CSL302: Compiler Design

Bottom Up Parsing

Vishwesh Jatala

Assistant Professor

Department of CSE

Indian Institute of Technology Bhilai

vishwesh@iitbhilai.ac.in

Acknowledgement

- Today's slides are modified from that of *Stanford University:*
    - *https://web.stanford.edu/class/archive/cs/cs143/cs143.1128/*

Exercise: Construct Parser Table

$E \longrightarrow T$

$T \longrightarrow T$ `*` F

$T \longrightarrow F$

$F \longrightarrow$ `id`

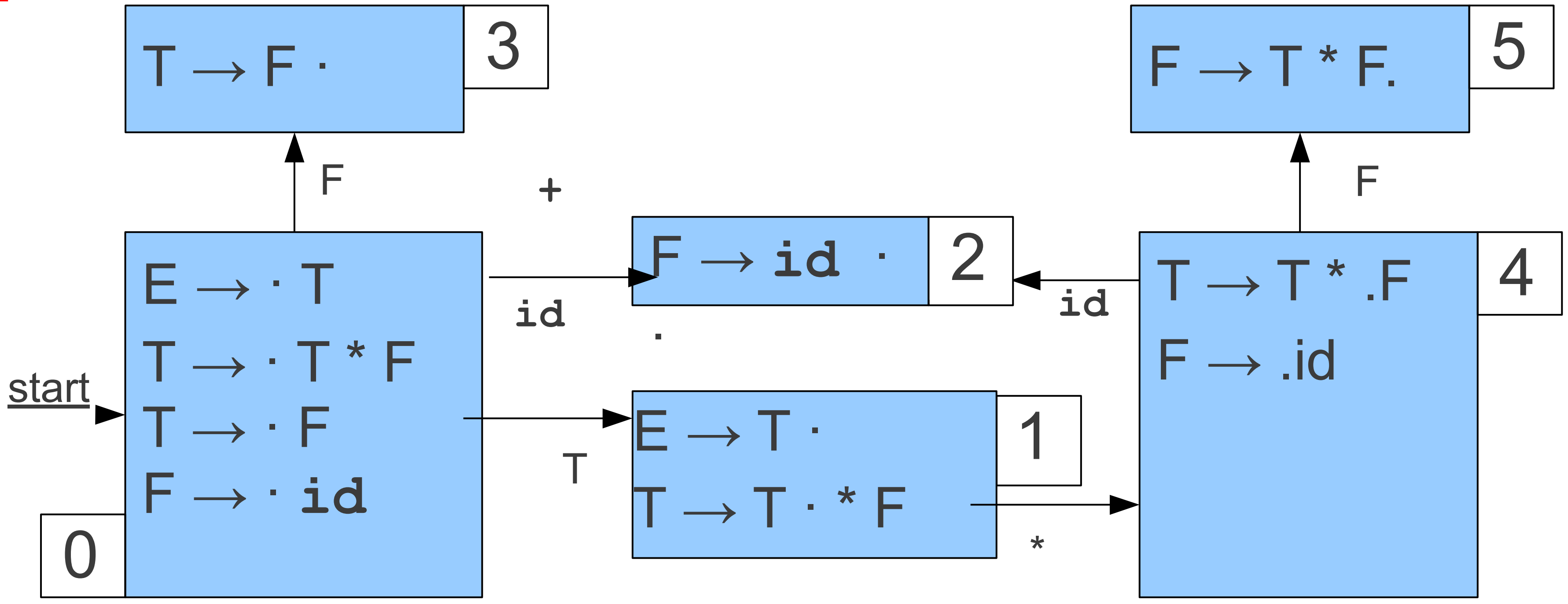
A Deterministic Automaton

$E \rightarrow T$

$T \rightarrow T * F$

$T \rightarrow F$

$F \rightarrow$ id



LR(0) Tables

(1) $E \rightarrow T$

(2) $T \rightarrow T * F$

(3) $T \rightarrow F$

(4) $F \rightarrow id$

	Action		Goto	
	id	*	T	F
0	S2		S1	S3
1	r1	S4/r1		
2	r4	r4		
3	r3	r3		
4	S2			S5
5	r2	r2		

