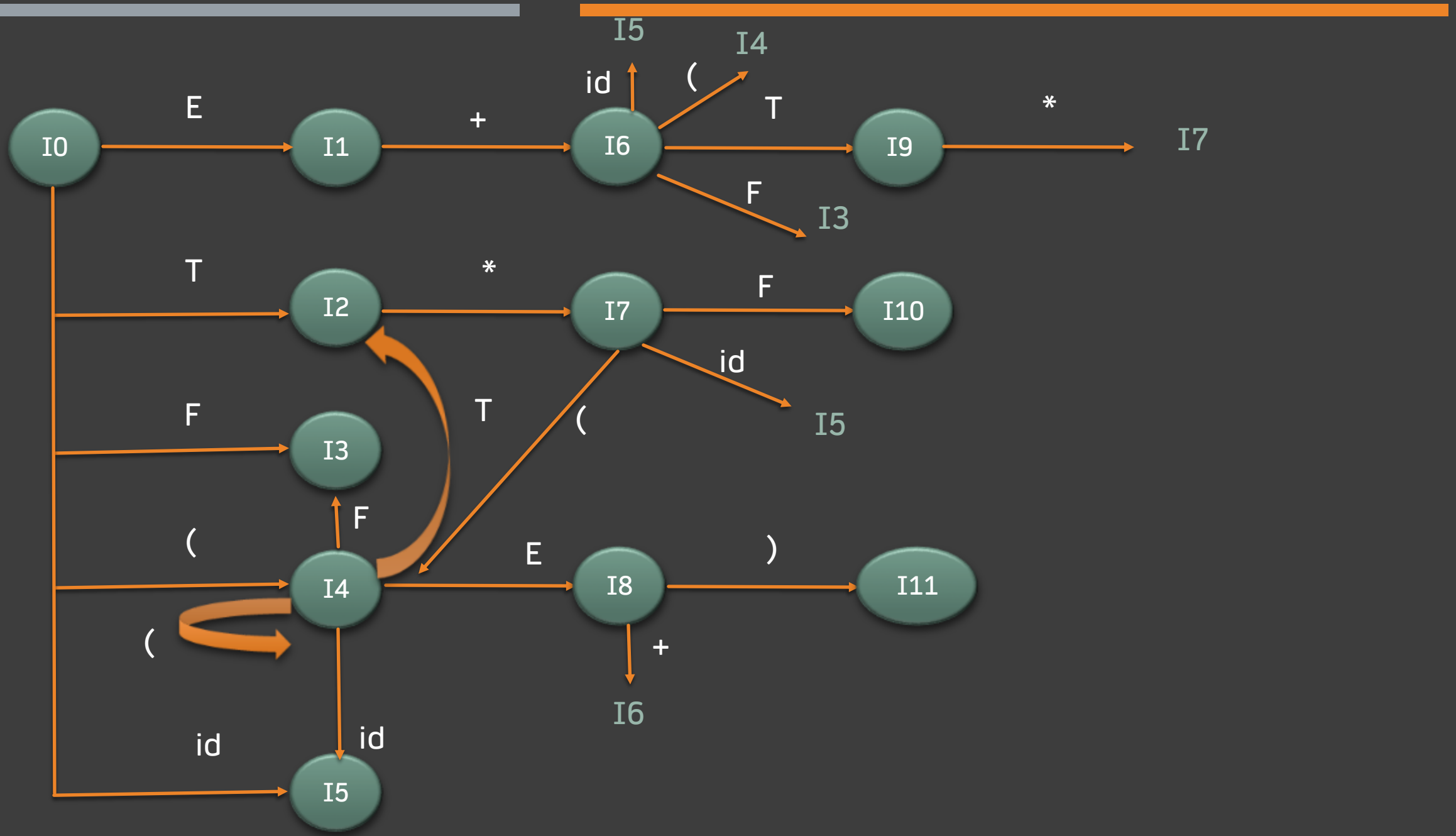
T → T * F | F

F → (E) | id

State I11:

F → (E).

