Compiler Principle

Prof. Dongming LU

Mar. 11th, 2024

Content

1. INTRODUCTION

2. LEXICAL ANALYSIS
3. **PARSING**
4. ABSTRACT SYNTAX
5. SEMANTIC ANALYSIS

6. ACTIVATION RECORD
7. TRANSLATING INTO INTERMEDIATE CODE

8. OTHERS

3 Parsing

3.3 LR Parsing

Bottom-up Parsing

- The weakness of LL(*k*) parsing techniques is that they must ***predict*** which production to use, having seen only the first *k* tokens of the right-hand side.

- LR(*k*) parsing, is able to ***postpone*** the decision until it has seen input tokens corresponding to the entire right-hand side of the production in question.

Bottom-up Parsing

- **Shift-reduce parsing**
  - ✓ Bottom-up parsing
  - ✓ LR(k) Left-to-right parse,
    Rightmost derivation
    k-token lookahead
- **More powerful than LL(k) parsers**

LALR variant:
- ✓ The basis for parsers for most modern programming languages
- ✓ Implemented in tools such as Yacc

Example

Grammar:

Parsing Table

$A \rightarrow S\ EOF \qquad L \rightarrow L\ ;\ S \qquad S \rightarrow (\ L\ )$

$\qquad\qquad\qquad\quad L \rightarrow S \qquad\qquad S \rightarrow id = num$

Example

Grammar:

$A \rightarrow S\ EOF \qquad L \rightarrow L\ ;\ S \qquad S \rightarrow (\ L\ )$

$\qquad\qquad\qquad\quad L \rightarrow S \qquad\qquad S \rightarrow id = num$

Parsing Table

Input from lexer: $\overbrace{(\ id = num\ ;\ id = num\ )\ EOF}^{\text{yet to read}}$

State of parse so far:

Example

Grammar:

$A \rightarrow S$ EOF $L \rightarrow L$; S $S \rightarrow (L)$
$L \rightarrow S$ $S \rightarrow$ id = num

Parsing Table

yet to read

Input from lexer: (id = num ; id = num) EOF

State of parse so far: (

SHIFT

Example

Grammar:

A→S EOF   L → L ; S   S →(L)
           L → S   S → id = num

Parsing Table

Input from lexer: $(\; \overbrace{\text{id = num ; id = num) EOF}}^{\text{yet to read}}$

State of parse so far: (id

SHIFT

Example

Grammar:

A→S EOF   L → L ; S   S →(L)

L → S   S → id = num

Parsing Table

yet to read

Input from lexer: (id = num ; id = num) EOF

State of parse so far: (id =

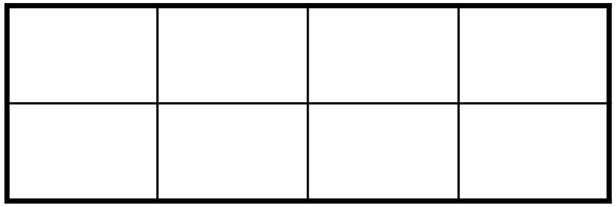
SHIFT

Example

Grammar:

A → S EOF L → L ; S S → (L)
           L → S   S → id = num

Parsing Table



Input from lexer: $(\text{ id = num } \overbrace{\text{; id = num) EOF}}^{\text{yet to read}}$

State of parse so far: (id = num

SHIFT

Example

Grammar:

A → S EOF   L → L ; S   S → (L)
             L → S   S → id = num

Parsing Table

yet to read

Input from lexer: (id = num ; id = num) EOF

State of parse so far: (L

REDUCE

S → id = num

L → S

Example

Grammar:

A→S EOF   L → L ; S   S →(L)
            L → S   S → id = num

Parsing Table

yet to read

Input from lexer: (id = num ; id = num) EOF

SHIFT

State of parse so far: (L ;

Example

Grammar:

$A \rightarrow S\ EOF$ $\quad L \rightarrow L\ ;\ S$ $\quad S \rightarrow (\ L\ )$

$L \rightarrow S$ $\quad S \rightarrow id = num$

Parsing Table

yet to read

Input from lexer: (id = num ; id = num) $\underbrace{}_{}$EOF

State of parse so far: (L ; id = num

SHIFT
SHIFT
SHIFT

Example

Grammar:

A→S EOF L → L ; S S →(L)

L → S S → id = num

Parsing Table

yet to read

Input from lexer: (id = num ; id = num) EOF

State of parse so far: (L ; S

REDUCE

S → id = num

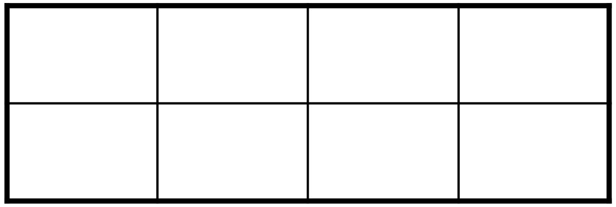
Example

Grammar:

A→S EOF   L → L ; S   S →(L)

L → S   S → id = num

Parsing Table



Input from lexer: (id = num ; id = num $\overbrace{\text{) EOF}}^{\text{yet to read}}$

State of parse so far: (L

REDUCE

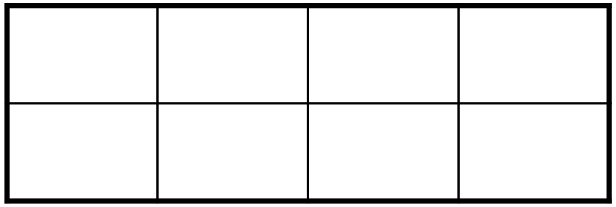
S → L ; S

Example

Grammar:

A→S EOF   L → L ; S   S →(L)
L → S   S → id = num

Parsing Table



Input from lexer: (id = num ; id = num) $\overbrace{\textbf{EOF}}^{\text{yet to read}}$

State of parse so far: (L)

SHIFT

Example

Grammar:

A → S EOF   L → L ; S   S → (L)
             L → S   S → id = num

Parsing Table

Input from lexer: (id = num ; id = num) $\overbrace{\text{EOF}}^{\text{yet to read}}$

State of parse so far: S

REDUCE
S → (L)

Example

Parsing Table

Grammar:

