CSCI 742 - Compiler Construction

Lecture 20

Parsing Wrap-up

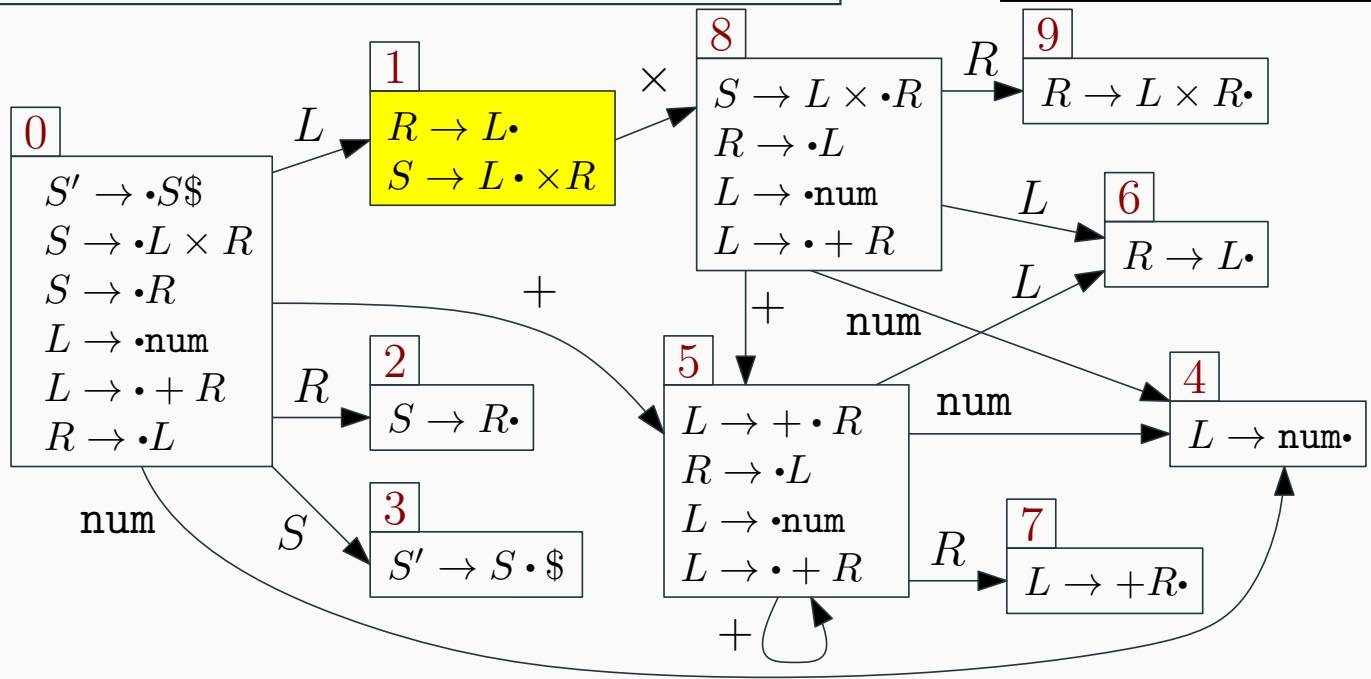
Instructor: Hossein Hojjat

March 10, 2017

LR(0) Automaton Example

	+	×	num	$	S	R	L
0	$s5$		$s4$		$g3$	$g2$	$g1$
1	$r5$	$r5/s8$	$r5$	$r5$			
2	$r2$	$r2$	$r2$	$r2$			
3				accept			
4	$r3$	$r3$	$r3$	$r3$			
5	$s5$		$s4$			$g7$	$g6$
6	$r5$	$r5$	$r5$	$r5$			
7	$r4$	$r4$	$r4$	$r4$			
8	$s5$		$s4$			$g9$	$g6$
9	$r1$	$r1$	$r1$	$r1$			

r1	$S \to L \times R$
r2	$S \to R$
r3	$L \to \text{num}$
r4	$L \to +R$
r5	$R \to L$