No possible GOTO operation


I0
E
I1
+
I6
id
I5
(
I4
T
I9
*
I7
F
I3
T
I2
*
I7
F
I10
id
I5
(
T
F
I3
F
(
I4
E
I8
)
I11
(
+
I6
id
id
I5

Construction of SLR Parsing Table

Input:

An augmented Grammar G'

Output:

The SLR Parsing Table with functions ACTION and GOTO for G'

Construction of SLR Parsing Table

Method:

1. Construct C = { I0, I1, … In } the collection of sets of LR (0) items for G'

2. State i is constructed from Ii.

   The parsing action for state i are determined as follows:

   a. If [A → α . a β] is in Ii and GOTO (Ii , a) = Ij then
      Set Action [i , a] to **" Shift j "**
      Here a must be terminal

   b. If [A → α .] then
      Set Action [i , a] to " Reduce A → α "
      For all a in FOLLOW (A)

   c. If [S' → S .] is in Ii then
      Set Action [i , $] to " Accept "

If any conflicting actions are generated by the above rules
Then grammar is not SLR (1)

Construction of SLR Parsing Table

Method:

3. The GOTO transitions for state i are constructed for all non – terminals A using the rule

   If GOTO [Ii , A] = Ij then

   Set 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' → . S]

Construction of SLR Parsing Table

Given Grammar:

$E \rightarrow E + T$

$E \rightarrow T$

$T \rightarrow T * F$

$T \rightarrow F$

$F \rightarrow (E)$

$F \rightarrow id$

State	Action						GOTO		
	id	+	*	(	)	$	E	T	F
0									
1									
2									
3									
4									
5									
6									
7									
8									
9									
10									
11									

Construction of SLR Parsing Table

Given Grammar:

E → E + T

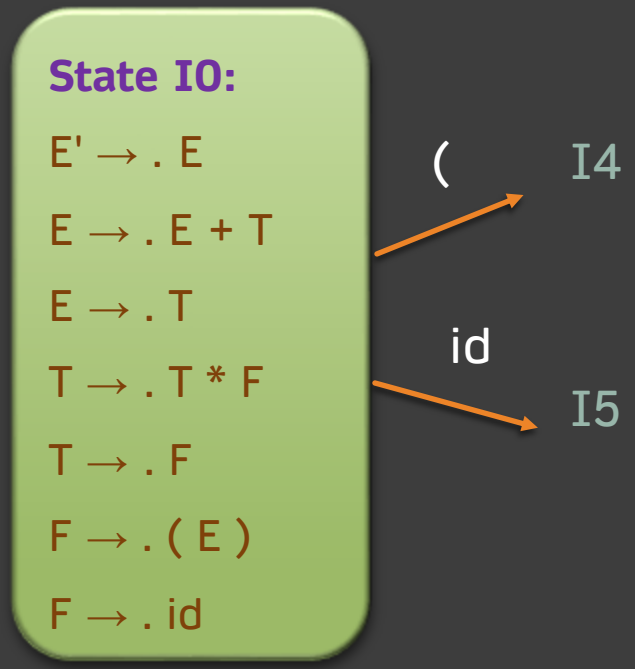
E → T

T → T * F

T → F

F → (E)

F → id



State	Action						GOTO		
	id	+	*	(	)	$	E	T	F
0	s5			s4					
1									
2									
3									
4									
5									
6									
7									
8									
9									
10									
11									

Construction of SLR Parsing Table

Given Grammar:

E → E + T

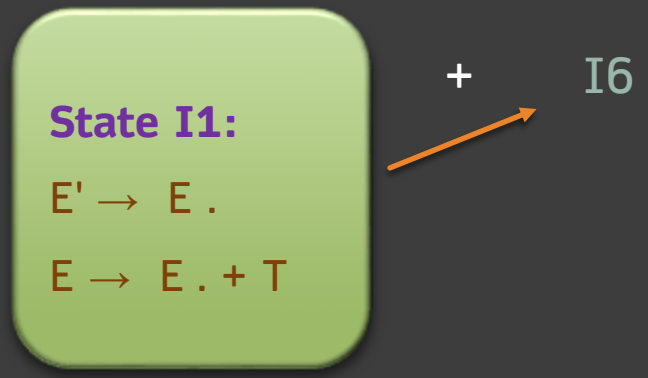
E → T

T → T * F

T → F

F → (E)

F → id



State	Action						GOTO		
	id	+	*	(	)	$	E	T	F
0	s5			s4					
1		s6				Accept			
2									
3									
4									
5									
6									
7									
8									
9									
10									
11									

Construction of SLR Parsing Table

Given Grammar:

E → E + T

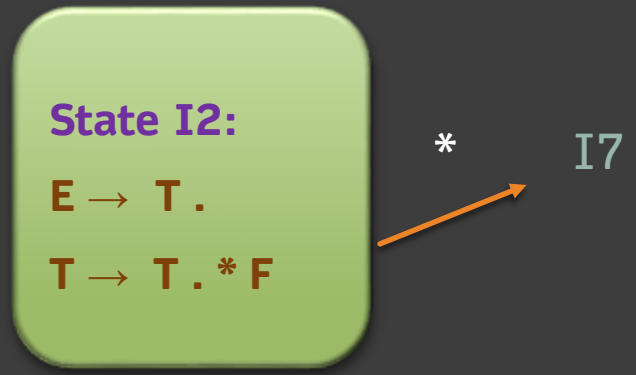
E → T

T → T * F

T → F

F → (E)