A → S EOF   L → L ; S   S → (L)

L → S   S → id = num

Input from lexer: (id = num ; id = num) EOF

State of parse so far: A

SHIFT

REDUCE

A → S EOF

ACCEPT

Example

Parsing Table

Grammar:

A → S EOF   L → L ; S   S → (L)
            L → S   S → id = num

Input from lexer: (id = num ; id = num) EOF

State of parse so far: A

A successful parse! **Is this grammar LL(1)?**

Bottom-up Parsing

- **Parser keeps track of**
  - ✓ **Position in current input (what input to read next)**
  - ✓ **A stack of terminal & non-terminal symbols representing the "parse so far"**

- **Based on next input symbol & stack, parser table indicates**
  - ✓ **shift: push next input on to top of stack**
  - ✓ **reduce R:**
    - ➢ **top of stack should match RHS of rule**
    - ➢ **replace top of stack with LHS of rule**
  - ✓ **error**
  - ✓ **accept (we shift EOF & can reduce what remains on stack to start symbol)**

LR PARSING ENGINE

How does the LR parser know **when to shift and when to reduce**?

Using a deterministic finite automaton ！

LR PARSING ENGINE

Shift-reduce algorithm (details)

- The parser summarizes the current “parse state” using an integer
  - ✓ The integer is actually a state in a finite automaton
  - ✓ The current parse state can be computed by running the automaton over the current parse stack

a :Accept;
Error :
(denoted by a blank entry in the table).

***sn* :Shift into state *n*;**
***gn* :Goto state *n*;**
***rk*: Reduce by rule *k*;**

	id	num	print	;	,	+	:=	(	)	$	*S*	*E*	*L*
1	s4		s7								g2		
2				s3						a			
3	s4		s7								g5		
4							s6						
5				r1	r1					r1			
6	s20	s10						s8				g11	
7								s9					
8	s4		s7								g12		
9	s20	s10						s8				g15	g14
10				r5	r5	r5			r5	r5			
11				r2	r2	s16				r2			
12				s3	s18								
13				r3	r3					r3			
14					s19				s13				
15					r8				r8				
16	s20	s10						s8				g17	
17				r6	r6	s16			r6	r6			
18	s20	s10						s8				g21	
19	s20	s10						s8				g23	
20				r4	r4	r4			r4	r4			
21									s22				
22				r7	r7	r7			r7	r7			
23					r9	s16			r9				

LR PARSING ENGINE

Shift-reduce algorithm (details)

- **Algorithm:** **Based on next input symbol & the parse state (as opposed to the entire stack), parser table indicates**

  Shift(n): **Advance input one token; push *n* on stack.**

  Reduce(k):

  - ✓ **Pop stack as many times as the number of symbols on the right-hand side of rule *k*;**
  - ✓ **Let *X* be the left-hand-side symbol of rule *k*;**
  - ✓ **In the state now on top of stack, look up *X* to get "goto *n*"; Push *n* on top of stack.**

  Accept: **Stop parsing, report success.**

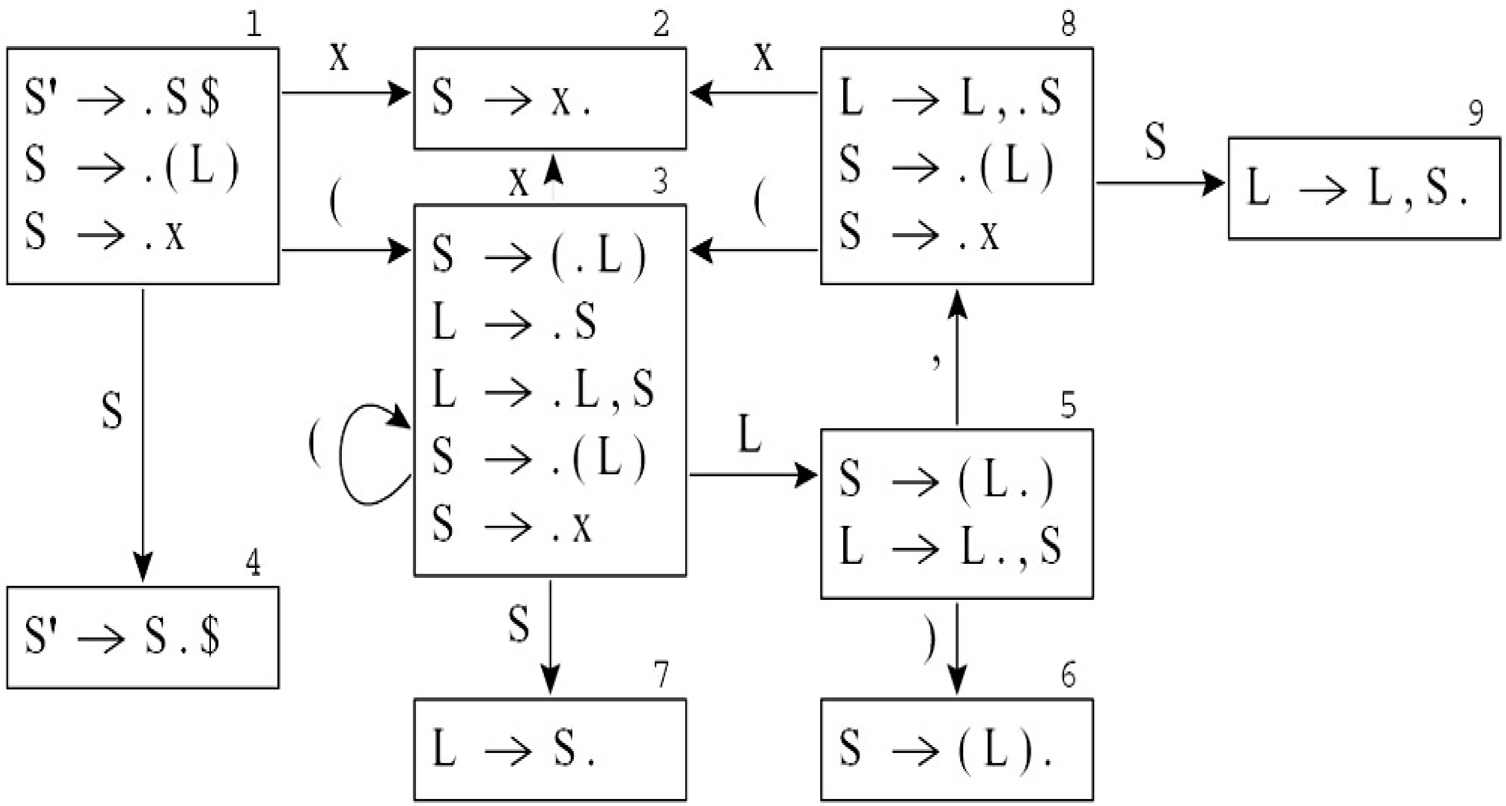
  Error: **Stop parsing, report failure.**

