University of Mosul
College Computer Science & Mathematics
Department of Computer Science

Compiler 2

Lecture 2: LR Parser Family

BY: Assist Prof. Dr. Ielaf O Abdul Majjed

The LR(k) parsing technique was introduced by Knuth in 1965

L is for Left-to-right scanning of input, R corresponds to a Rightmost derivation done in reverse, and k is the number of lookahead symbols used to make parsing decisions.

There are three widely used Algorithms available for constructing an LR parser:

SLR (1) – Simple LR Parser.
LR (1) – LR Parser.
LALR (1) – Look-Ahead LR Parser

Rules for LR parser:

The rules of LR parser as follows:

- The first item from the given grammar rules adds itself as the first closed set.
- If an object is present in the closure of the form A⟶ α. β. γ, where the next symbol after the symbol is non-terminal, add the symbol's production rules where the dot precedes the first item.
- Repeat steps (B) and (C) for new items added under (B).

The LR-Parsing Algorithm

A schematic of an LR parser consists of an ***input, an output, a stack, a driver program, and a parsing table*** that has two pasts (ACTION and GOTO).

- The driver program is the same for all LR parsers; only the **parsing table** changes from one parser to another.
- The parsing program reads characters from an **input buffer** one at a time.
- Where a shift-reduce parser would **shift a symbol**, an LR parser shifts a state.
- Each state summarizes the information contained in the **stack** below it.

Parsing Table:

Parsing table is divided into two parts- ***Action table and Go-To table***. The action table gives a grammar rule to implement the given current state and current terminal in the input stream.

1. The **ACTION function** takes as arguments a state i and a terminal a (or $, the input endmarker).

The value of ACTION [i, a] can have one of four forms:

a. Shift j, where j is a state : The action taken by the parser effectively shifts input a to the stack, but uses state j to represent a.

b. Reduce A β. The action of the parser effectively reduces β on the top of the stack to head A.

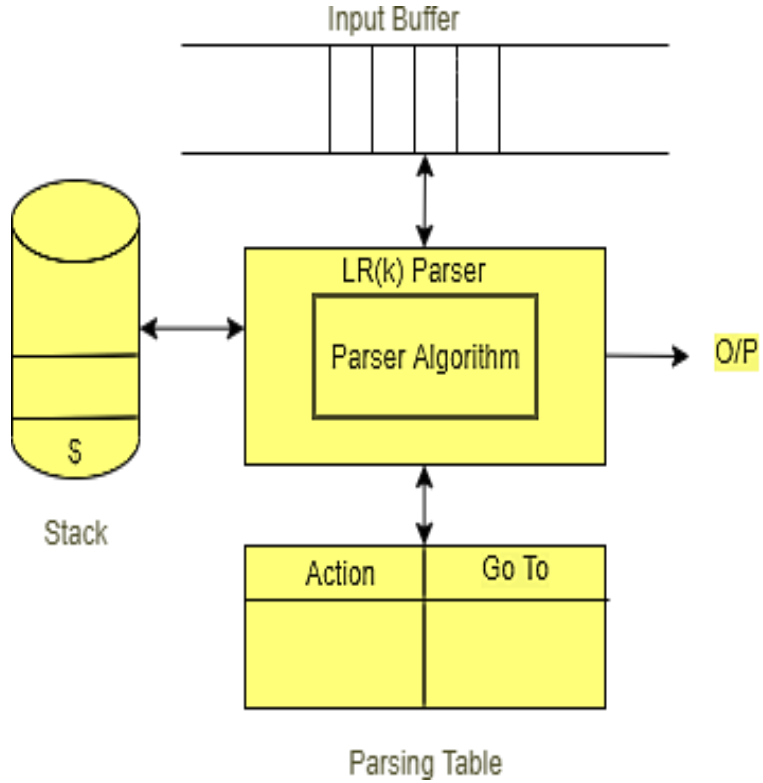
c. Accept. The parser accepts the input and finishes parsing.

d. Error. The parser discovers an error in its input and takes some corrective action.

2. We extend the **GOTO function**, defined on sets of items, to states:
if GOTO [Ii, A] = Ij, then GOTO also maps a state i and a nonterminal A to state j

Example: For the following grammar, the LR-parsing table's ACTION and GOTO functions are as follows:

E → E+T | T
T → T*F | F
F → (E) | id



Repeated with the productions numbered

1. **E → E + T**
2. **E → T**
3. **T → T * F**
4. **T → F**
5. **F → (E)**
6. **F → id**

The codes for the actions are:

1. ***si*** means **shift** and stack state ***i***,
2. **rj** means **reduce** by the production numbered **j**,
3. **acc** means **accept**,
4. **blank** means **error**.

