Introduction

- Shift/Reduce Conflict
- Error Recovery in LR Parsing

Shift/Reduce Conflict

- We say that we cannot introduce a shift/reduce conflict during the shrink process for the creation of the states of a LALR parser.
- Assume that we can introduce a shift/reduce conflict. In this case, a state of LALR parser must have:

$$A \rightarrow \alpha\,.\,,a \qquad \text{and} \qquad B \rightarrow \beta\,.\,a\gamma,b$$

- This means that a state of the canonical LR(1) parser must have:

$$A \rightarrow \alpha\,.\,,a \qquad \text{and} \qquad B \rightarrow \beta\,.\,a\gamma,c$$

But, this state has also a shift/reduce conflict. i.e. The original canonical LR(1) parser has a conflict.

(Reason for this, the shift operation does not depend on lookaheads)

Reduce/Reduce Conflict

- But, we may introduce a reduce/reduce conflict during the shrink process for the creation of the states of a LALR parser.

$I_1 : A \rightarrow \alpha\,.\,,a$
$\quad B \rightarrow \beta\,.\,,b$

$I_2 : A \rightarrow \alpha\,.\,,b$
$\quad B \rightarrow \beta\,.\,,c$

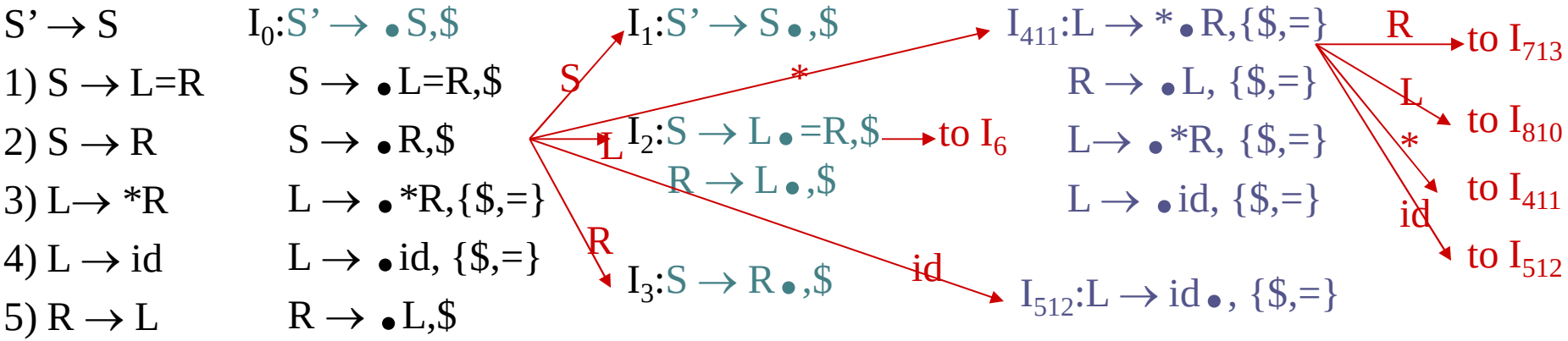
$\Downarrow$

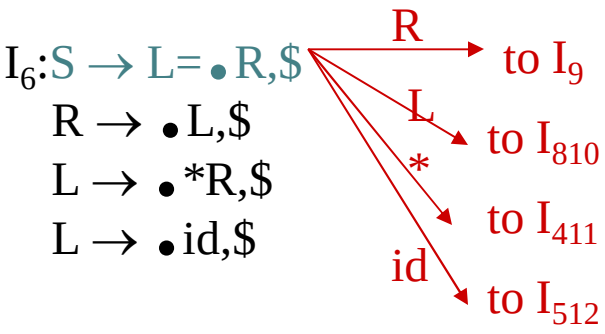
$I_{12} : A \rightarrow \alpha\,.\,, \{a,b\}$ ➔ reduce/reduce conflict

