LR Parsing

PRACTICE EXAMPLE
Canonical LR(1) Item set

Consider the example grammar that contains the productions

0. $G \rightarrow S$
1. $S \rightarrow E = E$
2. $S \rightarrow f$
3. $E \rightarrow T$
4. $E \rightarrow E + T$
5. $T \rightarrow f$
6. $T \rightarrow T * f$

Note: G is equivalent to S'.

PRACTICE EXAMPLE
Canonical LR(1) Item set

0. $G \rightarrow S$
1. $S \rightarrow E = E$
2. $S \rightarrow f$
3. $E \rightarrow T$
4. $E \rightarrow E + T$
5. $T \rightarrow f$
6. $T \rightarrow T * f$

0	$*[G \rightarrow .S, \varepsilon]$
	$[S \rightarrow .E = E, \varepsilon]$
	$[S \rightarrow .f, \varepsilon]$
	$[E \rightarrow .T, = : +]$
	$[T \rightarrow .f, = : + : *]$
	$[T \rightarrow .T * f, = : + : *]$
	$[E \rightarrow .E + T, = : +]$
1	$*[S \rightarrow E. = E, \varepsilon]$
	$*[E \rightarrow E. + T, = : +]$
2	$*[E \rightarrow T., = : +]$
	$*[T \rightarrow T. * f, = : + : *]$
3	$*[S \rightarrow f., \varepsilon]$
	$*[T \rightarrow f., = : + : *]$
4	$*[S \rightarrow E = .E, \varepsilon]$
	$[E \rightarrow .T, \varepsilon : +]$
	$[T \rightarrow f, \varepsilon : + : *]$
	$[T \rightarrow .T * f, \varepsilon : + : *]$
	$[E \rightarrow .E + T, \varepsilon : +]$
5	$*[E \rightarrow E + .T, = : +]$
	$[T \rightarrow .f, = : + : *]$
	$[T \rightarrow .T * f, = : + : *]$

Note: G is equivalent to S'.
ε is equivalent to $

PRACTICE EXAMPLE
Canonical LR(1) Item set

0. $G \rightarrow S$
1. $S \rightarrow E = E$
2. $S \rightarrow f$
3. $E \rightarrow T$
4. $E \rightarrow E + T$
5. $T \rightarrow f$
6. $T \rightarrow T * f$

6	$\bullet[S \rightarrow E = E., \varepsilon]$
	$\bullet[E \rightarrow E. + T, \varepsilon : +]$
7	$\bullet[E \rightarrow T., \varepsilon : +]$
	$\bullet[T \rightarrow T. * f, \varepsilon : + : *]$
8	$\bullet[T \rightarrow f., \varepsilon : + : *]$
9	$\bullet[T \rightarrow f., = : + : *]$
10	$\bullet[E \rightarrow E + T., = : +]$
	$\bullet[T \rightarrow T. * f, = : + : *]$
11	$\bullet[T \rightarrow T * .f, = : + : *]$
12	$\bullet[T \rightarrow T * f., = : + : *]$
13	$\bullet[E \rightarrow E + .T, \varepsilon : +]$
	$[T \rightarrow .f, \varepsilon : + : *]$
	$[T \rightarrow .T * f, \varepsilon : + : *]$
14	$\bullet[E \rightarrow E + T., \varepsilon : +]$
	$\bullet[T \rightarrow T. * f, \varepsilon : + : *]$
15	$\bullet[T \rightarrow T * .f, \varepsilon : + : *]$
16	$\bullet[T \rightarrow T * f., \varepsilon : + : *]$
17	$\bullet[G \rightarrow S., \varepsilon]$

Note: G is equivalent to S'.
ε is equivalent to $

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.

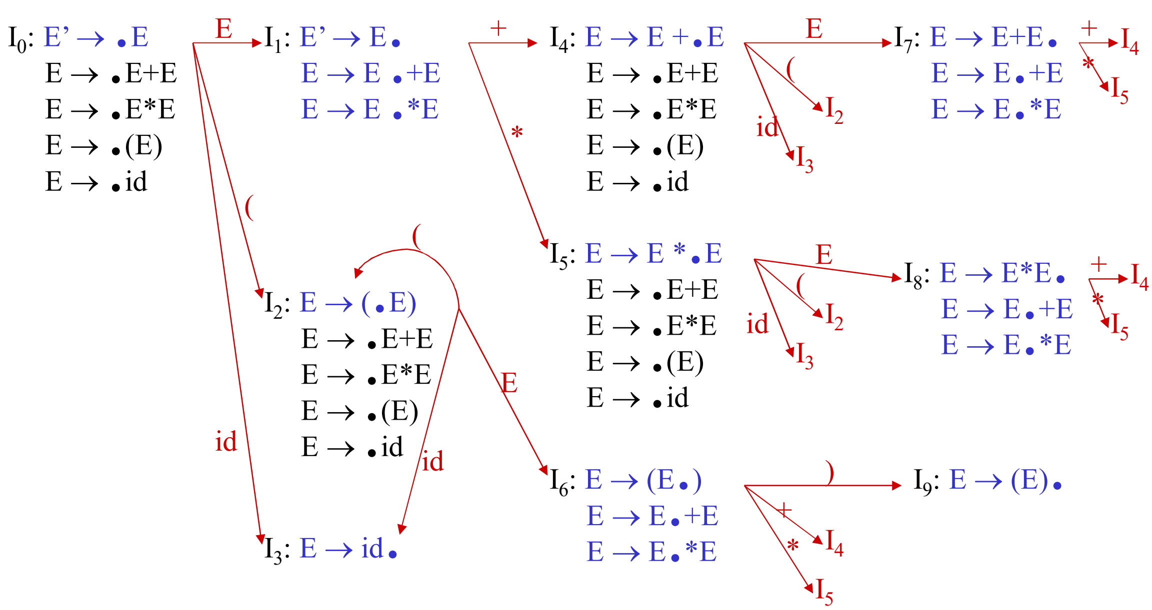
$E \rightarrow E+E \mid E*E \mid (E) \mid id$ ➔

$E \rightarrow E+T \mid T$

$T \rightarrow T*F \mid F$

$F \rightarrow (E) \mid id$

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.