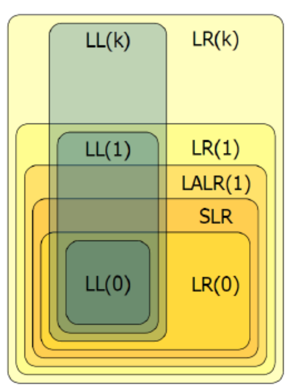
Compilers (CS30003)

Lecture 12-13

Pralay Mitra

Autumn 2023-24

LR Parsing: CFG Classes



- **LL(k), Top-Down, Predictive**: LL parser (Left-to-right, Leftmost derivation) with k look-ahead
- **LR(k), Bottom-Up, Shift-Reduce**: LR parser (Left-to-right, Rightmost derivation) with k look-ahead

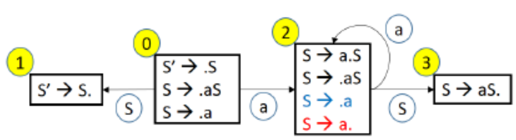
Autumn 2022-23 Pralay Mitra

LR Parsers
SLR(1) Parser

LR(0) Parser: Shift-Reduce Conflict

$G_5 = \{S \rightarrow aS|a\}$

SLR(1) Parser:
S' → S
S → aS | a
L(G) = {a+}

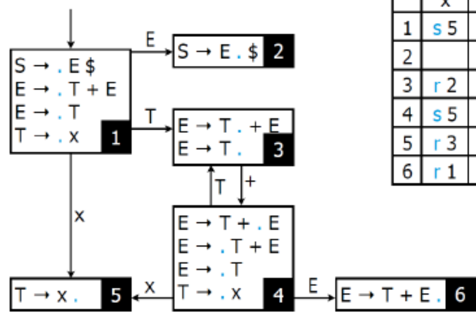


State	a	$	S
0	s2		1
1		Acc	
2	s2/r2	r2	3
3	r1	r1	

S → a$
S → aS$ → aa$
S → aS$ → aaS$ → aaa$

- Consider State 2.
  - By $S \rightarrow .a$, we should shift on a and remain in state 2
  - By $S \rightarrow a.$, we should reduce by production 2
- We have a Shift-Reduce Conflict
- As $FOLLOW(S) = \{\$\}$, we decide in favor of shift. Why?

LR(0) Parser: Shift-Reduce Conflict



	x	+	$	E	T
1	s 5			g 2	g 3
2			a		
3	r 2	?	r 2		
4	s 5			g 6	g 3
5	r 3	r 3	r 3		
6	r 1	r 1	r 1		

E → T + E
E → T
T → x

Consider State 3.

- By $E \rightarrow T. + E$, we should shift on $+$ and move to state 4
- By $E \rightarrow T.$, we should reduce by production 2

We have a Shift-Reduce Conflict

To resolve, we build SLR(1) Parser

SLR(1) Parser Construction

LR(0) Item: Canonical collection of LR(0) Items used in SLR(1) as well

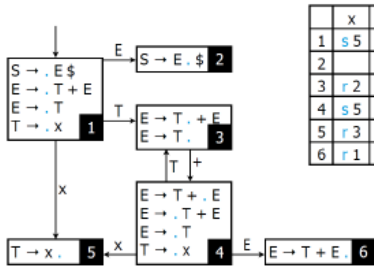
Closure: Same way as LR(0)

State: Collection of LR(0) items and their closures.

Actions: Shift (s#), Reduce (r#), Accept (acc), Reject (<space>), GOTO (#):

- Shift on input symbol to state#
- **Reduction by production# only on the input symbols that belong to the FOLLOW of the left-hand side**
- Accept on reduction by the augmented production
- GOTO on transition of non-terminal after reduction

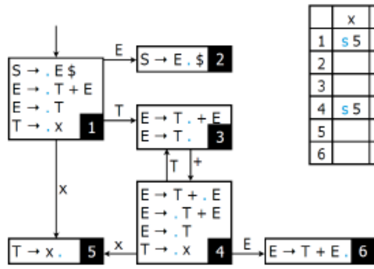
SLR Parse Table: Shift-Reduce Conflict on LR(0)



	x	+	$	E	T
1	s 5			g 2	g 3
2			a		
3	r 2	?	r 2		
4	s 5			g 6	g 3
5	r 3	r 3	r 3		
6	r 1	r 1	r 1		