F → id



Given Grammar:

FOLLOW (E) = { + ,) , $ }

FOLLOW (T) = { * , + ,) , $ }

FOLLOW (F) = { * , + ,) , $ }

State	Action						GOTO		
	id	+	*	(	)	$	E	T	F
0	s5			s4					
1		s6				Accept			
2		r2	s7		r2	r2			
3									
4									
5									
6									
7									
8									
9									
10									
11									

Construction of SLR Parsing Table

Given Grammar:

E → E + T

E → T

T → T * F

T → F

F → (E)

F → id

State I3:

T → F .

Given Grammar:

FOLLOW (E) = { + ,) , $ }

FOLLOW (T) = { * , + ,) , $ }

FOLLOW (F) = { * , + ,) , $ }

State	Action						GOTO		
	id	+	*	(	)	$	E	T	F
0	s5			s4					
1		s6				Accept			
2		r2	s7		r2	r2			
3		r4	r4		r4	r4			
4									
5									
6									
7									
8									
9									
10									
11									

Construction of SLR Parsing Table

Given Grammar:

E → E + T

E → T

T → T * F

T → F

F → (E)

F → id

State I4:

F → (. E)

E → . E + T

E → . T

T → . T * F

T → . F

F → . (E)

F → . id

(→ I4

id → I5

State	Action						GOTO		
	id	+	*	(	)	$	E	T	F
0	s5			s4					
1		s6				Accept			
2		r2	s7		r2	r2			
3		r4	r4		r4	r4			
4	s5			s4					
5									
6									
7									
8									
9									
10									
11									

Construction of SLR Parsing Table

Given Grammar:

E → E + T

E → T

T → T * F

T → F

F → (E)

F → id

State I5:

F → id .

Given Grammar:

FOLLOW (E) = { + ,) , $ }

FOLLOW (T) = { * , + ,) , $ }

FOLLOW (F) = { * , + ,) , $ }

State	Action						GOTO		
	id	+	*	(	)	$	E	T	F
0	s5			s4					
1		s6				Accept			
2		r2	s7		r2	r2			
3		r4	r4		r4	r4			
4	s5			s4					
5		r6	r6		r6	r6			
6									
7									
8									
9									
10									
11									

Construction of SLR Parsing Table

Given Grammar:

E → E + T

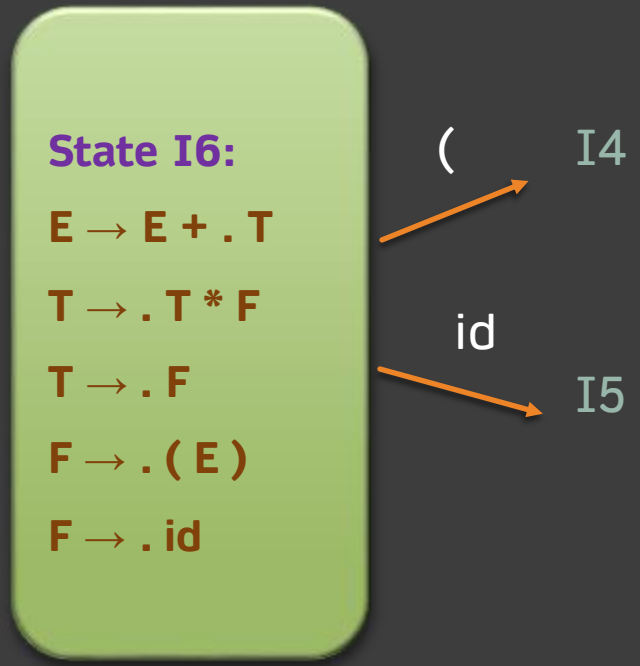
E → T

T → T * F

T → F

F → (E)

F → id



State	Action						GOTO		
	id	+	*	(	)	$	E	T	F
0	s5			s4					
1		s6				Accept			
2		r2	s7		r2	r2			
3		r4	r4		r4	r4			
4	s5			s4					
5		r6	r6		r6	r6			
6	s5			s4					
7									
8									
9									
10									
11									

Construction of SLR Parsing Table

Given Grammar:

E → E + T

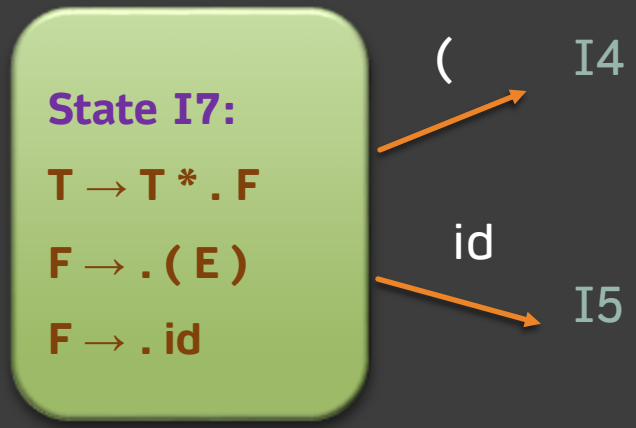
E → T

T → T * F

T → F

F → (E)

F → id



State	Action						GOTO		
	id	+	*	(	)	$	E	T	F
0	s5			s4					
1		s6				Accept			
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									
9									
10									
11									

Construction of SLR Parsing Table

Given Grammar:

E → E + T

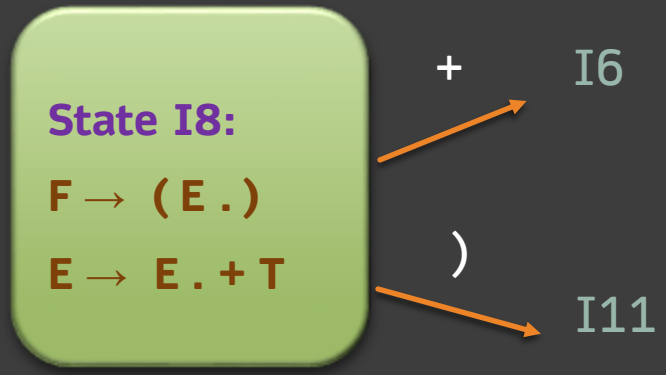
E → T

T → T * F

T → F

F → (E)

F → id



State	Action						GOTO		
	id	+	*	(	)	$	E	T	F
0	s5			s4					
1		s6				Accept			
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									
10									
11									

Construction of SLR Parsing Table

Given Grammar:

E → E + T

E → T

T → T * F

T → F

F → (E)

F → id

State I9:

E→ E + T .

T → T . * F

* → I7

Given Grammar:

FOLLOW (E) = { + ,) , $ }

FOLLOW (T) = { * , + ,) , $ }

FOLLOW (F) = { * , + ,) , $ }

State	Action						GOTO		
	id	+	*	(	)	$	E	T	F
0	s5			s4					
1		s6				Accept			
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									
11									

Construction of SLR Parsing Table

Given Grammar:

E → E + T

E → T

T → T * F

T → F

F → (E)

F → id

State I10:

T → T * F .

Given Grammar:

FOLLOW (E) = { + ,) , $ }

FOLLOW (T) = { * , + ,) , $ }

FOLLOW (F) = { * , + ,) , $ }

State	Action						GOTO		
	id	+	*	(	)	$	E	T	F
0	s5			s4					
1		s6				Accept			
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									

Construction of SLR Parsing Table

Given Grammar:

E → E + T

E → T

T → T * F

T → F

F → (E)

F → id

State I11:

F → (E) .