LR(0) Parser Generation

- **Making shift/reduce decisions without any lookahead.**

Items and States

$$S' \rightarrow S\$$$
$$S \rightarrow (L)$$
$$S \rightarrow x$$
$$L \rightarrow S$$
$$L \rightarrow L, S$$

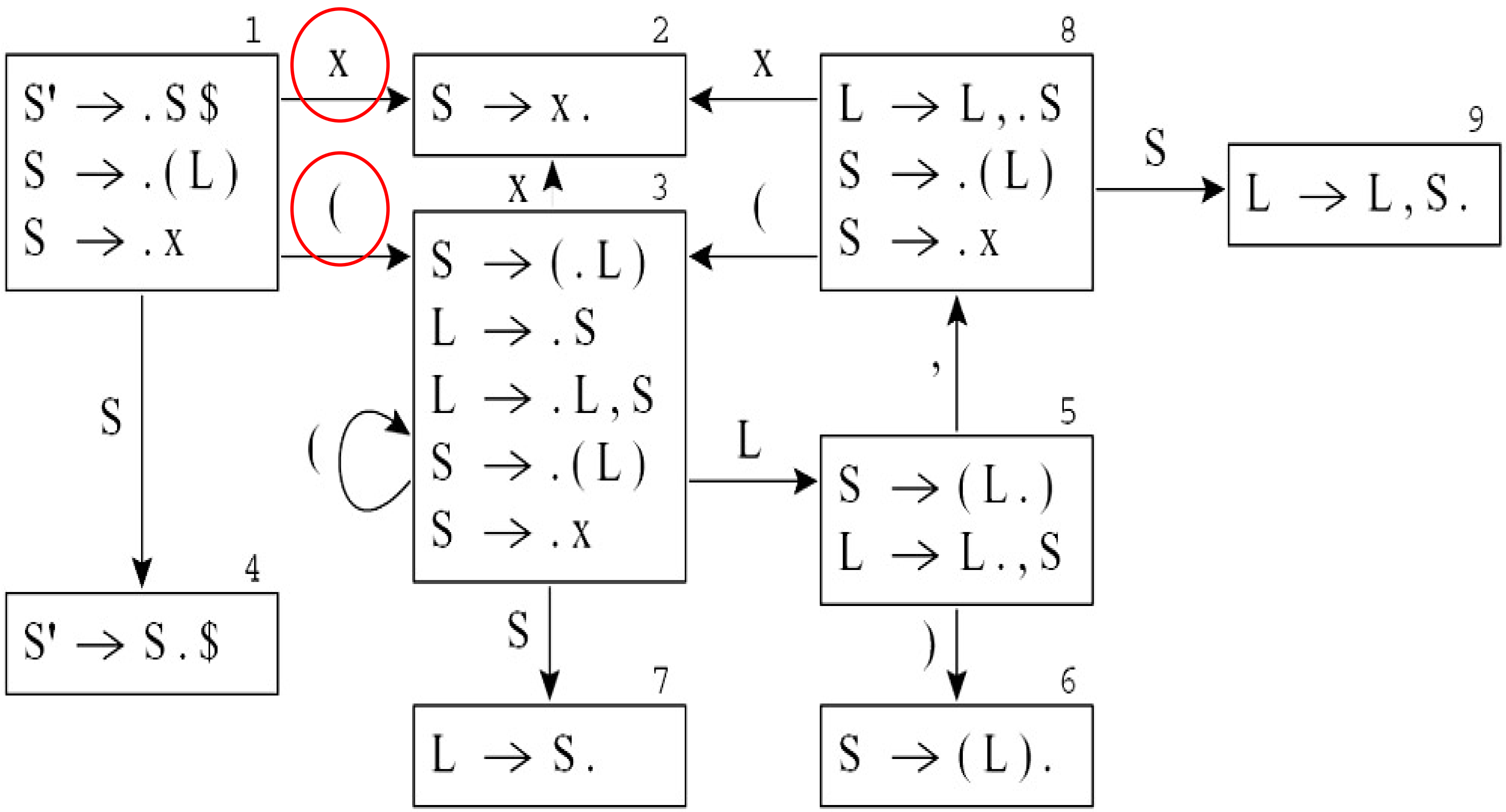


LR(0) Parser Generation

- **Making shift/reduce decisions** **without any lookahead.**

$S' \rightarrow S\$$
$S \rightarrow (L)$
$S \rightarrow x$
$L \rightarrow S$
$L \rightarrow L, S$