E → T + E
E → T
T → x

Reduce a production $S \rightarrow \ldots$ on symbols $k \in T$ if $k \in Follow(S)$



	x	+	$	E	T
1	s 5			g 2	g 3
2			a		
3		s 4	r 2		
4	s 5			g 6	g 3
5		r 3	r 3		
6			r 1		

E → T + E
E → T
T → x

LR Parsers

LR(1) Parser

SLR(1) Parse: Shift-Reduce Conflict

Grammar G_9

1:	S	$\rightarrow$	$L = R$
2:	S	$\rightarrow$	R
3:	L	$\rightarrow$	$*R$
4:	L	$\rightarrow$	id
5:	R	$\rightarrow$	L

I_0: $S' \rightarrow \cdot S$, $S \rightarrow \cdot L = R$, $S \rightarrow \cdot R$, $L \rightarrow \cdot * R$, $L \rightarrow \cdot \mathbf{id}$, $R \rightarrow \cdot L$

I_1: $S' \rightarrow S \cdot$

I_2: $S \rightarrow L \cdot = R$, $R \rightarrow L \cdot$

I_3: $S \rightarrow R \cdot$

I_4: $L \rightarrow * \cdot R$, $R \rightarrow \cdot L$, $L \rightarrow \cdot * R$, $L \rightarrow \cdot \mathbf{id}$

I_5: $L \rightarrow \mathbf{id} \cdot$

I_6: $S \rightarrow L = \cdot R$, $R \rightarrow \cdot L$, $L \rightarrow \cdot * R$, $L \rightarrow \cdot \mathbf{id}$

I_7: $L \rightarrow * R \cdot$

I_8: $R \rightarrow L \cdot$

I_9: $S \rightarrow L = R \cdot$

- $= \in FOLLOW(R)$ as $S \Rightarrow L = R \Rightarrow *R = R$
- So in State#2 we have a shift/reduce Conflict on $=$
- The grammar is not ambiguous. Yet we have the shift/reduce conflict as SLR is not powerful enough to remember enough left context to decide what action the parser should take on input $=$, having seen a string reducible to L.
- To resolve, we build LR(1) Parser

Autumn 2023-24 Pralay Mitra

LR(1) Parse Construction

Sample Grammar G_7

1:	S	$\rightarrow$	CC
2:	C	$\rightarrow$	cC
3:	C	$\rightarrow$	d

Augmented Grammar G_7

0:	S'	$\rightarrow$	S
1:	S	$\rightarrow$	CC
2:	C	$\rightarrow$	cC
3:	C	$\rightarrow$	d

- **LR(1) Item**: An LR(1) item has the form $[A \rightarrow \alpha.\beta, a]$ where $A \rightarrow \alpha\beta$ is a production and a is the look-ahead symbol which is a terminal or \$. As the dot moves through the right-hand side of the production, token a remains attached to it. LR(1) item $[A \rightarrow \alpha., a]$ calls for a reduce action when the look-ahead is a. *Examples*: $[S \rightarrow .CC, \$]$, $[S \rightarrow C.C, \$]$, $[S \rightarrow CC., \$]$
- **Closure(S)**:

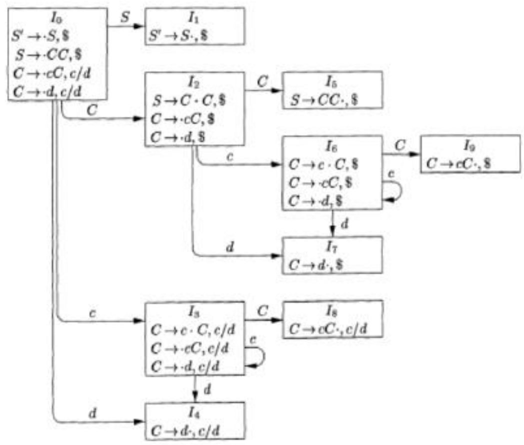
  For each item $[A \rightarrow \alpha.B\beta, t] \in S$,
  For each production $B \rightarrow \gamma \in G$,
  For each token $b \in FIRST(\beta t)$,
  Add $[B \rightarrow .\gamma, b]$ *to* S