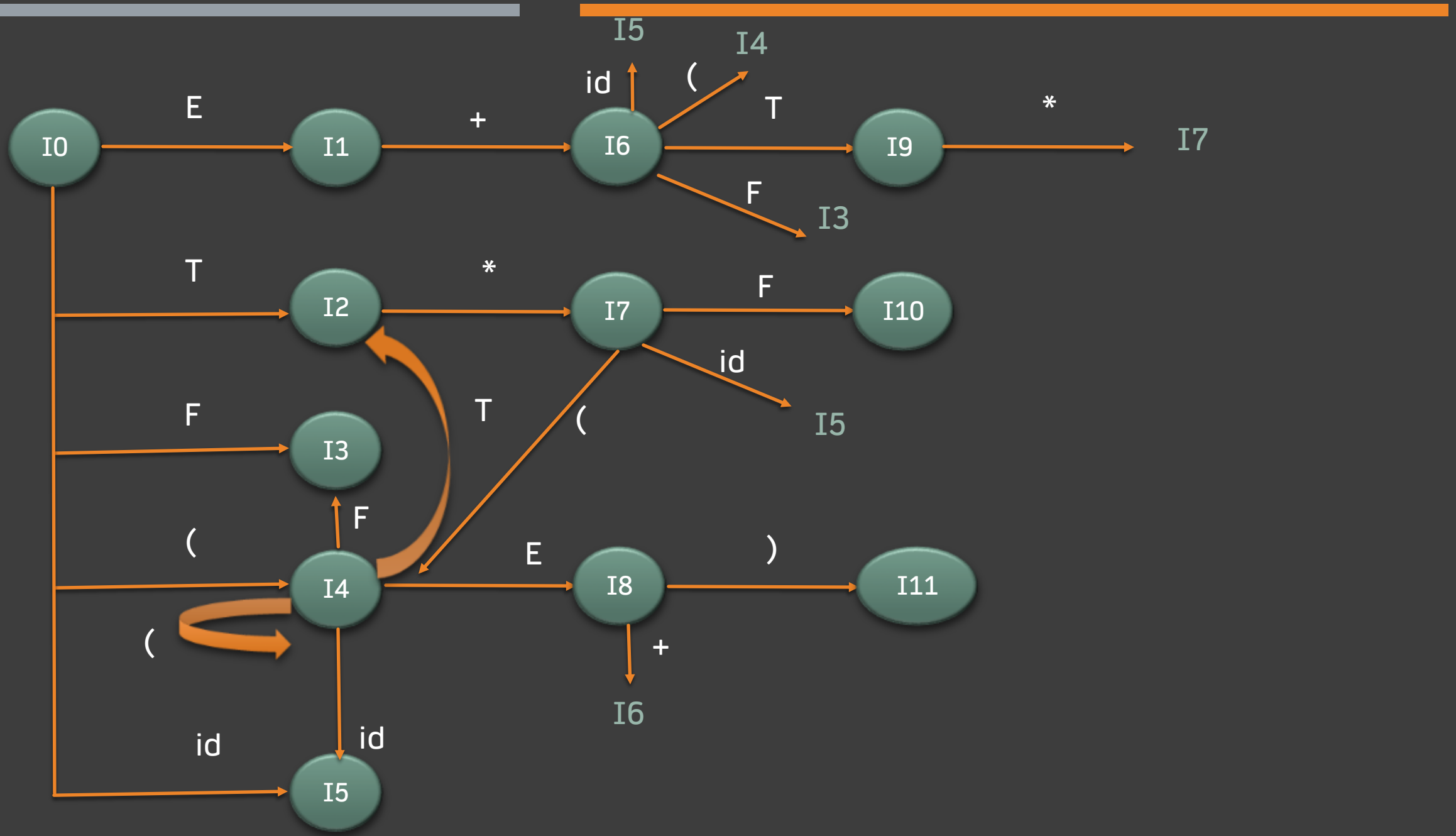
Given Grammar:

FOLLOW (E) = { + ,) , $ }

FOLLOW (T) = { * , + ,) , $ }

FOLLOW (F) = { * , + ,) , $ }

State	Action						GOTO		
	id	+	*	(	)	$	E	T	F
0	s5			s4					
1		s6				Accept			
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			


I0
E
I1
+
I6
id
I5
(
I4
T
I9
*
I7
F
I3
T
I2
*
I7
F
I10
id
I5
(
T
F
I3
F
(
I4
E
I8
)
I11
(
+
I6
id
id
I5

Construction of SLR Parsing Table

Given Grammar:

E → E + T

E → T

T → T * F

T → F

F → (E)

F → 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					
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			

Construction of SLR Parsing Table

Given Grammar:

E → E + T

E → T

T → T * F

T → F

F → (E)

F → 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					
7	s5			s4					
8		s6			s11				
9		r1	s7		r1	r1			
10		r3	r3		r3	r3			
11		r5	r5		r5	r5			

Construction of SLR Parsing Table

Given Grammar:

E → E + T

E → T

T → T * F

T → F

F → (E)

F → 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					
8		s6			s11				
9		r1	s7		r1	r1			
10		r3	r3		r3	r3			
11		r5	r5		r5	r5			

Construction of SLR Parsing Table

Given Grammar:

E → E + T

E → T

T → T * F

T → F

F → (E)

F → 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			

Construction of SLR Parsing Table

Given Grammar:

E → E + T

E → T

T → T * F

T → F

F → (E)

F → 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			

SLR 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 bottom-up parse for w; otherwise, there is an error

Method:

Initially, the parser has s0 on the stack where s0 is the initial state and w$ is in the input buffer.