$S \rightarrow E$

$E \rightarrow T$
$E \rightarrow E$ `+` T

$T \rightarrow$ `int`

$T \rightarrow$ `(`E`)`

LR (0) Parsing

S → E

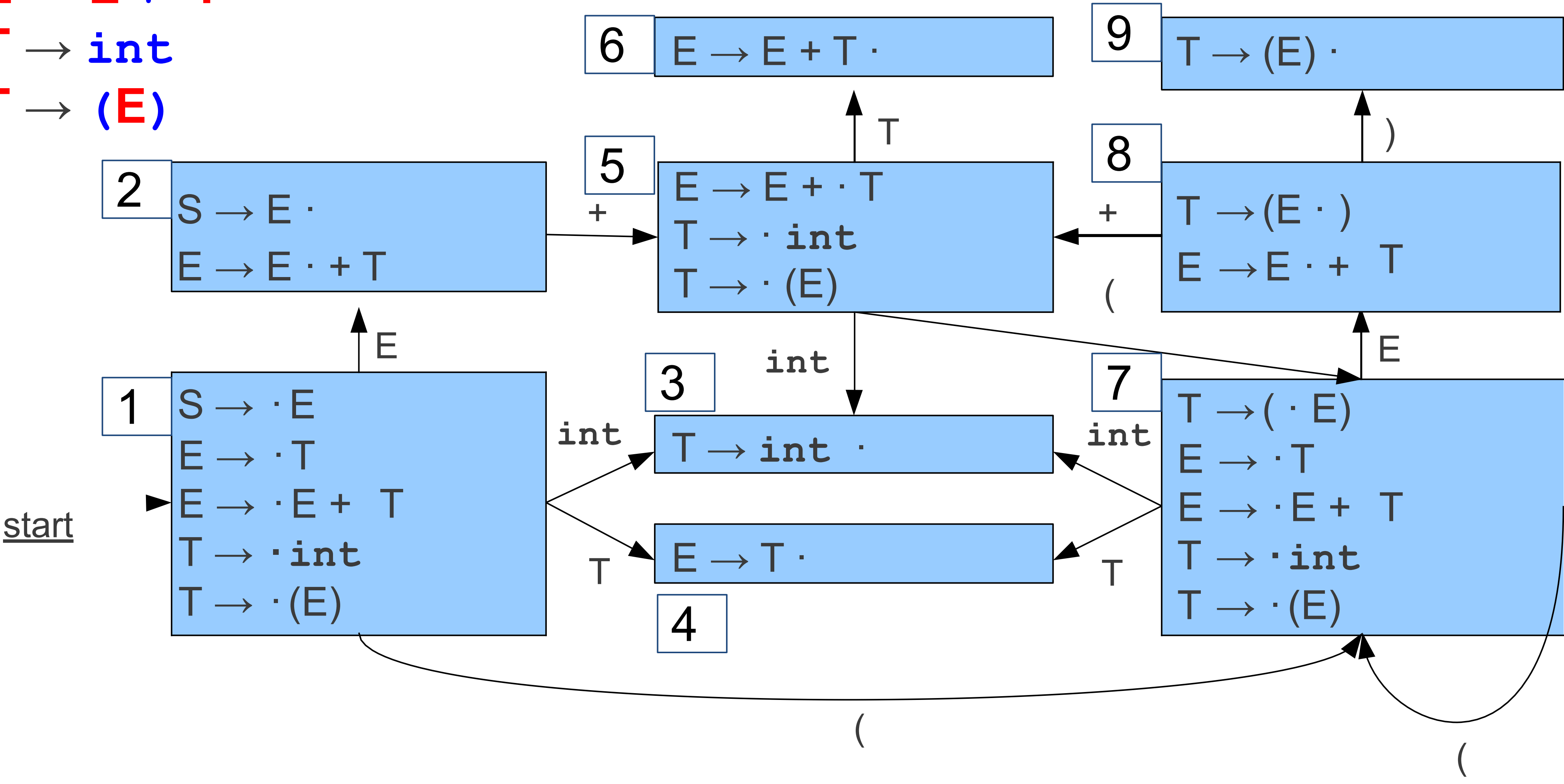
E → T

E → E + T

T → int

T → (E)