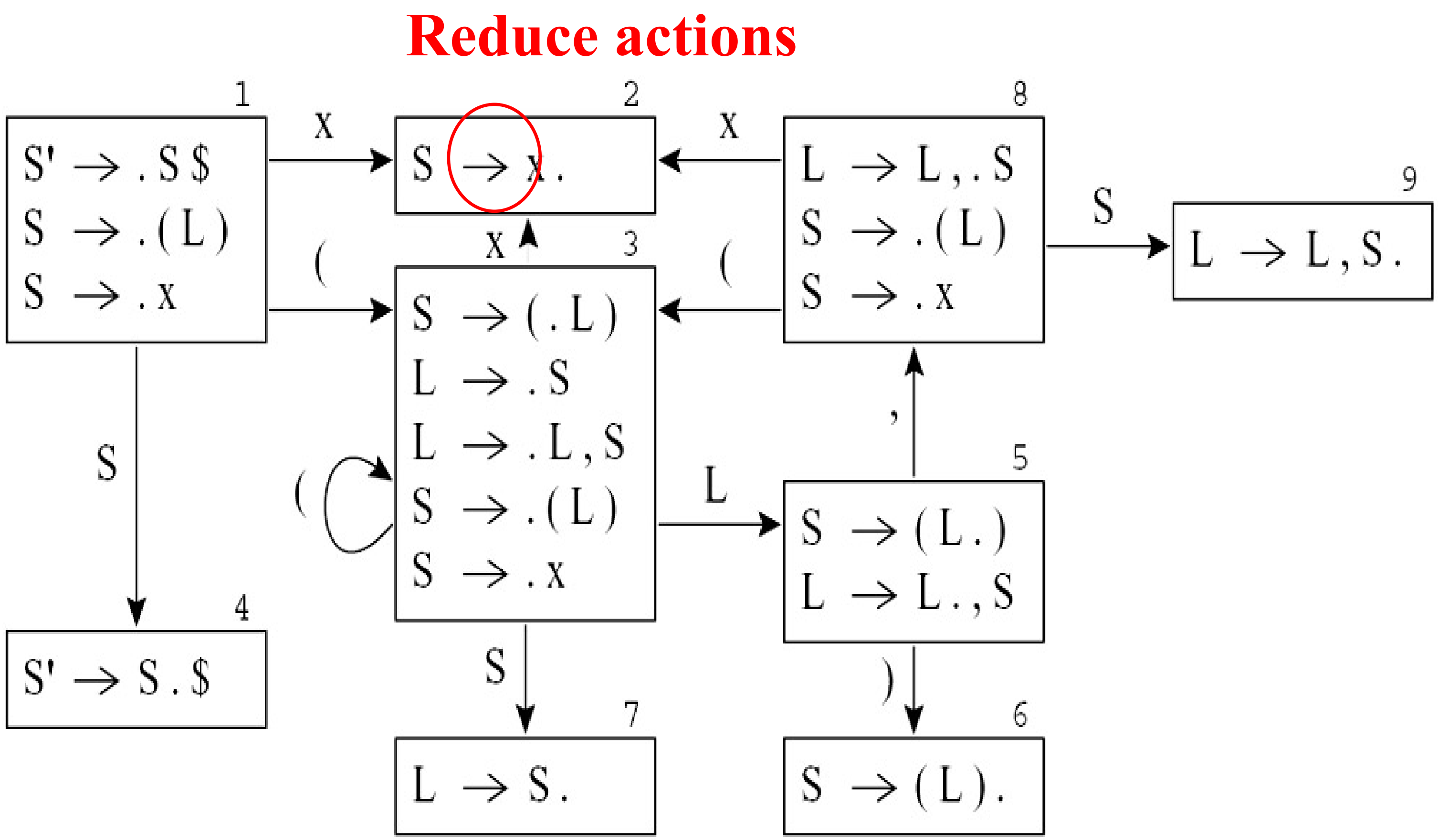
Shift actions



LR(0) Parser Generation

- Making shift/reduce decisions **without any lookahead.**

$S' \rightarrow S\$$
$S \rightarrow (L)$
$S \rightarrow x$
$L \rightarrow S$
$L \rightarrow L, S$