Execute the following Algorithm

SLR Parsing Algorithm

```
let a be the first symbol of w$;
while(1)                                /* repeat forever */
{
    let s be the state on top of the stack;
    if ( ACTION [ s , a ] = shift t )
    {
            push t onto the stack;
            let a be the next input symbol;
     }
    else if ( ACTION [ s , a ] = reduce A → β )
    {
            pop | β | symbols of the stack;
            let state t now be on top of the stack;
            push GOTO [ t , A] onto the stack;
            output the production A → β ;
     }
     else if ( ACTION [ s , a ] = accept )
            break; /* parsing is done */
     else call error-recovery routine;
}
```

SLR Parsing Algorithm

Stack	Symbols	Input	Action	Output
0		id * id + id $	Shift	

SLR Parsing Algorithm

Stack	Symbols	Input	Action	Output
0		id * id + id $	Shift	
0 5	id	* id + id $	Reduce by F → id Pop '5' and check GOTO (0,F)	F → id

SLR Parsing Algorithm

Stack	Symbols	Input	Action	Output
0		id * id + id $	Shift	
0 5	id	* id + id $	Reduce by F → id Pop '5' and check GOTO (0,F)	F → id
0 3	F	* id + id $	Reduce by T → F Pop '3' and check GOTO (0,T)	T → F

SLR Parsing Algorithm

Stack	Symbols	Input	Action	Output
0		id * id + id $	Shift	
0 5	id	* id + id $	Reduce by F → id Pop '5' and check GOTO (0,F)	F → id
0 3	F	* id + id $	Reduce by T → F Pop '3' and check GOTO (0,T)	T → F
0 2	T	* id + id $	Shift	

SLR Parsing Algorithm

Stack	Symbols	Input	Action	Output
0		id * id + id $	Shift	
0 5	id	* id + id $	Reduce by F → id Pop '5' and check GOTO (0,F)	F → id
0 3	F	* id + id $	Reduce by T → F Pop '3' and check GOTO (0,T)	T → F
0 2	T	* id + id $	Shift	
0 2 7	T *	id + id $	Shift	

SLR Parsing Algorithm

Stack	Symbols	Input	Action	Output
0		id * id + id $	Shift	
0 5	id	* id + id $	Reduce by F → id Pop '5' and check GOTO (0,F)	F → id
0 3	F	* id + id $	Reduce by T → F Pop '3' and check GOTO (0,T)	T → F
0 2	T	* id + id $	Shift	
0 2 7	T *	id + id $	Shift	
0 2 7 5	T * id	+ id $	Reduce by F → id Pop '5' and check GOTO (7,F)	F → id

SLR Parsing Algorithm

Stack	Symbols	Input	Action	Output
0		id * id + id $	Shift	
0 5	id	* id + id $	Reduce by F → id Pop '5' and check GOTO (0,F)	F → id
0 3	F	* id + id $	Reduce by T → F Pop '3' and check GOTO (0,T)	T → F
0 2	T	* id + id $	Shift	
0 2 7	T *	id + id $	Shift	
0 2 7 5	T * id	+ id $	Reduce by F → id Pop '5' and check GOTO (7,F)	F → id
0 2 7 10	T * F	+ id $	Reduce by T → T * F Pop '10','7','2' and check GOTO (0,T)	T → T * F

SLR Parsing Algorithm

Stack	Symbols	Input	Action	Output
0		id * id + id $	Shift	
0 5	id	* id + id $	Reduce by F → id Pop '5' and check GOTO (0,F)	F → id
0 3	F	* id + id $	Reduce by T → F Pop '3' and check GOTO (0,T)	T → F
0 2	T	* id + id $	Shift	
0 2 7	T *	id + id $	Shift	
0 2 7 5	T * id	+ id $	Reduce by F → id Pop '5' and check GOTO (7,F)	F → id
0 2 7 10	T * F	+ id $	Reduce by T → T * F Pop '10','7','2' and check GOTO (0,T)	T → T * F
0 2	T	+ id $	Reduce by E → T Pop '2' and check GOTO (0,E)	E → T

SLR Parsing Algorithm

Stack	Symbols	Input	Action	Output
0		id * id + id $	Shift	
0 5	id	* id + id $	Reduce by F → id Pop '5' and check GOTO (0,F)	F → id
0 3	F	* id + id $	Reduce by T → F Pop '3' and check GOTO (0,T)	T → F
0 2	T	* id + id $	Shift	
0 2 7	T *	id + id $	Shift	
0 2 7 5	T * id	+ id $	Reduce by F → id Pop '5' and check GOTO (7,F)	F → id
0 2 7 10	T * F	+ id $	Reduce by T → T * F Pop '10','7','2' and check GOTO (0,T)	T → T * F
0 2	T	+ id $	Reduce by E → T Pop '2' and check GOTO (0,E)	E → T
0 1	E	+ id $	Shift	

