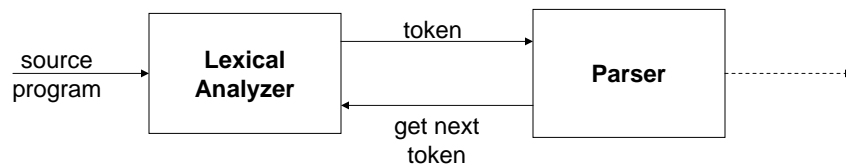


LR Parsing

Lexical Analyzer and Parser



Shift-Reduce or LR(k) Parsing

- Bottom-up parsing technique
- LR(k) parsing
 - L – scan the input from left-to-right
 - R – construct a right-most derivation
 - k – number of look-ahead symbols needed
- Called a shift-reduce parsing method
 - Shift and reduce are the major actions done by the parser

LR(k) Parsers

- Advantages
 - Can be constructed for virtually all programming language constructs for which a CFG can be written
 - Is more general than other parsers
 - Better than LL(k) parsers
 - Can detect syntactic errors as soon as possible on a left-to-right scan of the input
- Disadvantages
 - Too much work to implement by hand; you must use a parser generator

LR(k) Parsers

- Parser generator just constructs the parse table and we are ready to go
- Types of parse tables:
 - SLR(1) – simple LR(1) – uses LR(0) items
 - LALR(1) – look-ahead LR – uses LR(1) items compacted
 - LR(1) – most powerful – uses LR(1) items
- The parsing algorithm is always the same

LR(k) Grammars

- Let G be some grammar with start symbol S and consider a right-most derivation:
$$S \Rightarrow \alpha_1 \Rightarrow \alpha_2 \Rightarrow \dots \Rightarrow \omega$$
- Now consider a typical step in the derivation as follows: $\alpha A \gamma \Rightarrow \alpha \beta \gamma$ where $A \rightarrow \beta$ is a production used in this step, and $\alpha \beta \gamma$ is one of the α^i or ω itself
- We say G is LR(k) if, for every such derivation and derivation step, the production can be inferred by scanning $\alpha \beta$ and (at most) the first k symbols of γ

Why LR(k) Grammars are Useful

- The parser know when to cease scanning given a sentential form $\alpha\beta\gamma$, i.e., it can detect the boundary between β and γ
- The parser is able to identify the handle β
- The parser is able to uniquely select a production $A \rightarrow \beta$ that corresponds to the handle and to this sentential form
 - Grammars can be LR(k) and yet have productions $A \rightarrow \beta$, $B \rightarrow \beta$ with the same right-hand side
- The parser knows when to stop

CS 5300 - SJAllan

7

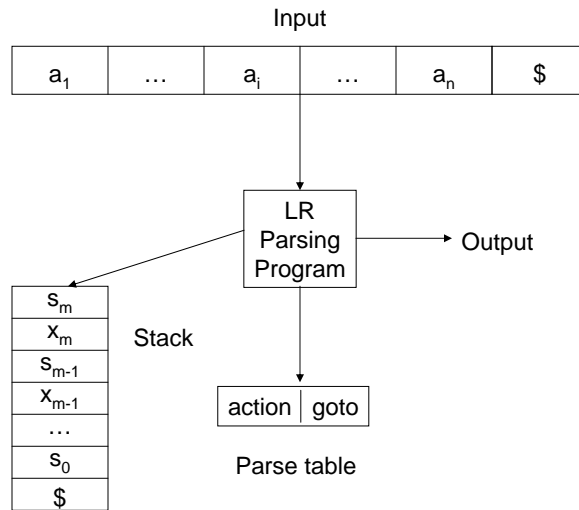
LR(K) (Shift-Reduce) Parsing

- Parsing performed by a finite state machine
- Parsing algorithm is language-independent
- FSM driven by table (s) generated automatically from grammar
- Language \longrightarrow Generator \longrightarrow Tables

