CS 4300: Compiler Theory

Chapter 4
Syntax Analysis

Dr. Xuejun Liang

Outlines (Sections)

1. Introduction
2. Context-Free Grammars
3. Writing a Grammar
4. Top-Down Parsing
5. Bottom-Up Parsing
6. Introduction to LR Parsing: Simple LR
7. More Powerful LR Parsers
8. Using Ambiguous Grammars
9. Parser Generators

Quick Review of Last Lecture

- SLR: Simple extension of LR(0) shift-reduce parsing
  - Ambiguity and Conflicts
  - Viable Prefixes and valid items for a viable prefix
- LR(1) Grammars
  - LR(1) item
  - Closure Operation for LR(1) Items
  - Goto Operation for LR(1) Items
  - Constructing the set of LR(1) Items of a Grammar
  - Constructing Canonical LR(1) Parsing Tables

LALR Parsing

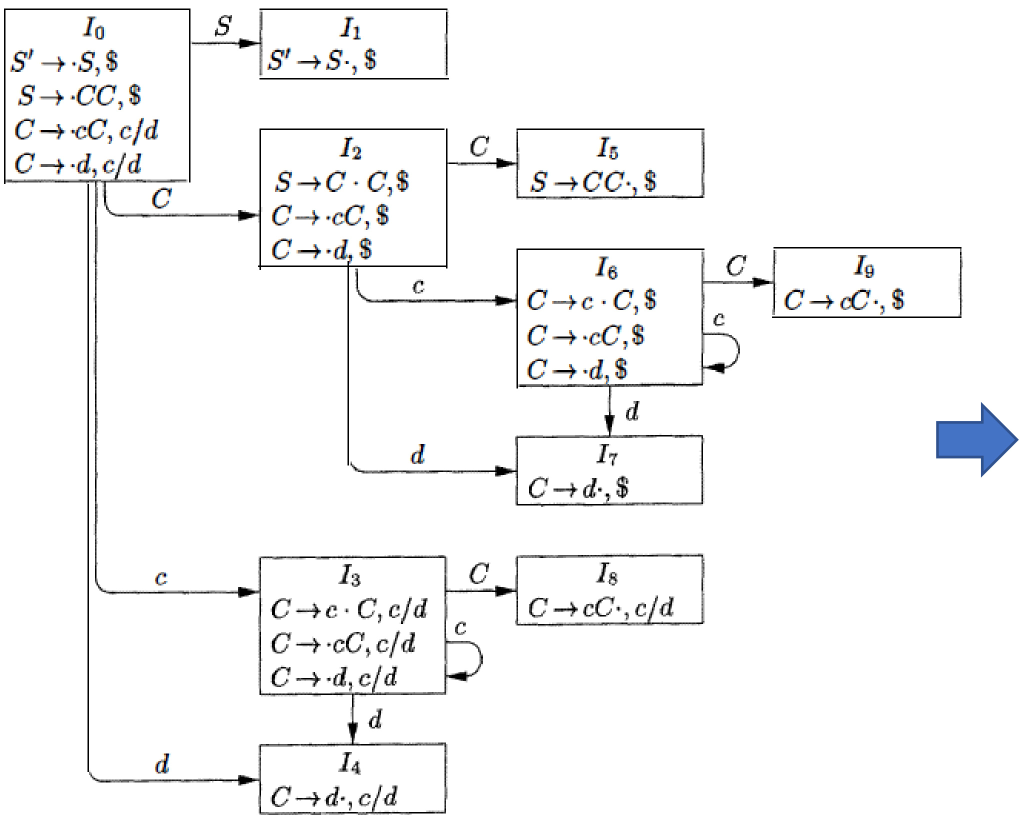
- LR(1) parsing tables have many states
- LALR parsing (Look-Ahead LR) merges two or more LR(1) state into one state to reduce table size
- Less powerful than LR(1)
  - Will not introduce shift-reduce conflicts, because shifts do not use lookaheads
  - May introduce reduce-reduce conflicts, but seldom do so for grammars of programming languages

Constructing LALR Parsing Tables (1)