LR (0) Parsing



LR(0) Table

(1) S → E

(2) E → T

(3) E → E + T

(4) T → `int`

(5) T → `(E)`

	Action					Goto	
	`int`	`+`	`(`	`)`	`$`	E	T
1	S3					2	4
2	r1	S5/r1	r1	r1	r1		
3	r4	r4	r4	r4	r4		
4	r2	r2	r2	r2	r2		
5	S3		S7				6
6	r3	r3	r3	r3	r3		
7	S3		S7			8	4
8		S5		S9			
9	r5	r5	r5	r5	r5		

LR Conflicts

- A **shift/reduce conflict** is an error where a shift/reduce parser cannot tell whether to shift a token or perform a reduction.
  - Often happens when two productions overlap.
- A **reduce/reduce conflict** is an error where a shift/reduce parser cannot tell which of many reductions to perform.
  - Often the result of ambiguous grammars.
- A grammar whose handle-finding automaton contains a shift/reduce conflict or a reduce/reduce conflict is not LR(0).
- Can you have a shift/shift conflict?

Example

(1) $S \longrightarrow E$

(2) $E \longrightarrow T$

(3) $E \longrightarrow E + T$

(4) $T \longrightarrow$ `int`

(5) $T \longrightarrow$ `(E)`

Try to parse int+int using LR(0)

SLR(1)

- **Simple LR(1)**
- Minor modification to LR(0) automaton that uses lookahead to avoid shift/reduce conflicts.