CS 5300 - SJAllan

8

Model of LR Parser



CS 5300 - SJAllan

9

LR Parsing Algorithm

```
set ip to point to the first symbol in  $\omega\$$ 
initialize stack to  $s_0$ 
repeat forever
  let s be topmost state on stack
  let a be symbol pointed to by ip
  if action[s,a] = shift s'
    push a then s' onto stack
    advance ip to next input symbol
  else if action[s,a] = reduce  $A \rightarrow B$ 
    pop  $2*|B|$  symbols of stack
    let s' be state now on top of stack
    push A then goto[s',A] onto stack
    output production  $A \rightarrow B$ 
  else if action[s,a] == accept
    return success
  else
    error()
```

CS 5300 - SJAllan

10

An LR Parsing Example

- 0. $S' \rightarrow S\$$
- 1. $S \rightarrow aSbS$
- 2. $S \rightarrow a$

Grammar

	action			goto
	a	b	\$	S
0	S2			1
1			acc	
2	S2	R2	R2	3
3		S4		
4	S2			5
5		R1	R1	

Parse Table

Stack	Input	Action
0	aaababa\$	S2
0a2	aababa\$	S2
0a2a2	ababa\$	S2
0a2a2a2	baba\$	R2
0a2a2S3	baba\$	S4
0a2a2S3b4	aba\$	S2
0a2a2S3b4a2	ba\$	R2
0a2a2S3b4S5	ba\$	R1
0a2S3	ba\$	S4
0a2S3b4	a\$	S2
0a2S3b4a2	\$	R2
0a2S3b4S5	\$	R1
0S1	\$	acc

Another Example (G_1)

- 1. $E \rightarrow E + T$
- 2. $E \rightarrow T$
- 3. $T \rightarrow T * F$
- 4. $T \rightarrow F$
- 5. $F \rightarrow (E)$
- 6. $F \rightarrow id$

Grammar

state	action						goto		
	id	+	*	()	\$	E	T	F
0	S5			S4			1	2	3
1		S6				acc			
2		R2	S7		R2	R2			
3		R4	R4		R4	R4			
4	S5			S4			8	2	3
5		R6	R6		R6	R6			
6	S5			S4				9	3
7	S5			S4					10
8		S6			S11				
9		R1	S7		R1	R1			
10		R3	R3		R3	R3			
11		R5	R5		R5	R5			

Parse Table

Another Example

Stack	Input	Action
(1) 0	id * id + id \$	shift
(2) 0 id 5	* id + id \$	reduce by $F \rightarrow id$
(3) 0 F 3	* id + id \$	reduce by $T \rightarrow F$
(4) 0 T 2	* id + id \$	shift
(5) 0 T 2 * 7	id + id \$	shift
(6) 0 T 2 * 7 id 5	+ id \$	reduce by $F \rightarrow id$
(7) 0 T 2 * 7 F 10	+ id \$	reduce by $T \rightarrow T * F$
(8) 0 T 2	+ id \$	reduce by $E \rightarrow T$
(9) 0 E 1	+ id \$	shift
(10) 0 E 1 + 6	id \$	shift
(11) 0 E 1 + 6 id 5	\$	reduce by $F \rightarrow id$
(12) 0 E 1 + 6 F 3	\$	reduce by $T \rightarrow F$
(13) 0 E 1 + 6 T 9	\$	reduce by $E \rightarrow E + T$
(14) 0 E 1	\$	accept

CS 5300 - SJAllan

13

Yet Another Example (G_2)

- 0. $E' \rightarrow E\$$ 5. $F \rightarrow F^*$
- 1. $E \rightarrow E+T$ 6. $F \rightarrow (E)$
- 2. $E \rightarrow T$ 7. $F \rightarrow a$
- 3. $T \rightarrow TF$ 8. $F \rightarrow b$
- 4. $T \rightarrow F$ 9. $F \rightarrow e$

Grammar

	action									goto		
	a	b	e	()	+	*	\$	E	T	F	
0	S5	S6	S7	S4					1	2	3	
1						S8		acc				
2	S5	S6	S7	S4	R2	R2		R2			9	
3	R4	R4	R4	R4	R4	R4	S10	R4				
4	S5	S6	S7	S4					11	2	3	
5	R7	R7	R7	R7	R7	R7	R7	R7				
6	R8	R8	R8	R8	R8	R8	R8	R8				
7	R9	R9	R9	R9	R9	R9	R9	R9				
8	S5	S6	S7	S4						12	3	
9	R3	R3	R3	R3	R3	R3	S10	R3				
10	R5	R5	R5	R5	R5	R5	R5	R5				
11					S13	S8						
12	S5	S6	S7	S4	R1	R1		R1			9	
13	R6	R6	R6	R6	R6	R6	R6	R6				