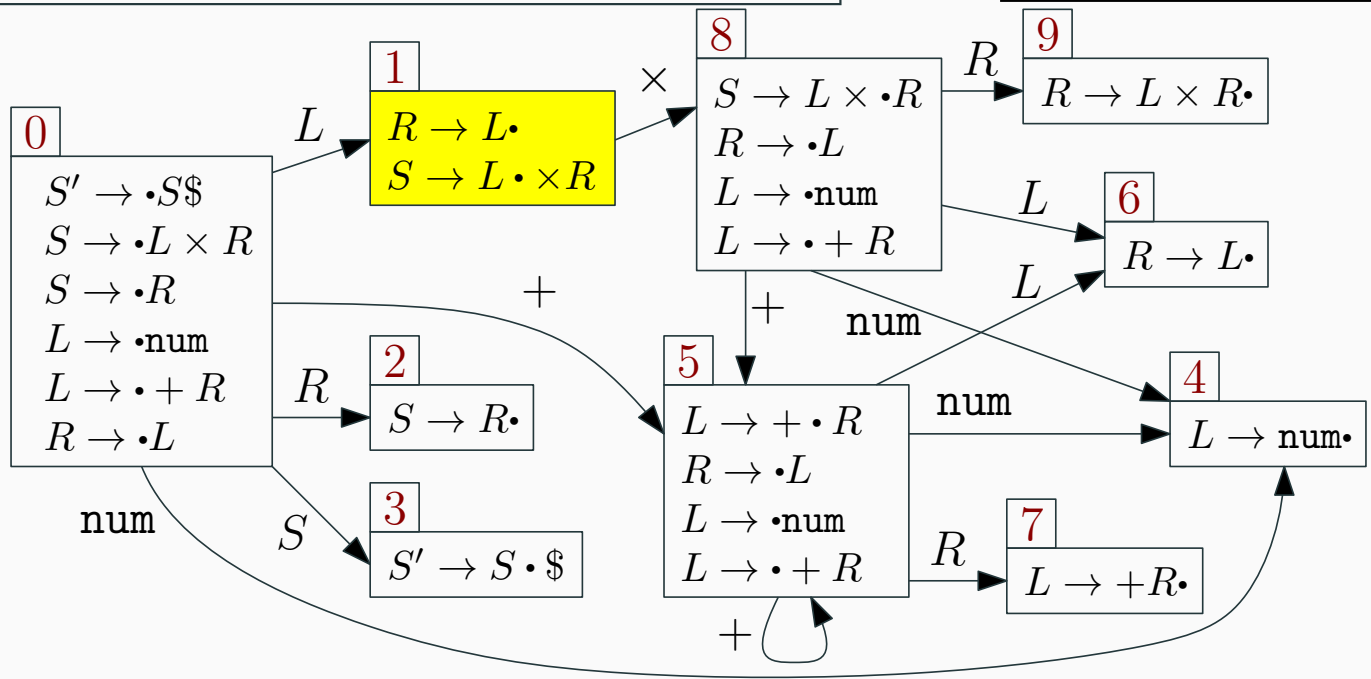


SLR Automaton Example

	+	×	num	$	S	R	L
0	$s5$		$s4$		$g3$	$g2$	$g1$
1		$r5/s8$		$r5$			
2				$r2$			
3				accept			
4		$r3$		$r3$			
5	$s5$		$s4$			$g7$	$g6$
6		$r5$		$r5$			
7		$r4$		$r4$			
8	$s5$		$s4$			$g9$	$g6$
9				$r1$			