1. Construct sets of LR(1) items: $I_1, I_2, \ldots, I_n$
2. Combine LR(1) sets with sets of items that share the same first part: $J_1, J_2, \ldots, J_m$

Grammar:

0. $S' \to S$
1. $S \to C\ C$
2. $C \to \boldsymbol{c}\ C$
3. $C \to \boldsymbol{d}$



I_{36}: $C \to c{\cdot}C,\ c/d/\$$
$C \to {\cdot}cC,\ c/d/\$$
$C \to {\cdot}d,\ c/d/\$$

I_{47}: $C \to d{\cdot},\ c/d/\$$

I_{89}: $C \to cC{\cdot},\ c/d/\$$

Constructing LALR Parsing Tables (2)

3. Build parsing action in the same way
4. Build goto: If $J_k = I_p \cup I_q$, then $J_{goto(k, X)} = I_{goto(p, X)} \cup I_{goto(q, X)}$.

Grammar:
0. $S' \rightarrow S$
1. $S \rightarrow C\ C$
2. $C \rightarrow \boldsymbol{c}\ C$
3. $C \rightarrow \boldsymbol{d}$

STATE	ACTION			GOTO	
	c	*d*	$	*S*	*C*
0	s3	s4		1	2
1			acc		
2	s6	s7			5
3	s3	s4			8
4	r3	r3			
5			r1		
6	s6	s7			9
7			r3		
8	r2	r2			
9			r2		

STATE	ACTION			GOTO	
	c	*d*	$	*S*	*C*
0	s36	s47		1	2
1			acc		
2	s36	s47			5
36	s36	s47			89
47	r3	r3	r3		
5			r1		
89	r2	r2	r2		

Another Example Grammar and LALR Parsing Table

- Unambiguous LR(1) grammar:

$$S \rightarrow L = R$$
$$\quad | \; R$$
$$L \rightarrow * \; R$$
$$\quad | \; \mathbf{id}$$
$$R \rightarrow L$$

- Augment with $S' \rightarrow S$
- LALR items (next slide)

Grammar:

1. $S' \rightarrow S$
2. $S \rightarrow L = R$
3. $S \rightarrow R$
4. $L \rightarrow * \; R$
5. $L \rightarrow \mathbf{id}$
6. $R \rightarrow L$

Constructing LALR Parsing Tables

1. Construct sets of LR(1) items
2. Combine LR(1) sets with sets of items that share the same first part

I_4: $[L \to *\bullet R,\ =]$
$[R \to \bullet L,\ =]$
$[L \to \bullet *R,\ =]$
$[L \to \bullet \mathbf{id},\ =]$

I_{11}: $[L \to *\bullet R,\ \$]$
$[R \to \bullet L,\ \$]$
$[L \to \bullet *R,\ \$]$
$[L \to \bullet \mathbf{id},\ \$]$

I_4:
$[L \to *\bullet R,\ =/\$]$
$[R \to \bullet L,\ =/\$]$
$[L \to \bullet *R,\ =/\$]$
$[L \to \bullet \mathbf{id},\ =/\$]$

Shorthand for two items in the same set

Constructing LALR Parsing Tables

1. Construct sets of LR(1) items
2. Combine LR(1) sets with sets of items that share the same first part

I_5: $[L \rightarrow \mathbf{id}\bullet, =]$
I_{12}: $[L \rightarrow \mathbf{id}\bullet, \$]$
$\}$ I_5: $[L \rightarrow \mathbf{id}\bullet, =/\$]$

I_7: $[L \rightarrow *R\bullet, =]$
I_{13}: $[L \rightarrow *R\bullet, \$]$
$\}$ I_7: $[L \rightarrow *R\bullet, =/\$]$

I_8: $[R \rightarrow L\bullet, =]$
I_{10}: $[R \rightarrow L\bullet, \$]$
$\}$ I_8: $[R \rightarrow L\bullet, =/\$]$