CS 5300 - SJAllan

Parse Table

14

Yet Another Example

Stack	Input	Action	Stack	Input	Action
0	(ab+a)*b\$	S4	0 4 11 8 12)*b\$	R1
0 4	ab+a)*b\$	S5	0 4 11)*b\$	S13
0 4 5	b+a)*b\$	R7	0 4 11 13	*b\$	R6
0 4 3	b+a)*b\$	R4	0 3	*b\$	S10
0 4 2	b+a)*b\$	S6	0 3 10	b\$	R5
0 4 2 6	+a)*b\$	R8	0 3	b\$	R4
0 4 2 9	+a)*b\$	R3	0 2	b\$	S6
0 4 2	+a)*b\$	R2	0 2 6	\$	R8
0 4 11	+a)*b\$	S8	0 2 9	\$	R3
0 4 11 8	a)*b\$	S5	0 2	\$	R2
0 4 11 8 5)*b\$	R7	0 1	\$	acc
0 4 11 8 3)*b\$	R4			

CS 5300 - SJAllan

15

LR(k) Parsing

- The only difference between parsing one LR language and another is the information in the *action* and *goto* fields of the parse table
- The parsing algorithm is always the same

CS 5300 - SJAllan

16

Constructing Parse Tables

- LR(0) items
 - A production with a dot in the rhs
 - Consider the production: $A \rightarrow BaC$
 - Can have the following items:
 - $[A \rightarrow \cdot BaC]$
 - $[A \rightarrow B \cdot aC]$
 - $[A \rightarrow Ba \cdot C]$
 - $[A \rightarrow BaC \cdot]$
 - $A \rightarrow \epsilon$ has one item $[A \rightarrow \cdot]$
 - How many items does $A \rightarrow \beta$ have?

CS 5300 - SJAllan

17

Items

- The item $A \rightarrow \beta_1 \cdot \beta_2$ tells us:
 - We are looking for a handle $\beta_1 \beta_2$ for the production $A \rightarrow \beta_1 \beta_2$ and we have seen β_1 thus far
- Item $S' \rightarrow \cdot S$ says we have seen nothing thus far and we expect a match with S before we can reduce by $S' \rightarrow S$.

CS 5300 - SJAllan

18

Item Set Construction

- Start operation:
 - If S is the start symbol, and $S \rightarrow \alpha$ is some production, the item $[S \rightarrow \cdot \alpha]$ is associated with the start state
- Completion operation (closure):
 - If $[A \rightarrow \alpha \cdot B \gamma]$, where $B \in V_n$, is an item in some state I , then every item of the form $[B \rightarrow \cdot \beta]$ must be included in state I . This rule is repeated until no more new items can be added to state I
- Read operation:
 - Let $[A \rightarrow \alpha \cdot X \gamma]$, where $X \in (V_n \cup V_t)$, be an item associated with some state I . Then $[A \rightarrow \alpha \cdot X \cdot \gamma]$ is associated with state J (possibly the same as I), and a transition from I to J on symbol X exists

Construction of Finite State System

1. Give the start state a number and use the *start operation* to put one item into it
 - Use the *completion operation* to get more items into this state
 - Eventually the completion operation has to end
2. Use the *read operation* to start one or more new states, based on the present state
 - It is possible for this new state to be equivalent to some previous state
 - If so, these states are merged
3. Complete the new state started previously by applying the *completion operation*
4. Repeat steps 2 and 3 until no new states are added

Example of LR(0) Items

0. $S' \rightarrow S$
 1. $S \rightarrow aSbS$
 2. $S \rightarrow a$

Grammar

0. $[S' \rightarrow S]$ 3. $[S \rightarrow aS \cdot bS]$
 $[S \rightarrow aSbS]$ 4. $[S \rightarrow aSb \cdot S]$
 $[S \rightarrow a]$ 5. $[S \rightarrow aSbS \cdot]$
 1. $[S' \rightarrow S \cdot]$ $[S \rightarrow a]$
 2. $[S \rightarrow a \cdot SbS]$ $[S \rightarrow a]$
 $[S \rightarrow a \cdot]$
 $[S \rightarrow aSbS]$
 $[S \rightarrow a]$