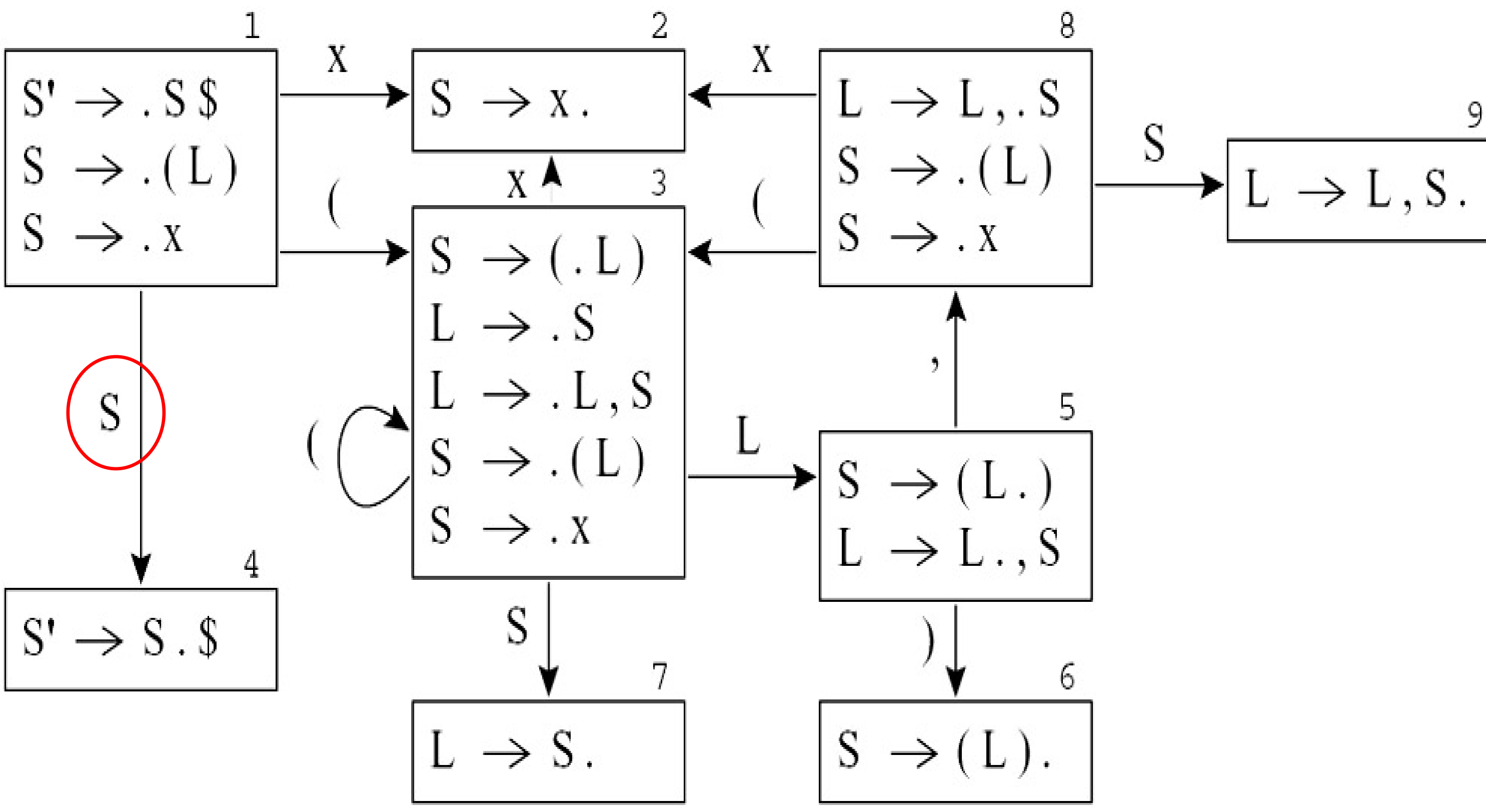


LR(0) Parser Generation

- **Making shift/reduce decisions** **without any lookahead.**

$$
\begin{aligned}
S' &\rightarrow S\$ \\
S &\rightarrow (L) \\
S &\rightarrow x \\
L &\rightarrow S \\
L &\rightarrow L, S
\end{aligned}
$$

Goto actions



LR(0) Parser Generation

Closure(I) =
repeat
for any item $A \rightarrow \alpha.X\beta$ in I
for any production $X \rightarrow \gamma$
$I \leftarrow I \cup \{X \rightarrow .\gamma\}$
until I does not change.
return I

	(	)	x	,	$	S	L
1	s3		s2			g4	
2	r2	r2	r2	r2	r2		
3	s3		s2			g7	g5
4					a		
5		s6		s8			
6	r1	r1	r1	r1	r1		
7	r3	r3	r3	r3	r3		
8	s3		s2			g9	
9	r4	r4	r4	r4	r4		

Items and States

Goto(I, X) =
set J to the empty set
for any item $A \rightarrow \alpha.X\beta$ in I
add $A \rightarrow \alpha X.\beta$ to J
return Closure(J)

LR(0) Parser Generation

The algorithm for LR(0) parser construction.

- Augment the grammar with an auxiliary start production $S' \rightarrow S\$$.
- T be the set of states seen so far
- E the set of (shift or goto) edges found so far.

Initialize T to {Closure({$S' \rightarrow .S\$$})}
Initialize E to empty.
repeat
  for each state I in T
    for each item $A \rightarrow \alpha.X\beta$ in I
      let J be Goto(I, X)
      $T \leftarrow T \cup \{J\}$
      $E \leftarrow E \cup \{I \xrightarrow{X} J\}$
until E and T did not change in this iteration

	(	)	x	,	\$	S	L
1	s3		s2			g4	
2	r2	r2	r2	r2	r2		
3	s3		s2			g7	g5
4					a		
5		s6		s8			
6	r1	r1	r1	r1	r1		
7	r3	r3	r3	r3	r3		
8	s3		s2			g9	
9	r4	r4	r4	r4	r4		

sn gk

LR(0) Parser Generation

The algorithm for LR(0) parser construction.

Compute set *R* of LR(0) **reduce actions:**

$R \leftarrow \{\}$

for each state I in T

for each item $A \rightarrow \alpha$

$R \leftarrow R \cup \{(I, A \rightarrow \alpha)\}$

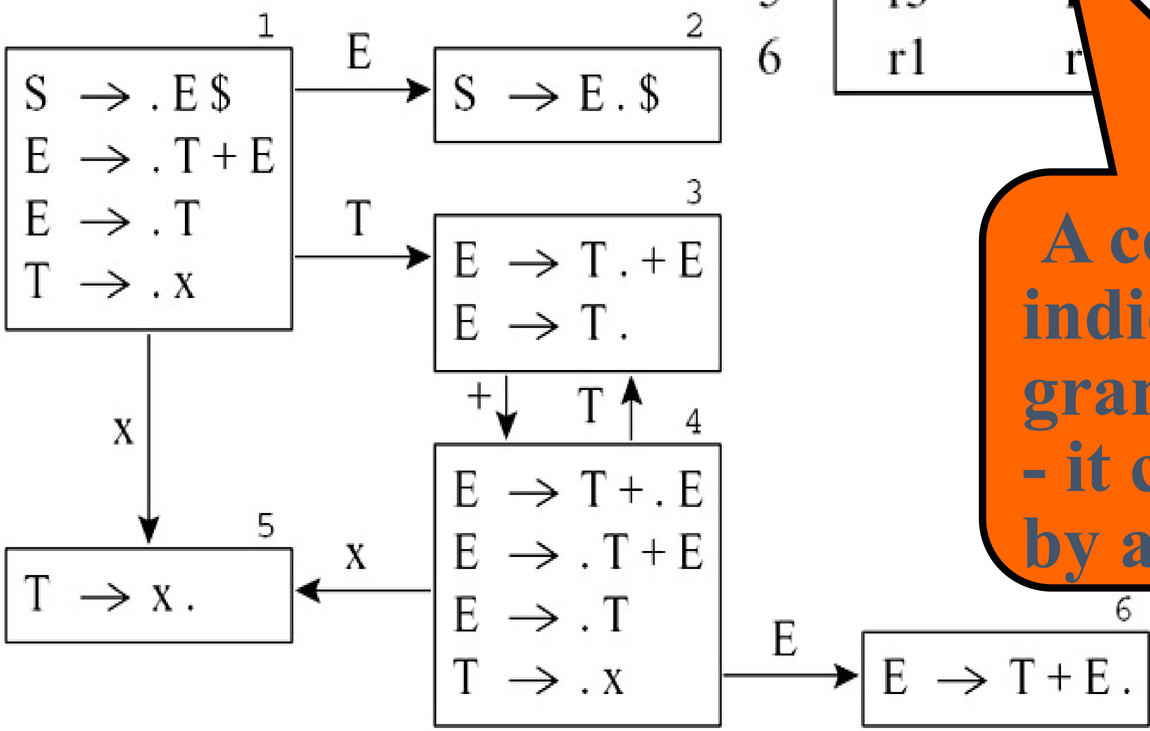
	(	)	x	,	$	S	L
1	s3		s2			g4	
2	r2	r2	r2	r2	r2		
3	s3		s2			g7	g5
4					a		
5		s6		s8			
6	r1	r1	r1	r1	r1		
7	r3	r3	r3	r3	r3		
8	s3		s2			g9	
9	r4	r4	r4	r4	r4		

ri

SLR Parser Generation

$S \rightarrow E\ \$$
$E \rightarrow T + E$
$E \rightarrow T$
$T \rightarrow x$

	x	+	$	*E*	*T*
1	s5			g2	g3
2			a		
3	r2	s4,r2	r2		
4	s5			g6	g3
5	r3	r	r3		
6	r1	r	r1		



A conflict and indicates that the grammar is not LR(0) - it cannot be parsed by an LR(0) parser.