I_0: $[S' \to \bullet S,\ \mathbf{\$}]$ goto$(I_0,S)=I_1$
$[S \to \bullet L\mathbf{=}R,\ \mathbf{\$}]$ goto$(I_0,L)=I_2$
$[S \to \bullet R,\ \mathbf{\$}]$ goto$(I_0,R)=I_3$
$[L \to \bullet\mathbf{*}R,\ \mathbf{=}]$ goto$(I_0,\mathbf{*})=I_4$
$[L \to \bullet\mathbf{id},\ \mathbf{=}]$ goto$(I_0,\mathbf{id})=I_5$
$[R \to \bullet L,\ \mathbf{\$}]$

I_1: $[S' \to S\bullet,\ \mathbf{\$}]$ goto$(I_1,\mathbf{\$})=$acc

I_2: $[S \to L\bullet\mathbf{=}R,\ \mathbf{\$}]$ goto$(I_2,\mathbf{=})=I_6$
$[R \to L\bullet,\ \mathbf{\$}]$

I_3: $[S \to R\bullet,\ \mathbf{\$}]$

I_4: $[L \to \mathbf{*}\bullet R,\ \mathbf{=/\$}]$ goto$(I_4,R)=I_7$
$[R \to \bullet L,\ \mathbf{=/\$}]$ goto$(I_4,L)=I_9$
$[L \to \bullet\mathbf{*}R,\ \mathbf{=/\$}]$ goto$(I_4,\mathbf{*})=I_4$
$[L \to \bullet\mathbf{id},\ \mathbf{=/\$}]$ goto$(I_4,\mathbf{id})=I_5$

I_5: $[L \to \mathbf{id}\bullet,\ \mathbf{=/\$}]$

I_6: $[S \to L\mathbf{=}\bullet R,\ \mathbf{\$}]$ goto$(I_6,R)=I_8$
$[R \to \bullet L,\ \mathbf{\$}]$ goto$(I_6,L)=I_9$
$[L \to \bullet\mathbf{*}R,\ \mathbf{\$}]$ goto$(I_6,\mathbf{*})=I_4$
$[L \to \bullet\mathbf{id},\ \mathbf{\$}]$ goto$(I_6,\mathbf{id})=I_5$

I_7: $[L \to \mathbf{*}R\bullet,\ \mathbf{=/\$}]$

I_8: $[R \to L\bullet,\ \mathbf{=/\$}]$

I_9: $[S \to L\mathbf{=}R\bullet,\ \mathbf{\$}]$

LR(1) Parsing Table

	id	*	=	**$**	*S*	*L*	*R*
0	s5	s4			1	2	3
1				acc			
2			s6	r6			
3				r3			
4	s5	s4				8	7
5			r5	r5			
6	s12	s11				10	9
7			r4	r4			
8			r6	r6			
9				r2			
10				r6			
11	s12	s11				10	13
12				r5			
13				r4			

LALR(1) Parsing Table

	id	*	=	**$**	*S*	*L*	*R*
0	s5	s4			1	2	3
1				acc			
2			s6	r6			
3				r3			
4	s5	s4				8	7
5			r5	r5			
6	s5	s4				8	9
7			r4	r4			
8			r6	r6			
9				r2			