LR(0) Items

	a	b	S
0	2		1
1			
2	2		3
3		4	
4	2		5
5			

Transition Table

Inadequate or Inconsistent States

- **Definition:**
 - Any state containing both a completed item $[A \rightarrow \alpha \cdot]$ and any other item is said to be *inadequate* or *inconsistent*
 - Such a state represents a conflict in a parsing decision
- If the item sets contain no inadequate states, the grammar is said to be LR(0)

Resolution of Inadequate States

- Look ahead may be determined by computing the *Follow* sets for the nonterminal on the lhs of the production in the inadequate state
 - This leads to an SLR(1) parser
- Discard the LR(0) item construction and use a larger k (i.e., 1, 2, ...)
 - This is only practical for small k or for a small grammar
- A full LR(1) item construction can be compressed in the interest of reducing the number of states and the state-set size by merging certain states
 - This leads to an LALR(1) parser

FIRST

- $FIRST(\alpha)$ is the set of all terminals that begin any string derived from α
- Computing FIRST:
 - If X is a terminal, $FIRST(X) = \{X\}$
 - If $X \rightarrow \varepsilon$ is a production, add ε to $FIRST(X)$
 - If X is a nonterminal and $X \rightarrow Y_1 Y_2 \dots Y_n$ is a production:
 - For all terminals a , add a to $FIRST(X)$ if a is a member of any $FIRST(Y_i)$ and ε is a member of $FIRST(Y_1), FIRST(Y_2), \dots, FIRST(Y_{i-1})$
 - If ε is a member of $FIRST(Y_1), FIRST(Y_2), \dots, FIRST(Y_n)$, add ε to $FIRST(X)$

FOLLOW

- $FOLLOW(A)$, for $A \in V_n$, is the set of terminals a that can appear immediately to the right of A in some sentential form
- More formally, a is in $FOLLOW(A)$ if and only if there exists a derivation of the form $S \Rightarrow^* \alpha A a \beta$
- $\$$ is in $FOLLOW(A)$ if and only if there exists a derivation of the form $S \Rightarrow^* \alpha A$

Computing FOLLOW

- Place $\$$ in $FOLLOW(S)$
- If there is a production $A \rightarrow \alpha B \beta$, then everything in $FIRST(\beta)$ (except for ϵ) is in $FOLLOW(B)$
- If there is a production $A \rightarrow \alpha B$, or a production $A \rightarrow \alpha B \beta$ where $FIRST(\beta)$ contains ϵ , then everything in $FOLLOW(A)$ is also in $FOLLOW(B)$

FIRST and FOLLOW Example

$$\begin{aligned} E &\rightarrow TE' \\ E' &\rightarrow +TE' \mid \varepsilon \\ T &\rightarrow FT' \\ T' &\rightarrow *FT' \mid \varepsilon \\ F &\rightarrow (E) \mid \text{id} \end{aligned}$$
$$\begin{aligned} \text{FIRST}(E) &= \text{FIRST}(T) = \text{FIRST}(F) = \{(, \text{id}\} \\ \text{FIRST}(E') &= \{+, \varepsilon\} \\ \text{FIRST}(T') &= \{*, \varepsilon\} \\ \text{FOLLOW}(E) &= \text{FOLLOW}(E') = \{), \$\} \\ \text{FOLLOW}(T) &= \text{FOLLOW}(T') = \{+,), \$\} \\ \text{FOLLOW}(F) &= \{+, *, \$\} \end{aligned}$$

SLR(1) Parse Table Construction

1. Construct LR(0) item sets $\{I_0, \dots, I_n\}$
2. State i is constructed from I_i :
 - a. If $[A \rightarrow \alpha \cdot a\beta] \in I_i$, $\text{goto}(I_i, a) = I_j$, and $a \in V_t$, then set $\text{action}[i, a] = \text{shift } j$
 - b. If $[A \rightarrow \alpha \cdot] \in I_i$ then $\text{action}[i, a] = \text{reduce } A \rightarrow \alpha \ \forall \text{ Follow}(A)$
 - c. If $[S' \rightarrow S \cdot] \in I_i$, then $\text{action}[i, \$] = \text{accept}$
3. The goto transitions for state i are constructed for all nonterminals A using the rule: If $\text{goto}(I_i, A) = I_j$, then $\text{goto}[i, A] = j$
4. All entries not defined by rules 2 and 3 are made *error*
5. The initial state of the parser is the one constructed from the set of items containing $[S' \rightarrow S \cdot]$