SLR Parser Generation

- Put reduce actions into the table only where indicated **by the FOLLOW set.**

$R \leftarrow \{\}$
for each state I in T
  for each item $A \rightarrow \alpha.$ in I
    for each token X in FOLLOW(A)
      $R \leftarrow R \cup \{(I, X, A \rightarrow \alpha)\}$

	x	+	\$	E	T
1	s5			g2	g3
2			a		
3		s4	r2		
4	s5			g6	g3
5		r3	r3		
6			r1		

LR(1) Items

An LR(1) item

- Consists of *a grammar production*, a *right-hand-side position* (represented by the dot), and *a lookahead symbol*.

An item ($A \to \alpha.\beta$, x)

- Indicates that the sequence α is on top of the stack, and at the head of the input is a string derivable from βx.

Generate LR(1) Parsing Table

Closure(I) =
  repeat
    for any item ($A \to \alpha.X\beta$, z) in I
      for any production $X \to \gamma$
        for any $w \in$ FIRST(βz)
          $I \leftarrow I \cup \{(X \to .\gamma, w)\}$
  until I does not change
return I

The start state is the closure of the item ($S' \to .S\ \$$, ?)

$R \leftarrow \{\ \}$
for each state I in T
  for each item ($A \to \alpha.$, z) in I
    $R \leftarrow R \cup \{(I, z, A \to \alpha)\}$

Goto(I, X) =
  $J \leftarrow \{\}$
  for any item ($A \to \alpha.X\beta$, z) in I

    add ($A \to \alpha X.\beta$, z) to J
  return Closure(J).

Generte LR(1) Parsing Table

$S' \rightarrow .S\ \$$		?
$S \rightarrow .V = E$		\$
$S \rightarrow .E$		\$
$E \rightarrow .V$		\$
$V \rightarrow .\mathrm{x}$		\$
$V \rightarrow .*E$		\$
$V \rightarrow .\mathrm{x}$		=
$V \rightarrow .*E$		=

$S' \rightarrow .S\ \$$		?
$S \rightarrow .V = E$		\$
$S \rightarrow .E$		\$
$E \rightarrow .V$		\$
$V \rightarrow .\mathrm{x}$		\$, =
$V \rightarrow .*E$		\$, =

Closure(I) =
  repeat
    for any item ($A \rightarrow \alpha.X\beta$, z) in I
      for any production $X \rightarrow \gamma$
        for any $w \in$ FIRST(βz)
          $I \leftarrow I \cup \{(X \rightarrow .\gamma, w)\}$
  until I does not change
return I

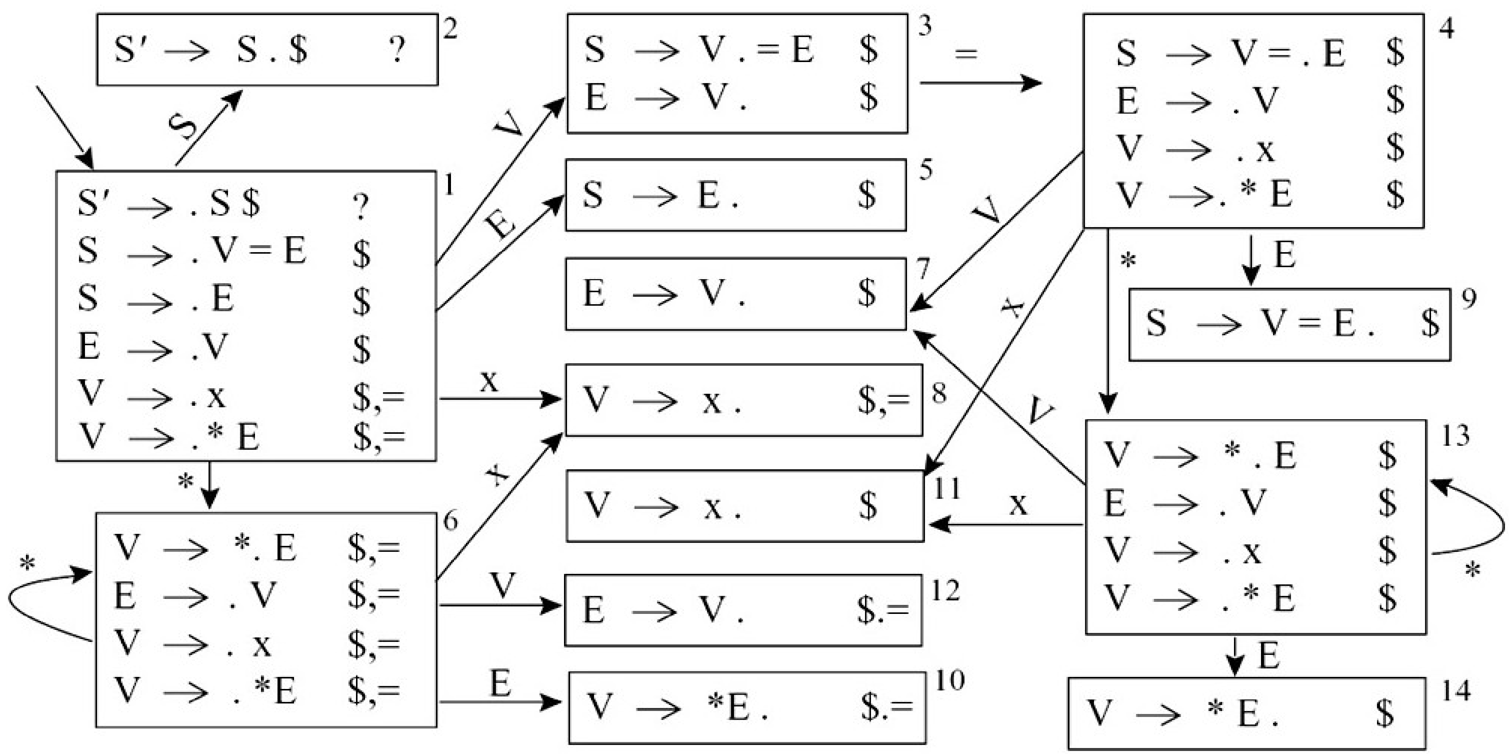
1. $S' \rightarrow S\ \$$
2. $S \rightarrow V = E$
3. $S \rightarrow E$
4. $E \rightarrow V$
5. $V \rightarrow \mathrm{x}$
6. $V \rightarrow *\ E$