First construct the set of items I0:

I0:

E →	**•E + T**	**r1**
E →	**•T**	**r2**
T →	**•T * F**	**r3**
T →	**•F**	**r4**
F →	**•(E)**	**r5**
F →	**•id**	**r6**

I1: Goto [I0, E]

E → E• + T ... Accept

I2: Goto [I0, T]

E → T• ... Complete

T → T• * F

I3: Goto [I0, F]

T → F•

I4: Goto [I0, ( ]

F → (•E)

E → •E + T

E → •T

T → •T * F

T → •F

F → •(E)

F → •id

I5: Goto [I0, id]

F → id• … Complete

I6: Goto [I1, +]

E → E +• T

T → •T * F

T → •F

F → •(E)

F → •id

I7: Goto [I2, *]

T → T *• F

F → •(E)

F → •id

I8: Goto [I4, E]

F → (E•)

E → E• + T

Goto [I4, T] = I2 قركم تلاح

Goto [I4, F] = I3

Goto [I4, (] = I4

Goto [I4, id] = I5

I9: Goto [I6, T]

E → E + T• … Complete

Goto [I6, T] = I2

Goto [I6, F] = I3

Goto [I6, (] = I4

Goto [I6, id] = I5

I10: Goto [I7, F]

T → T * F• … Complete

Goto [I7, (] = I4

Goto [I7, id] = I5

I11: Goto [I8,)]

F → (E) • … Complete

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

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

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

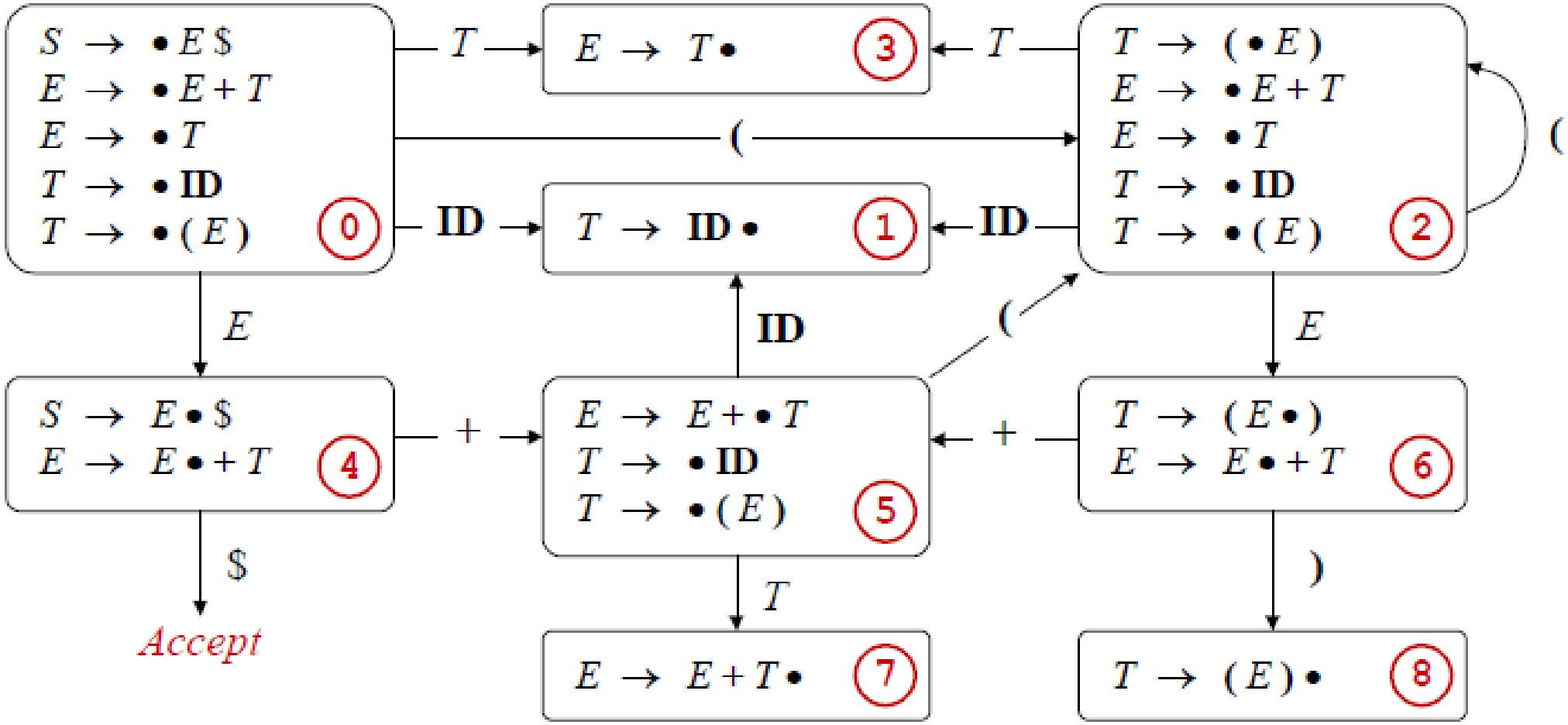
1. E → E + T
2. E → T
3. T → T * F
4. T → F
5. F → (E)
6. F → id