$B \rightarrow \beta\,.\,,\{b,c\}$

Canonical LALR(1) Collection – Example2

S' → S
1) S → L=R
2) S → R
3) L→ *R
4) L → id
5) R → L





I9:S → L=R•,$

I713:L → *R•, {$,=}

I810: R → L•, {$,=}

Same Cores

I4 and I11

I5 and I12

I7 and I13

I8 and I10

LALR(1) Parsing Tables – (for Example2)

	id	*	=	$	S	L	R
0	s5	s4			1	2	3
1				acc			
2			s6	r5			
3				r2			
4	s5	s4				8	7
5			r4	r4			
6	s12	s11				10	9
7			r3	r3			
8			r5	r5			
9				r1			

no shift/reduce or
no reduce/reduce conflict

so, it is a LALR(1) grammar

Using Ambiguous Grammars

- All grammars used in the construction of LR-parsing tables must be un-ambiguous.
- Can we create LR-parsing tables for ambiguous grammars ?
  - Yes, but they will have conflicts.
  - We can resolve these conflicts in favor of one of them to disambiguate the grammar.
  - At the end, we will have again an unambiguous grammar.
- Why we want to use an ambiguous grammar?
  - Some of the ambiguous grammars are **much natural**, and a corresponding unambiguous grammar can be very complex.
  - Usage of an ambiguous grammar may **eliminate unnecessary reductions**.
- Ex.

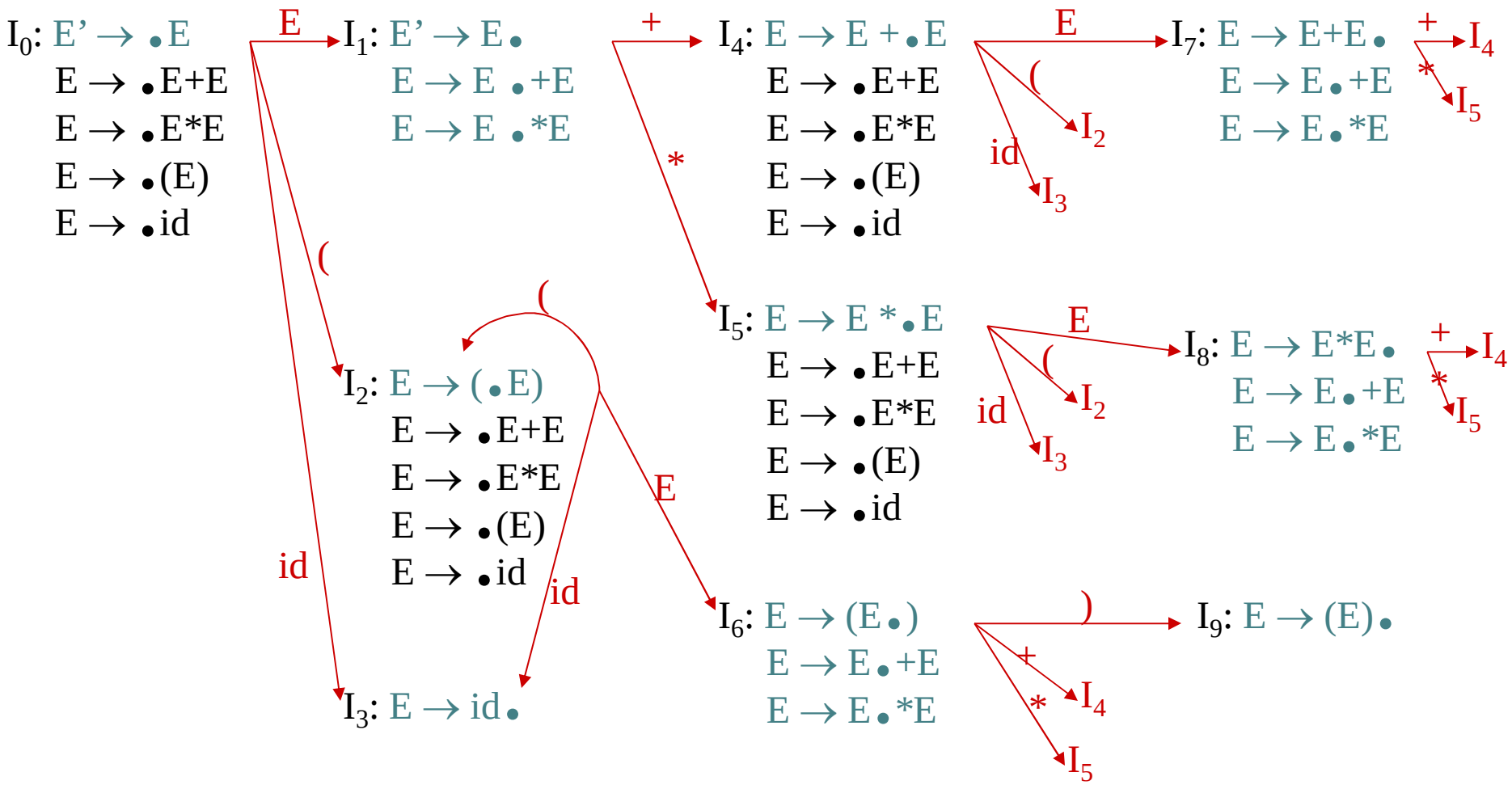
$E \rightarrow E+E \mid E*E \mid (E) \mid id$ ➔