  Closure is computed transitively. *Examples*:
  - Closure($[S \rightarrow C.C, \$]$) = $\{[S \rightarrow C.C, \$], [C \rightarrow .cC, \$], [C \rightarrow .d, \$]\}$
  - Closure($[C \rightarrow c.C, c/d]$) = $\{[C \rightarrow c.C, c/d], [C \rightarrow .cC, c/d], [C \rightarrow .d, c/d]\}$
- **State**: Collection of LR(1) items and their closures. *Examples*:
  - $\{[S \rightarrow C.C, \$], [C \rightarrow .cC, \$], [C \rightarrow .d, \$]\}$
  - $\{[C \rightarrow c.C, c/d], [C \rightarrow .cC, c/d], [C \rightarrow .d, c/d]\}$

LR(1) Parser: Example

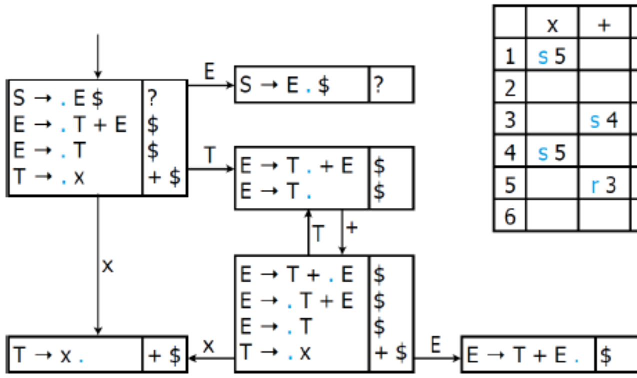
Construct an LR(1) parser for G_7:

1: $S \rightarrow CC$
2: $C \rightarrow cC$
3: $C \rightarrow d$



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		

LR(1) Parser: Example



	x	+	$	E	T
1	s 5			g 2	g 3
2			a		
3		s 4	r 2		
4	s 5			g 6	g 3
5		r 3	r 3		
6			r 1		

E → T + E
E → T
T → x

Source: https://www.slideshare.net/eelcovisser/lr-parsing-71059803?from_action=save

LR Parsers
LALR(1) Parser

LALR(1) Parser Construction

Sample Grammar G_7				Augmented Grammar G_7			
				0:	S'	$\rightarrow$	S
1:	S	$\rightarrow$	CC	1:	S	$\rightarrow$	CC
2:	C	$\rightarrow$	cC	2:	C	$\rightarrow$	cC
3:	C	$\rightarrow$	d	3:	C	$\rightarrow$	d

- **LR(1) States**: Construct the Canonical LR(1) parse table.
- **LALR(1) States**: Two or more LR(1) states having the same set of core LR(0) items may be merged into one by combining the look-ahead symbols for every item. Transitions to and from these merged states may also be merged accordingly. All other states and transitions are retained. *Examples*:
  - Merge
    State#3 = $\{[C \rightarrow c.C, c/d], [C \rightarrow .cC, c/d], [C \rightarrow .d, c/d]\}$ with
    State#6 = $\{[C \rightarrow c.C, \$], [C \rightarrow .cC, \$], [C \rightarrow .d, \$]\}$ to get
    State#36 = $\{[C \rightarrow c.C, c/d/\$], [C \rightarrow .cC, c/d/\$], [C \rightarrow .d, c/d/\$]\}$
  - Merge