	Action						Goto		
	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			

LR Parser Table:

Stack	Input Symbols	Output
0	id * id + id $	Shift
0 id 5	* id + id $	Reduce F → id
0 F 3	* id + id $	Reduce T → F
0 T 2	* id + id $	Shift
0 T 2 * 7	id + id $	Shift
0 T 2 * 7 id 5	+ id $	Reduce F → id
0 T 2 * 7 F 10	+ id $	Reduce T → T * F
0 T 2	+ id $	Reduce E → T
0 E 1	+ id $	Shift
0 E 1 + 6	id $	Shift
0 E 1 + 6 id 5	$	Reduce F → id
0 E 1 + 6 F 3	$	Reduce T → F
0 E 1 + 6 T 9	$	Reduce E → E + T
0 E 1	$	Accept

Example 2


S → • E $
E → • E + T
E → • T
T → • ID
T → • (E)
0
T
E → T •
3
T
T → (• E)
E → • E + T
E → • T
T → • ID
T → • (E)
2
(
(
ID
T → ID •
1
ID
E
ID
(
E
S → E • $
E → E • + T
4
+
E → E + • T
T → • ID
T → • (E)
5
+
T → (E •)
E → E • + T
6
$
T
)
Accept
E → E + T •
7
T → (E) •
8

Constructing SLR-Parsing Tables

We shall refer to the parsing table constructed by this method as an SLR table, and to an LR parser using an SLR-parsing table as an SLR parser.

The other two methods augment the SLR method with lookahead information.

The ***action*** and ***goto*** entries in the parsing table are then constructed using the following algorithm. It requires us to know FOLLOW(A) for each nonterminal A of a grammar.

Constructing an SLR-parsing table Algorithm:

INPUT: An augmented grammar G'.

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

METHOD:

1. Construct C = {I_0, I_1, …., I_n}, the collection of sets of LR(0) items for G'.
2. State ***i*** is constructed from I***i***.

   The parsing actions for state ***i*** are determined as follows:

   a. If [**A** → **α.αβ**] is in I***i*** and GOTO (I***i***, ***α***) = I**j**, then set ACTION [***i***, ***α***] to "**shift j**." Here ***α*** must be a terminal.

   b. If [**A** → **α.**] is in I***i***, then set ACTION [***i***, ***α***] to "**reduce A** → **α**" for all ***α*** in FOLLOW(A); here **A** may not be **S'**.

   c. if [**S'** → **S.**] is in I***i***, then set ACTION [***i***, **S**] to "**accept**." If any 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 nonterrninals **A** using the rule: If GOTO (I***i***, A) = I**j**, then 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**].

Example:

Let us construct the SLR table for the **augmented** expression grammar. The canonical collection of sets of LR(0) items for the grammar.

I0:

E' →	**•E**	**r1**
E →	**•E + T**	**r2**
E →	**•T**	**r3**
T →	**•T * F**	**r4**
T →	**•F**	**r5**
F →	**•(E)**	**r6**
F →	**•id**	**r7**

I1: Goto [I0, E]

E' → E• … Accept

E → E• + T

I2: Goto [I0, T]

E → T• … Complete

T → T• * F

I3: Goto [I0, F]

T → F• … Complete

I4: Goto [I0, (]

F → (•E)

E → •E + T

E → •T

T → •T * F

T → •F

F → •(E)

F → •id

I5: Goto [I0, id]

F → id• … Complete

I6: Goto [I1, +]

E → E +• T

T → •T * F

T → •F

F → •(E)

F → •id

I7: Goto [I2, *]

T → T *• F

F → •(E)

F → •id

I8: Goto [I4, *]

F → (E•)

Goto [I4, E] = I1

Goto [I4, T] = I2

Goto [I4, F] = I3

Goto [I4, (] = I4

Goto [I4, id] = I5

I9: Goto [I6, T]

E → E + T• … Complete

Goto [I6, T] = I2

Goto [I6, F] = I3

Goto [I6, (] = I4

Goto [I6, id] = I5

I10: Goto [I7, F]

T → T * F• … Complete

Goto [I7, (] = I4

Goto [I7, id] = I5

I11: Goto [I8,)]

F → (E)• … Complete

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

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

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

SLR-Parsing Tables:

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			