$E \rightarrow E+T \mid T$

$T \rightarrow T*F \mid F$

$F \rightarrow (E) \mid id$

Sets of LR(0) Items for Ambiguous Grammar



SLR-Parsing Tables for Ambiguous Grammar

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

State I_7 has shift/reduce conflicts for symbols + and *.

$$I_0 \xrightarrow{E} I_1 \xrightarrow{+} I_4 \xrightarrow{E} I_7$$

when current token is +

shift ➔ + is right-associative

reduce ➔ + is left-associative

when current token is *

shift ➔ * has higher precedence than +

reduce ➔ + has higher precedence than *

SLR-Parsing Tables for Ambiguous Grammar

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

State I_8 has shift/reduce conflicts for symbols + and *.

$$I_0 \xrightarrow{E} I_1 \xrightarrow{*} I_5 \xrightarrow{E} I_8$$

when current token is *

shift ➔ * is right-associative

reduce ➔ * is left-associative

when current token is +

shift ➔ + has higher precedence than *

reduce ➔ * has higher precedence than +

SLR-Parsing Tables for Ambiguous Grammar

	Action							Goto
	id	**+**	*****	**(**	**)**	**$**		**E**
0	s3			s2				1
1		s4	s5			acc		
2	s3			s2				6
3		r4	r4		r4	r4		
4	s3			s2				7
5	s3			s2				8
6		s4	s5		s9			
7		r1	s5		r1	r1		
8		r2	r2		r2	r2		
9		r3	r3		r3	r3		

Error Recovery in LR Parsing

- An LR parser will detect an error when it consults the parsing action table and finds an error entry. All empty entries in the action table are error entries.
- Errors are never detected by consulting the goto table.
- An LR parser will announce error as soon as there is no valid continuation for the scanned portion of the input.
- A canonical LR parser (LR(1) parser) will never make even a single reduction before announcing an error.
- The SLR and LALR parsers may make several reductions before announcing an error.
- But, all LR parsers (LR(1), LALR and SLR parsers) will never shift an erroneous input symbol onto the stack.

Panic Mode Error Recovery in LR Parsing

- Scan down the stack until a state **s** with a goto on a particular nonterminal **A** is found. (Get rid of everything from the stack before this state s).
- Discard zero or more input symbols until a symbol **a** is found that can legitimately follow A.
  - The symbol a is simply in FOLLOW(A), but this may not work for all situations.
- The parser stacks the nonterminal **A** and the state **goto[s,A]**, and it resumes the normal parsing.
- This nonterminal A is normally is a basic programming block (there can be more than one choice for A).
  - stmt, expr, block, ...

Phrase-Level Error Recovery in LR Parsing

- Each empty entry in the action table is marked with a specific error routine.
- An error routine reflects the error that the user most likely will make in that case.
- An error routine inserts the symbols into the stack or the input (or it deletes the symbols from the stack and the input, or it can do both insertion and deletion).
  - missing operand
  - unbalanced right parenthesis