Item Sets Using G_1

I_0 : $[E' \rightarrow \cdot E]$ $[E \rightarrow \cdot E+T]$ $[E \rightarrow \cdot T]$ $[T \rightarrow \cdot T * F]$ $[T \rightarrow \cdot F]$ $[F \rightarrow \cdot (E)]$ $[F \rightarrow \cdot a]$	I_1 : $[E' \rightarrow E \cdot]$ $[E \rightarrow E \cdot +T]$	I_2 : $[E \rightarrow T \cdot]$ $[T \rightarrow T \cdot * F]$
I_3 : $[T \rightarrow F \cdot]$	I_4 : $[F \rightarrow (\cdot E)]$ $[E \rightarrow E \cdot +T]$ $[E \rightarrow \cdot T]$ $[T \rightarrow \cdot T * F]$ $[T \rightarrow \cdot F]$ $[F \rightarrow (\cdot E)]$ $[F \rightarrow \cdot a]$	I_5 : $[F \rightarrow a \cdot]$
I_6 : $[E \rightarrow E + \cdot T]$ $[T \rightarrow \cdot T * F]$ $[T \rightarrow \cdot F]$ $[F \rightarrow (\cdot E)]$ $[F \rightarrow \cdot a]$	I_7 : $[T \rightarrow T * \cdot F]$ $[F \rightarrow (\cdot E)]$ $[F \rightarrow \cdot a]$	I_8 : $[F \rightarrow (E \cdot)]$ $[E \rightarrow E \cdot +T]$
I_9 : $[E \rightarrow E + T \cdot]$ $[T \rightarrow T \cdot * F]$	I_{10} : $[T \rightarrow T * F \cdot]$	I_{11} : $[F \rightarrow (E) \cdot]$

CS 5300 - SJAllan

29

Example G_1

	a	+	*	()	E	T	A
I_0	I_5			I_4		I_1	I_2	I_3
I_1		I_6						
I_2			I_7					
I_3								
I_4	I_5			I_4		I_8	I_2	I_3
I_5								
I_6	I_5			I_4			I_9	I_3
I_7	I_5			I_4				I_{10}
I_8		I_6				I_{11}		
I_9			I_7					
I_{10}								
I_{11}								

Transition Table

Follow(E) = {+,), \$}
 Follow(T) = {+,), \$, *}
 Follow(F) = {+,), \$, *}

Follow Sets

The SLR(1) table is seen in slide 12

CS 5300 - SJAllan

30

Item Sets Using G_2

Follow(E) = {+,), \$}
 Follow(T) = {+,), \$, (, a, b, e}
 Follow(F) = {+,), \$, *, (, a, b, e}

The SLR(1) table is shown on slide 14

I_0 : [E'→E] [E→E+T] [E→T] [T→TF] [T→F] [F→F*] [F→(E)] [F→a] [F→b] [F→e]	I_4 : [F→(E)] [E→E+T] [E→T] [T→TF] [T→F] [F→F*] [F→(E)] [F→a] [F→b] [F→e]	I_9 : [T→TF] [F→F*] I_{10} : [F→F*] I_{11} : [F→(E)] [E→E+T] I_{12} : [E→E+T] [T→TF] [F→F*] [F→(E)] [F→a] [F→b] [F→e]
I_1 : [E'→E.] [E→E+T]	I_5 : [F→a.] I_6 : [F→b.]	I_{13} : [F→(E.)]
I_2 : [E→T.] [T→TF] [F→F*] [F→(E)] [F→a] [F→b] [F→e]	I_7 : [F→e.] I_8 : [E→E+T.] [T→TF] [T→F] [F→F*] [F→(E)] [F→a] [F→b]	
I_3 : [T→F.] [F→F*.]	[F→b.] [F→e.]	