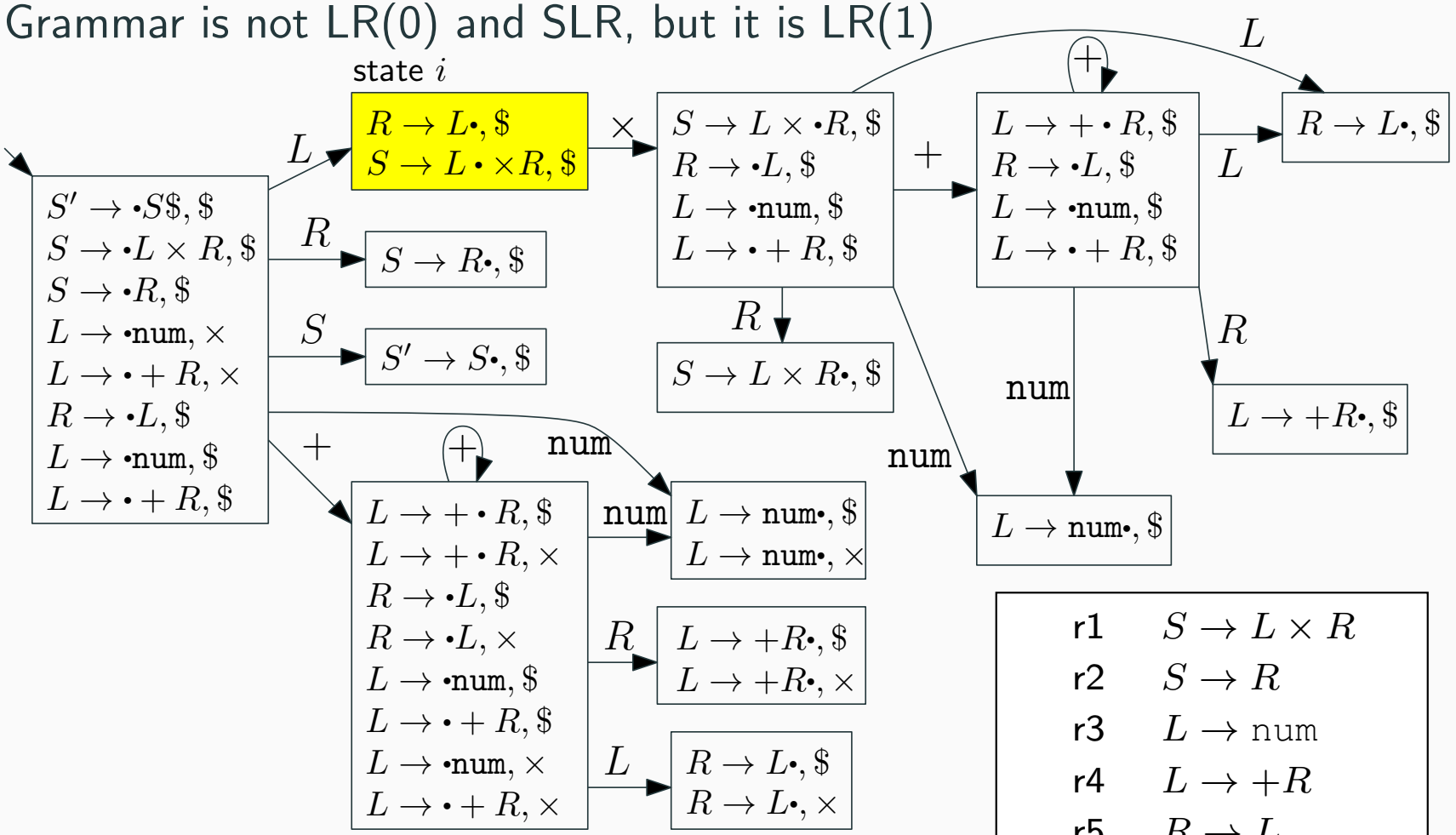
$\text{FOLLOW}(S) = \$$
$\text{FOLLOW}(L) =$
$\text{FOLLOW}(R) = \{\times, \$\}$

r1	$S \to L \times R$
r2	$S \to R$
r3	$L \to \text{num}$
r4	$L \to +R$
r5	$R \to L$



LR(1) Automaton Example

Grammar is not LR(0) and SLR, but it is LR(1)



There is no more shift/reduce conflict in the automaton:

	+	×	num	$	S	R	L
state i		$s8$		$r(R \to L)$			

LALR

- Drawback: LR(1) parse engine has a large number of states
- LALR (Look-Ahead LR parser): Simple technique to eliminate states
- If two LR(1) states are identical except for the look ahead symbol of their items, merge them
- Result is LALR(1) DFA
- It is more memory efficient, typically merges several LR(1) states
- May also have more reduce/reduce conflicts
- Power of LALR parsing is enough for many mainstream computer languages
- Several automatic parser generators such as Yacc or GNU Bison