S → E

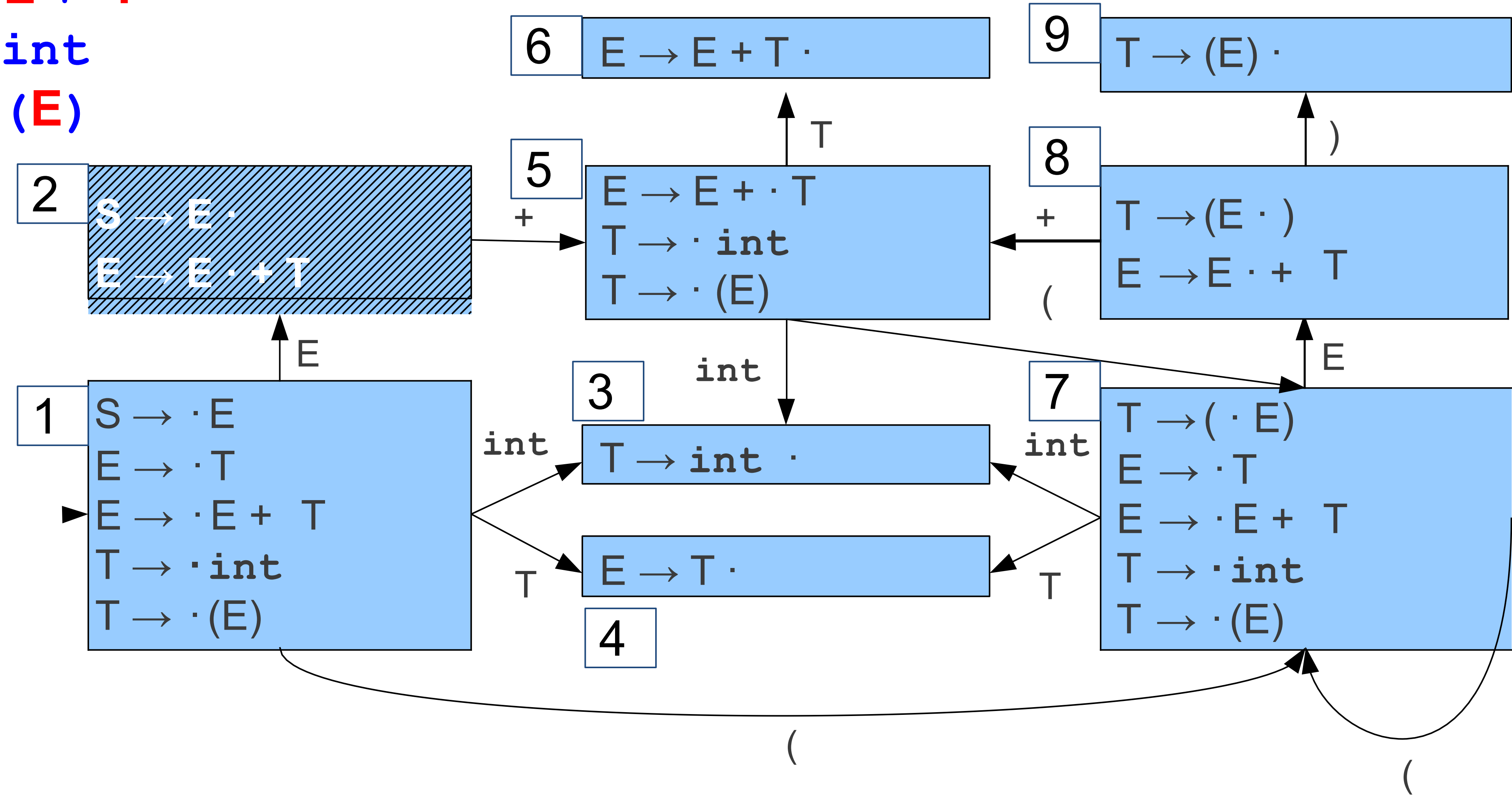
E → T

E → E + T

T → int

T → (E)

SLR(1)Parsing



SLR(1) Table

(1) S → E
(2) E → T
(3) E → E + T
(4) T → `int`
(5) T → `(E)`

	Action					Goto	
	`int`	`+`	`(`	`)`	`$`	E	T
1	S3					2	4
2		S5					
3							
4							
5	S3		S7				6
6							
7	S3		S7			8	4
8		S5		S9			
9							

SLR(1) Table

(1) $S \longrightarrow E$

(2) $E \longrightarrow T$

(3) $E \longrightarrow E + T$

(4) $T \longrightarrow$ `int`

(5) $T \longrightarrow$ `(E)`

	Action					Goto	
	`int`	`+`	`(`	`)`	`$`	E	T
1	S3					2	4
2		S5			**r1**		
3		**r4**		**r4**	**r4**		
4		**r2**		**r2**	**r2**		
5	S3		S7				6
6		**r3**		**r3**	**r3**		
7	S3		S7			8	4
8		S5		S9			
9		**r5**		**r5**	**r5**		

SLR(1)

- **Simple LR(1)**
- Idea: Only reduce **A** $\rightarrow$ ***ω*** if the next token `t` is in FOLLOW(**A**).
- Automaton identical to LR(0) automaton; only change is when we choose to reduce.
-

Example

(1) S $\rightarrow$ E

(2) E $\rightarrow$ T

(3) E $\rightarrow$ E + T

(4) T $\rightarrow$ `int`

(5) T $\rightarrow$ `(E)`

Try to parse int+int using SLR(1)

Analysis of SLR(1)

- Exploits lookahead in a small space.
  - Small automaton – same number of states as in as LR(0).
  - Works on many more grammars than LR(0)
- Too weak for most grammars: lose context from not having extra states.

The Limits of SLR(1)

$S \rightarrow E$

$E \rightarrow L$ `=` R

$E \rightarrow R$

$L \rightarrow$ `id`

$L \rightarrow$ `*`R

$R \rightarrow L$