Grammar G_3

0. $S' \rightarrow S$
1. $S \rightarrow Aa$
2. $S \rightarrow dAb$
3. $S \rightarrow dca$
4. $S \rightarrow cb$
5. $A \rightarrow c$

Grammar G_3

	action					goto	
	a	b	c	d	\$	S	A
0			S4	S3		1	2
1					acc		
2	S5						
3			S7				6
4	R5	S8/R5					
5					R1		
6		S9					
7	R5/S10	R5					
8					R4		
9					R2		
10					R3		

SLR(1) Parse Table

Looking at G_3

- Consider the string cb
- We run into problems in state 4
 - Notice that we can never have a b following an A (under the assumption we reduce)
 - Ab is not a viable prefix
- A *viable prefix* is so called because it is always possible to add terminal symbols to the end of a viable prefix to obtain a right sentential form

Construction of LR(1) Parse Table

- Using LR(0) item, inadequate are resolved as follows:
 - On the item $[A \rightarrow \alpha \cdot]$, the reduce operation is used on the $\text{Follow}(A)$
 - As we have seen, this doesn't always work
- We want to carry more information in the item
 - We will add look ahead symbols
 - $[A \rightarrow \alpha \cdot \beta, a]$ where
 - $A \rightarrow \alpha \beta$
 - a is a terminal or $\$$

LR(1) Items

- How does look ahead work
 - No effect for an item of the form $[A \rightarrow \alpha \cdot \beta, a]$, where $\beta \neq \epsilon$
 - An item like $[A \rightarrow \alpha \cdot, a]$ calls for the reduction by $A \rightarrow \alpha$ only if the next input symbol is a

Construction of LR(1) Items

- *Initial state construction*
 - If $S \rightarrow \alpha$ is a production, then add $[S \rightarrow \alpha, \$]$ to the initial state; S is the start symbol
 - If $[A \rightarrow B\alpha, \omega]$ is in the initial state, and $B \rightarrow \beta$ is a production, add all items of the form $[B \rightarrow \beta, z]$ to the initial state where $z \in \text{Follow}(\alpha\omega)$. Repeat this step until no more items are added
- *Starting a new state*
 - Let $[A \rightarrow \alpha \cdot X\gamma, \omega]$ be in state I , where $X \in (V_t \cup V_n)$. Locate all items in I such that X follows the “.”, then start a new state J with these items, but move the dot past X
 - Merge states if the items and the look ahead are the same.
- *Closing a new state*
 - For every item of the form $[A \rightarrow \alpha \cdot B\gamma, \omega]$ in state J , where $B \in V_n$, and for every production $B \rightarrow \beta$, we add to J all items of the form $[B \rightarrow \beta, u]$, where $u \in \text{First}(\gamma\omega)$. This step is repeated until no new items can be added to J .
- Steps 2 and 3 are repeated alternately, in that order, until no more new state can be constructed or closed

LR(1) Parsing Tables – Example 1

0. $S' \rightarrow S$
 1. $S \rightarrow Aa$
 2. $S \rightarrow dAb$
 3. $S \rightarrow dca$
 4. $S \rightarrow cb$
 5. $A \rightarrow c$

0. $[S' \rightarrow S, \$]$ 1. $[S' \rightarrow S, \$]$ 2. $[S \rightarrow A \cdot a, \$]$
 $[S \rightarrow Aa, \$]$
 $[S \rightarrow dAb, \$]$
 $[S \rightarrow dca, \$]$
 $[S \rightarrow cb, \$]$
 $[A \rightarrow c, a]$
 3. $[S \rightarrow d \cdot Ab, \$]$ 4. $[S \rightarrow c \cdot b, \$]$ 5. $[S \rightarrow Aa, \$]$
 $[S \rightarrow d \cdot ca, \$]$ $[A \rightarrow c, a]$
 $[A \rightarrow c, b]$
 6. $[S \rightarrow dA \cdot b, \$]$ 7. $[S \rightarrow dc \cdot a, \$]$ 8. $[S \rightarrow cb, \$]$
 $[A \rightarrow c, b]$
 9. $[S \rightarrow dAb, \$]$ 10. $[S \rightarrow dca, \$]$