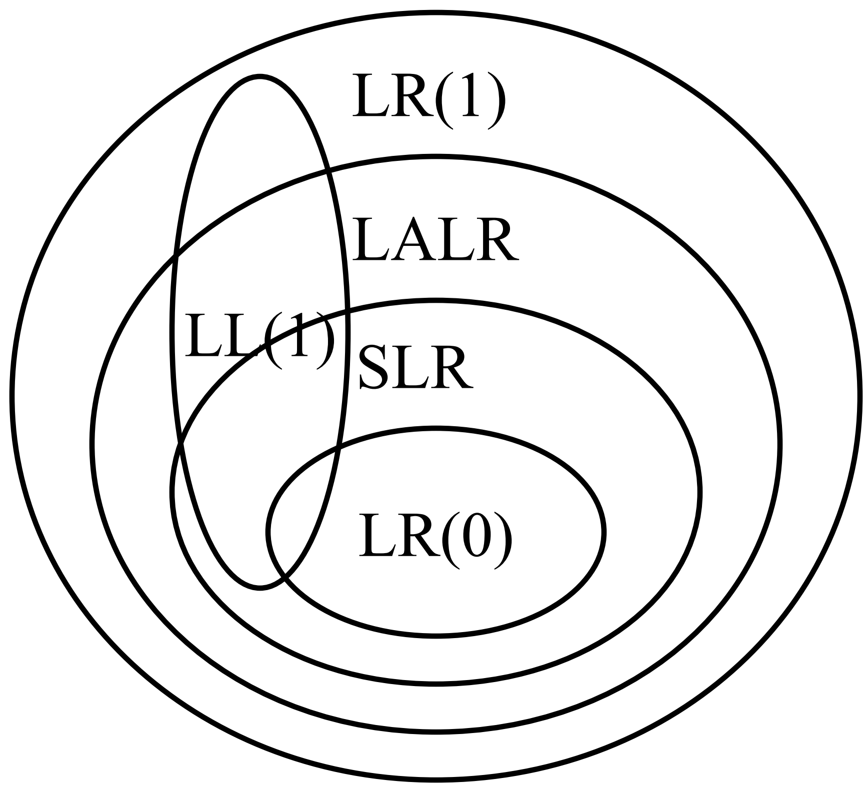
Grammar:

1. $S' \to S$
2. $S \to L = R$
3. $S \to R$
4. $L \to * R$
5. $L \to \textbf{id}$
6. $R \to L$

LL, SLR, LR, LALR Summary

- LL parse tables computed using FIRST/FOLLOW
  - Nonterminals $\times$ terminals $\rightarrow$ productions
  - Computed using FIRST/FOLLOW
- LR parsing tables computed using closure/goto
  - LR states $\times$ terminals $\rightarrow$ shift/reduce actions
  - LR states $\times$ nonterminals $\rightarrow$ goto state transitions
- A grammar is
  - LL(1) if its LL(1) parse table has no conflicts
  - SLR if its SLR parse table has no conflicts
  - LALR if its LALR parse table has no conflicts
  - LR(1) if its LR(1) parse table has no conflicts

LL, SLR, LR, LALR Grammars



8. Dealing with Ambiguous Grammars

1. $S' \rightarrow E$
2. $E \rightarrow E + E$
3. $E \rightarrow$ **id**

	id	+	**$**	*E*
0	s2			1
1		s3	acc	
2		r3	r3	
3	s2			4
4		s3/r2	r2	

Shift/reduce conflict:
action[4,+] = shift 4
action[4,+] = reduce $E \rightarrow E + E$

stack	symbols	input
0	**$**	**id+id+id$**
…		…
0 1 3 4	**$***E*+*E*	**+id$**

When shifting on +:
yields right associativity
id+(id+id)

When reducing on +:
yields left associativity
(id+id)+id

Using Associativity and Precedence to Resolve Conflicts

- Left-associative operators: reduce
- Right-associative operators: shift
- Operator of higher precedence on stack: reduce
- Operator of lower precedence on stack: shift

$S' \rightarrow E$
$E \rightarrow E + E$
$E \rightarrow E * E$
$E \rightarrow \mathbf{id}$

stack	symbols	input	
0	**$**	**id*id+id$**	
…		…	
0 1 3 5	$E*E	**+id$**	reduce $E \rightarrow E*E$
≈	≈	≈	≈

Error Detection in LR Parsing

- An LR parser will detect an error when it consults the parsing action table and finds an error entry.
- Canonical LR parser uses full LR(1) parse tables and will never make a single reduction before announcing the error when a syntax error occurs on the input
- SLR and LALR may still reduce when a syntax error occurs on the input, but will never shift the erroneous input symbol

Error Recovery in LR Parsing

- Panic mode
  - Pop until state with a goto on a nonterminal *A* is found, (where *A* represents a major programming construct), push *A*
  - Discard input symbols until one is found in the FOLLOW set of *A*
- Phrase-level recovery
  - Implement error routines for every error entry in table
- Error productions
  - Pop until state has error production, then shift on stack
  - Discard input until symbol is encountered that allows parsing to continue