SLR Parsing Algorithm

Stack	Symbols	Input	Action	Output
0		id * id + id $	Shift	
0 5	id	* id + id $	Reduce by F → id Pop '5' and check GOTO (0,F)	F → id
0 3	F	* id + id $	Reduce by T → F Pop '3' and check GOTO (0,T)	T → F
0 2	T	* id + id $	Shift	
0 2 7	T *	id + id $	Shift	
0 2 7 5	T * id	+ id $	Reduce by F → id Pop '5' and check GOTO (7,F)	F → id
0 2 7 10	T * F	+ id $	Reduce by T → T * F Pop '10','7','2' and check GOTO (0,T)	T → T * F
0 2	T	+ id $	Reduce by E → T Pop '2' and check GOTO (0,E)	E → T
0 1	E	+ id $	Shift	
0 1 6	E +	id $	Shift	

SLR Parsing Algorithm

Stack	Symbols	Input	Action	Output
0		id * id + id $	Shift	
0 5	id	* id + id $	Reduce by F → id Pop '5' and check GOTO (0,F)	F → id
0 3	F	* id + id $	Reduce by T → F Pop '3' and check GOTO (0,T)	T → F
0 2	T	* id + id $	Shift	
0 2 7	T *	id + id $	Shift	
0 2 7 5	T * id	+ id $	Reduce by F → id Pop '5' and check GOTO (7,F)	F → id
0 2 7 10	T * F	+ id $	Reduce by T → T * F Pop '10','7','2' and check GOTO (0,T)	T → T * F
0 2	T	+ id $	Reduce by E → T Pop '2' and check GOTO (0,E)	E → T
0 1	E	+ id $	Shift	
0 1 6	E +	id $	Shift	
0 1 6 5	E + id	$	Reduce by F → id Pop '5' and check GOTO (6,F)	F → id