Generate LR(1) Parsing Table

Goto(I, X) =
  $J \leftarrow \{\}$
  for any item $(A \rightarrow \alpha.X\beta, z)$ in I
    add $(A \rightarrow \alpha X.\beta, z)$ to J
  return Closure(J).

	x	*	=	\$	*S*	*E*	*V*
1	s8	s6			g2	g5	g3
2				a			
3			s4	r3			
4	s11	s13				g9	g7
5				r2			
6	s8	s6				g10	g12
7				r3			
8			r4	r4			
9				r1			
10			r5	r5			
11				r4			
12			r3	r3			
13	s11	s13				g14	g7
14				r5			

(a) LR(1)

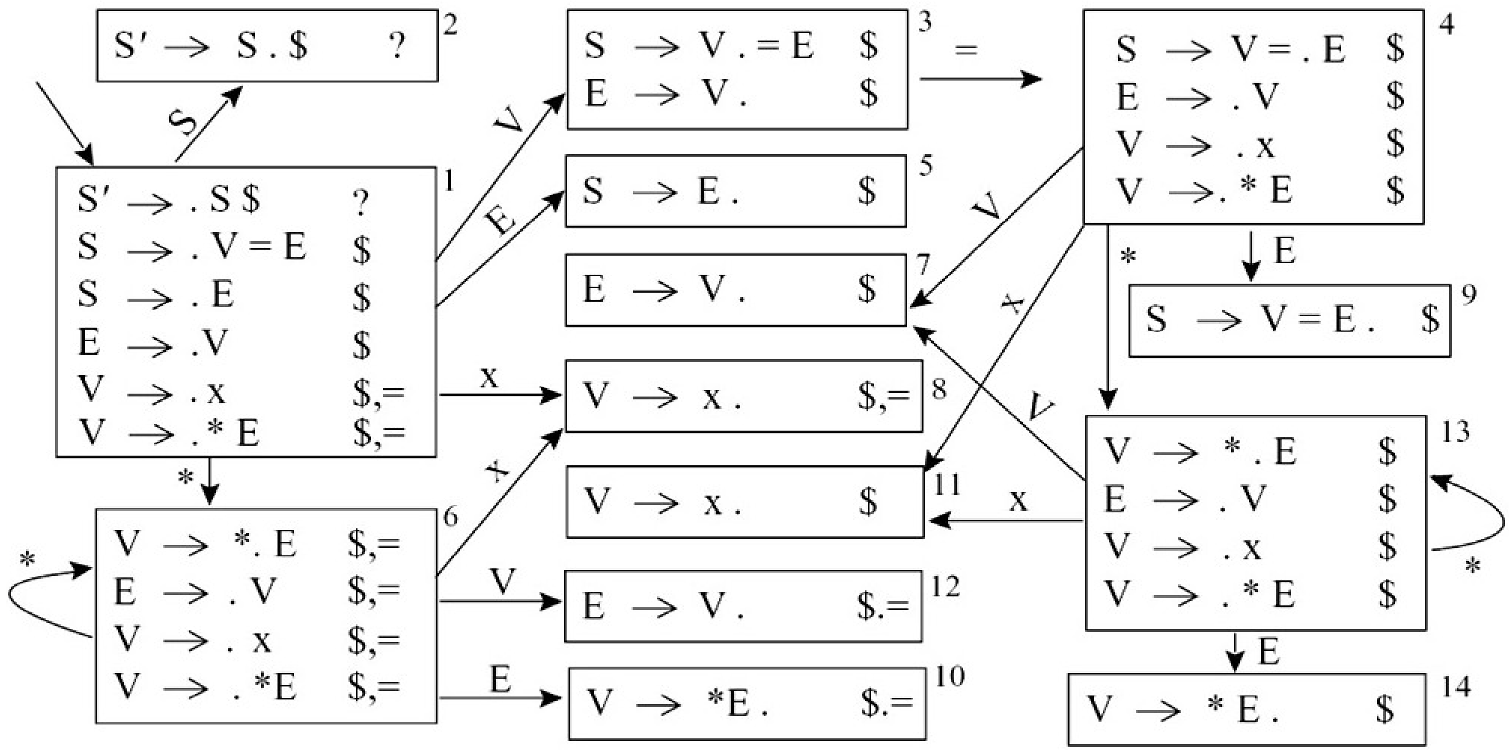


Generate LR(1) Parsing Table

$R \leftarrow \{\,\}$
for each state I in T
for each item $(A \rightarrow \alpha., z)$ in I
$R \leftarrow R \cup \{(I, z, A \rightarrow \alpha)\}$

	x	*	=	$	S	E	V
1	s8	s6			g2	g5	g3
2				a			
3			s4	r3			
4	s11	s13				g9	g7
5				r2			
6	s8	s6				g10	g12
7				r3			
8			r4	r4			
9				r1			
10			r5	r5			
11				r4			
12			r3	r3			
13	s11	s13				g14	g7
14				r5			

(a) LR(1)



An LR(1) Parsing Table

	x	*	=	$	S	E	V
1	s8	s6			g2	g5	g3
2				a			
3			s4	r3			
4	s11	s13				g9	g7
5				r2			
6	s8	s6				g10	g12
7				r3			
8			r4	r4			
9				r1			
10			r5	r5			
11				r4			
12			r3	r3			
13	s11	s13				g14	g7
14				r5			

(a) LR(1)

LALR(1) Parsing Tables

- LR(1) parsing tables can be very large, with many states.
- A smaller table can be made by merging any two states whose items are identical except for lookahead sets.
- The result parser is called an LALR(1) parser.