LR(1) Parsing Tables – Example 1

	<i>action</i>					<i>goto</i>	
	a	b	c	d	\$	S	A
0			S4	S3		1	2
1					acc		
2	S5						
3			S7				6
4	R5	S8					
5					R1		
6		S9					
7	S10	R5					
8					R4		
9					R2		
10					R3		

LR(1) Parsing Tables – Example 2

0. $S' \rightarrow S$
 1. $S \rightarrow CC$
 2. $C \rightarrow eC$
 3. $C \rightarrow d$

0. $[S' \rightarrow S, \$]$ 1. $[S' \rightarrow S, \$]$ 2. $[S \rightarrow C \cdot C, \$]$
 $[S \rightarrow CC, \$]$ $[C \rightarrow eC, \$]$
 $[C \rightarrow eC, e/d]$ $[C \rightarrow d, \$]$
 $[C \rightarrow d, e/d]$
 3. $[C \rightarrow e \cdot C, e/d]$ 4. $[C \rightarrow d \cdot, e/d]$ 5. $[S \rightarrow CC \cdot, \$]$
 $[C \rightarrow eC, e/d]$ $[C \rightarrow d, e/d]$
 $[C \rightarrow d, e/d]$
 6. $[C \rightarrow e \cdot C, \$]$ 7. $[C \rightarrow d \cdot, \$]$ 8. $[C \rightarrow eC \cdot, e/d]$
 $[C \rightarrow eC, \$]$ $[C \rightarrow d, \$]$
 $[C \rightarrow d, \$]$
 9. $[C \rightarrow eC \cdot, \$]$

LR(1) Parsing Tables – Example 2

	<i>action</i>			<i>goto</i>	
	<i>e</i>	<i>d</i>	<i>\$</i>	<i>S</i>	<i>C</i>
0	S3	S4		1	2
1			acc		
2	S6	S7			5
3	S3	S4			8
4	R3	S3			
5			R1		
6	S6	S7			9
7			R3		
8	R2	R2			
9			R2		

LALR(1) Parsing Tables

- LR(1) parse tables are too large for practical use
- In the previous example (2), consider states 4 and 7
 - They are the same but the look ahead is different
- For LALR(1) parsing table, we combine states that are like this

LALR(1) Parse Table Construction

1. Construct I_0, \dots, I_n , the LR(1) items
2. For each core present among the sets, find all sets having that core, and replace these sets by their union
3. Let J_0, \dots, J_m be the resulting sets.
 - a. The parsing actions are created as before
4. The Goto table is constructed as follows:
 - a. If J is the union of one or more LR(1) items, then the cores of $\text{Goto}(I_1, X), \dots, \text{Goto}(I_k, X)$ are the same.
 - b. $\text{Goto}(J, X) = \cup_{i=1}^k \text{Goto}(I_i, X)$

LALR(1) Parse Table – Example 2

36. $[C \rightarrow e.C, e/d/\$]$
 $[C \rightarrow ec, e/d/\$]$
 $[C \rightarrow d, e/d/\$]$
 47. $[C \rightarrow d., e/d/\$]$
 89. $[C \rightarrow eC., e/d/\$]$

Merged States

Stack	Input	Action
0	eed\$	S36
0 36	ed\$	S36
0 36 36	d\$	S47
0 36 36 47	\$	R3
0 36 36 89	\$	R2
0 36 89	\$	R2
0 2	\$	error

LALR Parse of eed\$

	action			goto	
	e	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		

LALR(1) Parse Table

Ambiguous Grammars

- Every ambiguous grammar, such as $E \rightarrow E-E \mid a$ fails to be LR
- Consider the grammar:
 - Parse a-a-a
- Why would you want an ambiguous grammar – $E \rightarrow E+E \mid E^*E \mid (E) \mid a$?
 - Gets rid of a lot of reductions of single nonterminal symbols
 - Tools such as Bison allow ambiguous grammars but let you specify precedence and associativity

Conflicts in Shift-Reduce Parsing

- There are grammars for which shift-reduce parsing cannot be used
- Shift/reduce conflict:
 - Cannot decide whether to shift or reduce
- Reduce/reduce conflict:
 - Cannot decide which of multiple possible reductions to make
- Sometimes can add rule to adapt for use with ambiguous grammar