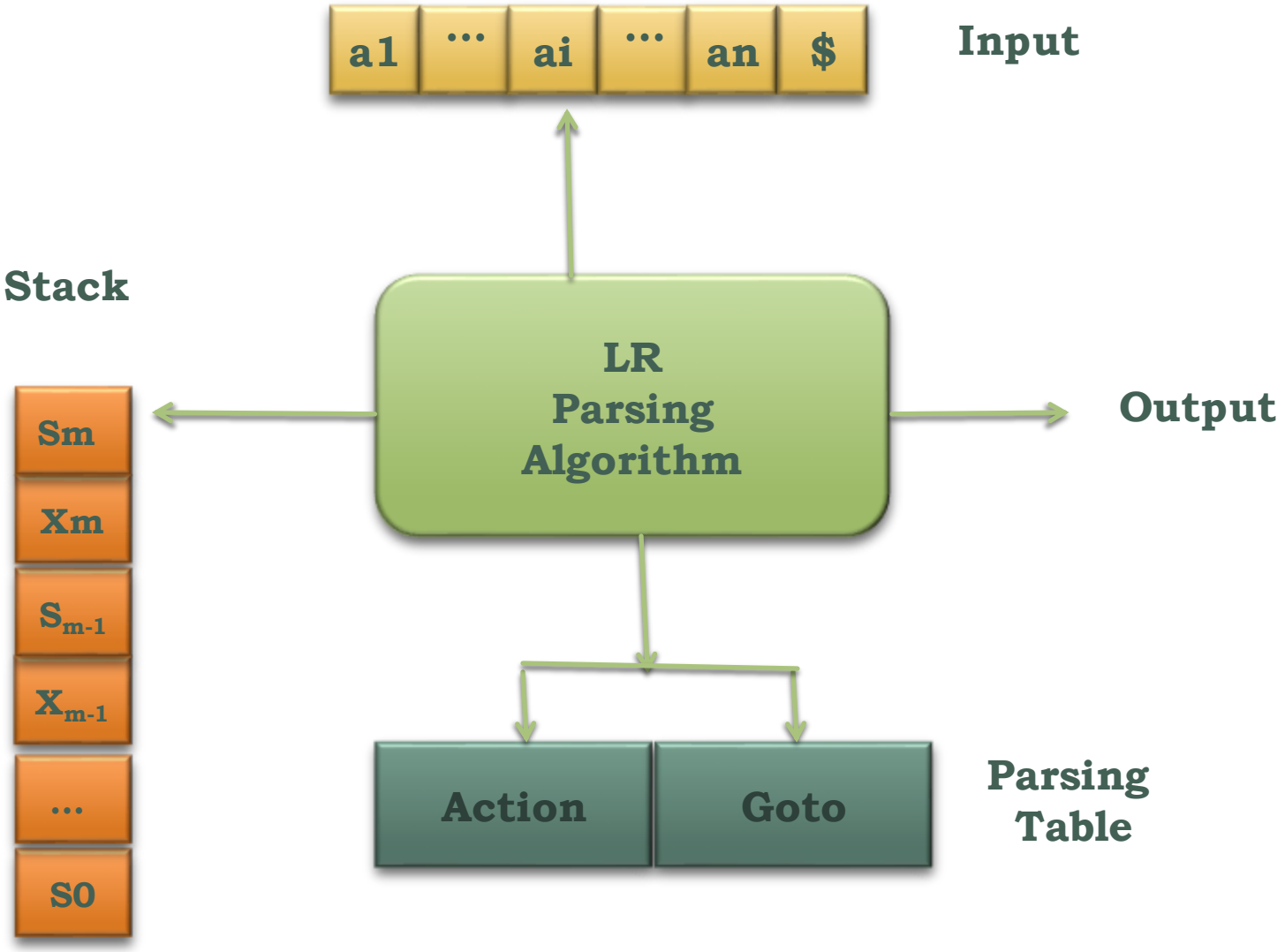
SLR Parsing Algorithm

Stack	Symbols	Input	Action	Output
0		id * id + id $	Shift	
0 5	id	* id + id $	Reduce by F → id Pop '5' and check GOTO (0,F)	F → id
0 3	F	* id + id $	Reduce by T → F Pop '3' and check GOTO (0,T)	T → F
0 2	T	* id + id $	Shift	
0 2 7	T *	id + id $	Shift	
0 2 7 5	T * id	+ id $	Reduce by F → id Pop '5' and check GOTO (7,F)	F → id
0 2 7 10	T * F	+ id $	Reduce by T → T * F Pop '10','7','2' and check GOTO (0,T)	T → T * F
0 2	T	+ id $	Reduce by E → T Pop '2' and check GOTO (0,E)	E → T
0 1	E	+ id $	Shift	
0 1 6	E +	id $	Shift	
0 1 6 5	E + id	$	Reduce by F → id Pop '5' and check GOTO (6,F)	F → id
0 1 6 3	E + F	$	Reduce by T → F Pop '3' and check GOTO (6,T)	T → F

SLR Parsing Algorithm

Stack	Symbols	Input	Action	Output
0		id * id + id $	Shift	
0 5	id	* id + id $	Reduce by F → id Pop '5' and check GOTO (0,F)	F → id
0 3	F	* id + id $	Reduce by T → F Pop '3' and check GOTO (0,T)	T → F
0 2	T	* id + id $	Shift	
0 2 7	T *	id + id $	Shift	
0 2 7 5	T * id	+ id $	Reduce by F → id Pop '5' and check GOTO (7,F)	F → id
0 2 7 10	T * F	+ id $	Reduce by T → T * F Pop '10','7','2' and check GOTO (0,T)	T → T * F
0 2	T	+ id $	Reduce by E → T Pop '2' and check GOTO (0,E)	E → T
0 1	E	+ id $	Shift	
0 1 6	E +	id $	Shift	
0 1 6 5	E + id	$	Reduce by F → id Pop '5' and check GOTO (6,F)	F → id
0 1 6 3	E + F	$	Reduce by T → F Pop '3' and check GOTO (6,T)	T → F
0 1 6 9	E + T	$	Reduce by E → E + T Pop '9', '6', '1' and check GOTO (0,E)	E → E + T

Stack	Symbols	Input	Action	Output
0		id * id + id $	Shift	
0 5	id	* id + id $	Reduce by F → id Pop '5' and check GOTO (0,F)	F → id
0 3	F	* id + id $	Reduce by T → F Pop '3' and check GOTO (0,T)	T → F
0 2	T	* id + id $	Shift	
0 2 7	T *	id + id $	Shift	
0 2 7 5	T * id	+ id $	Reduce by F → id Pop '5' and check GOTO (7,F)	F → id
0 2 7 10	T * F	+ id $	Reduce by T → T * F Pop '10','7','2' and check GOTO (0,T)	T → T * F
0 2	T	+ id $	Reduce by E → T Pop '2' and check GOTO (0,E)	E → T
0 1	E	+ id $	Shift	
0 1 6	E +	id $	Shift	
0 1 6 5	E + id	$	Reduce by F → id Pop '5' and check GOTO (6,F)	F → id
0 1 6 3	E + F	$	Reduce by T → F Pop '3' and check GOTO (6,T)	T → F
0 1 6 9	E + T	$	Reduce by E → E + T Pop '9', '6', '1' and check GOTO (0,E)	E → E + T
0 1	E	$	Accept	



MODEL OF SLR PARSER

On Stack:

Each Xi – Grammar Symbol

Si – State

The parsing Table consists of two parts

1. **Action**
2. **Goto**