LALR States

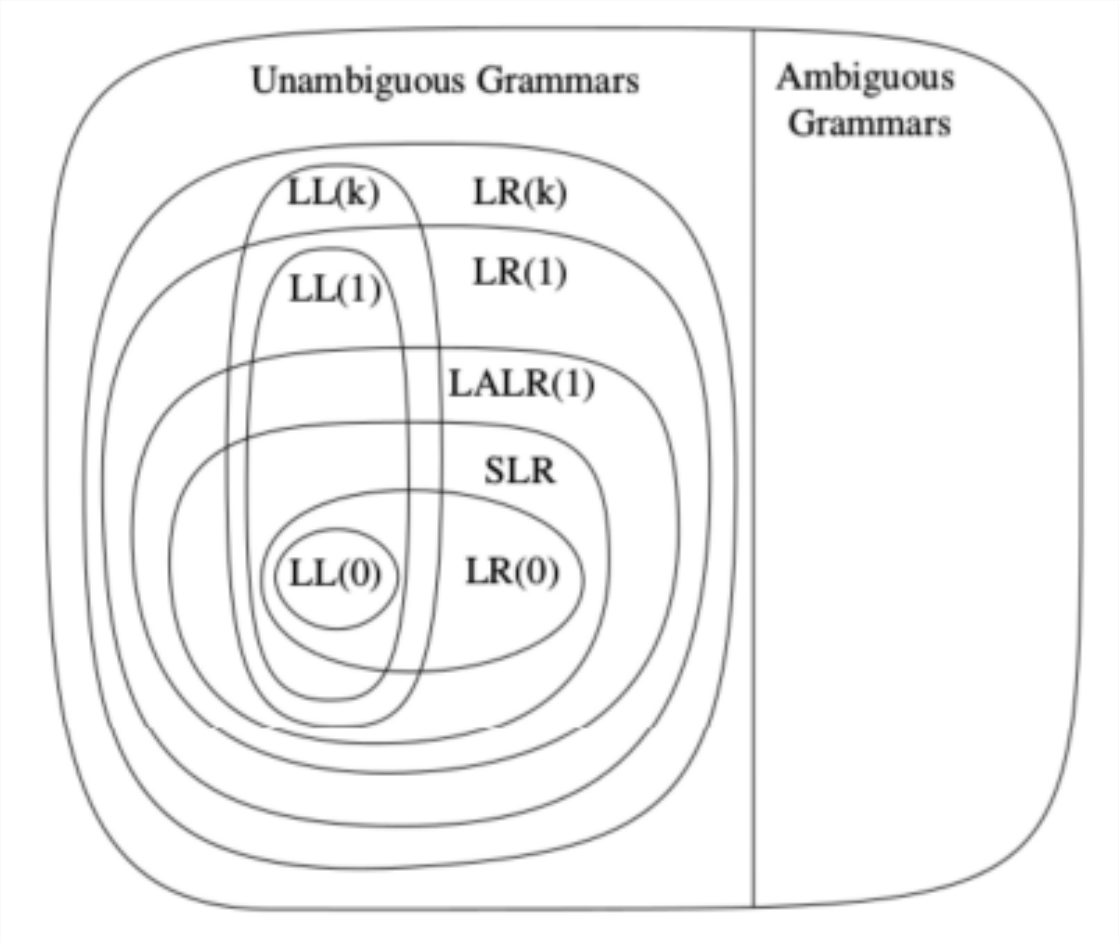
- Consider for example these two LR(1) states

$$\boxed{\begin{array}{l} X \to \alpha\bullet, a \\ Y \to \beta\bullet, c \end{array}} \qquad \boxed{\begin{array}{l} X \to \alpha\bullet, b \\ Y \to \beta\bullet, d \end{array}}$$

- They will be merged into the following LALR(1) states

$$\boxed{\begin{array}{l} X \to \alpha\bullet, \{a, b\} \\ Y \to \beta\bullet, \{c, d\}] \end{array}}$$

Hierarchy of Grammar Classes



"Modern Compiler Implementation in Java",
Andrew W. Appel, Jens Palsberg

Parsing

- Task of a parser:
  find a derivation of a string in given a context-free grammar
- CYK recognizes languages defined by context-free grammars
  - cubic time $O(n^3)$
- Restricted forms of CFG can be parsed in linear time:
  - LL (left to right, left-most derivation)
  - LR (left to right, reverse right-most derivation)
- Simple top-down parser: LL(1)
  - Basic recursive-descent implementation
- More powerful parser: LR(1), bottom-up
- An efficiency hack on top of LR(1): LALR(1)

What to expect next?

- Is “x” an array, integer or a function? Is it declared?
- Is the expression “x + z” type-consistent?
- In “x[i]”, is “x” an array? Does it have the correct number of dimensions?
- Where can “x” be stored? (register, local, global, heap, static)
- How many arguments does “f()” take? What about “printf ()” ?


KEEP
CALM
AND
HAVE A SAFE
SPRING BREAK