LALR(1) Parsing Tables

	x	*	=	$	S	E	V
1	s8	s6			g2	g5	g3
2				a			
3			s4	r3			
4	s11	s13				g9	g7
5				r2			
6	s8	s6				g10	g12
7				r3			
8			r4	r4			
9				r1			
10			r5	r5			
11				r4			
12			r3	r3			
13	s11	s13				g14	g7
14				r5			

(a) LR(1)

	x	*	=	$	S	E	V
1	s8	s6			g2	g5	g3
2				a			
3			s4	r3			
4	s8	s6				g9	g7
5				r2			
6	s8	s6				g10	g7
7			r3	r3			
8			r4	r4			
9				r1			
10			r5	r5			

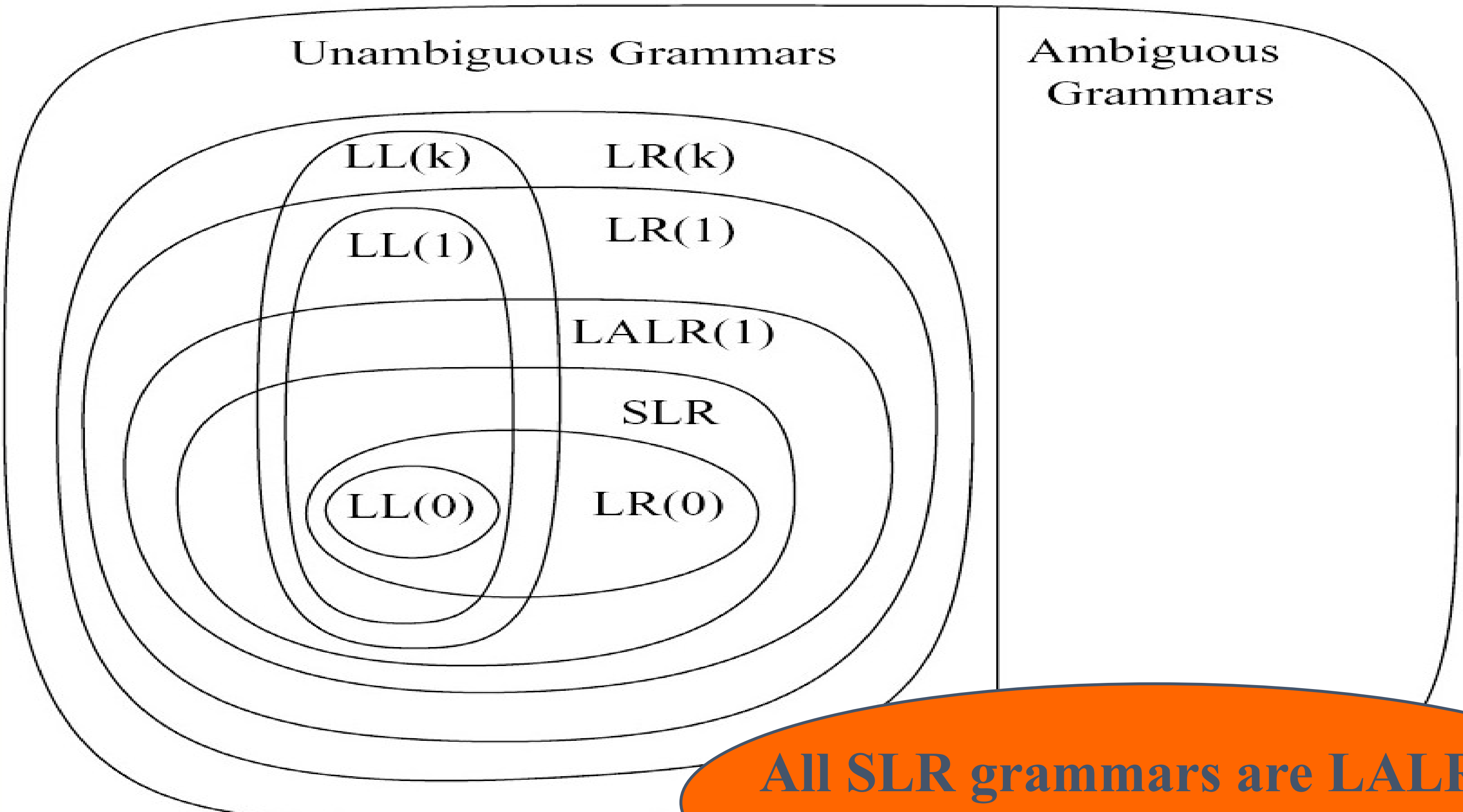
(b) LALR(1)

(6,13), (7,12), (8,11), (10,14): merged

- **For some grammars, the LALR(1) table contains reduce-reduce conflicts where the LR(1) table has none.**
- **In practice the difference matters little.**

Hierarchy of Grammar Classes

The relationship between several classes of grammars.



All SLR grammars are LALR(1) but not vice versa.

LR Parsing of Ambiguous Grammars

S → **if** *E* **then** *S* **else** *S*

S → **if** *E* **then** *S*

S → **other**

if a then if b then s1 else s2

(1) if a then { if b then s1 else s2 }

(2) if a then { if b then s1 } else s2

A shift-reduce conflict:

$S \to$ if E then S .	else
$S \to$ if E then S . else S	(*any*)

LR Parsing of Ambiguous Grammars

- **The grammar unchanged. In constructing the parsing table this conflict should be resolved by shifting(prefer interpretation (1))**

- **The ambiguity can be eliminated by introducing auxiliary nonterminals *M***

$$S \rightarrow M$$
$$S \rightarrow U$$
$$M \rightarrow \textbf{if } E \textbf{ then } M \textbf{ else } M$$
$$M \rightarrow \textbf{other}$$
$$U \rightarrow \textbf{if } E \textbf{ then } S$$
$$U \rightarrow \textbf{if } E \textbf{ then } M \textbf{ else } U$$

M **:for *matched statement***

U **:for *unmatched statement***

The end of Chapter 3(3)