    State#4 = $\{[C \rightarrow d., c/d]\}$ with

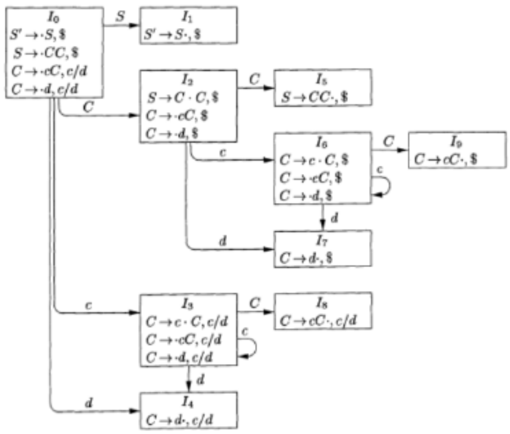
    State#7 = $\{[C \rightarrow d., \$]\}$ to get

    State#47 = $\{[C \rightarrow d., c/d/\$]\}$
- **Reduce/Reduce Conflict**: LR(1) to LALR(1) transformation cannot introduce any new shift/reduce conflict. But it may introduce reduce/reduce conflict.

LALR(1) Parser Example

Construct an LALR(1) parser for G_7:

$$\begin{array}{llll} 1: & S & \rightarrow & CC \\ 2: & C & \rightarrow & cC \\ 3: & C & \rightarrow & d \end{array}$$



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		

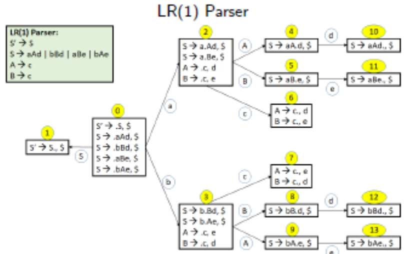
LALR(1) Parser: Reduce-Reduce Conflict

Consider $G_{10} = < \{a, b, c, d, e\}, \{S, A, B\}, S, P >$ where $P =$

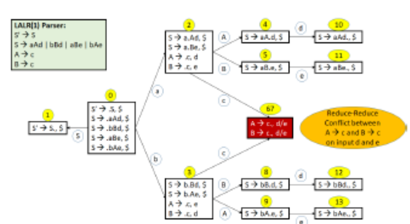
$$\begin{array}{llll} 0: & S' & \rightarrow & S \\ 1: & S & \rightarrow & aAd \\ 2: & S & \rightarrow & bBd \\ 3: & S & \rightarrow & aBe \\ 4: & S & \rightarrow & bAe \\ 5: & A & \rightarrow & c \\ 6: & B & \rightarrow & c \end{array}$$

Clearly, $L(G) = \{acd, bcd, ace, bce\}$

LR(1) Parser



LALR(1) Parser

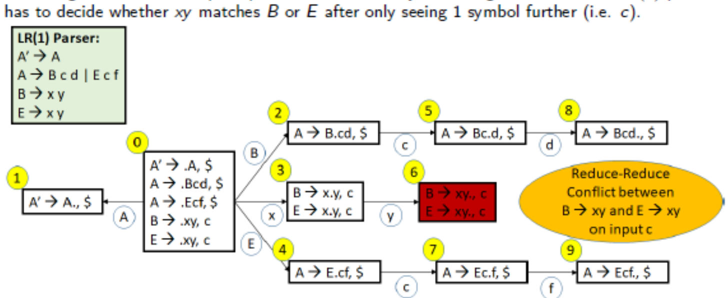


LR(1) Parser: Shift-Reduce Conflict

Grammar G_{11}

1:	A	$\rightarrow$	$B\ c\ d$
2:	A	$\rightarrow$	$E\ c\ f$
3:	B	$\rightarrow$	$x\ y$
4:	E	$\rightarrow$	$x\ y$

- For this grammar, an example input that starts with xyc is enough to confuse an LR(1) parser, as it has to decide whether xy matches B or E after only seeing 1 symbol further (i.e. c).



- An LL(1) parser would also be confused, but at the x - should it expand A to $B\ c\ d$ or to $E\ c\ f$, as both can start with x. An LL(2) or LL(3) parser would have similar problems at the y or c respectively.
- An LR(2) parser would be able to also see the d or f that followed the c and so make the correct choice between B and E.
- An LL(4) parser would also be able to look far enough ahead to see the d or f that followed the c and so make the correct choice between expanding A to $B\ c\ d$ or to $E\ c\ f$.

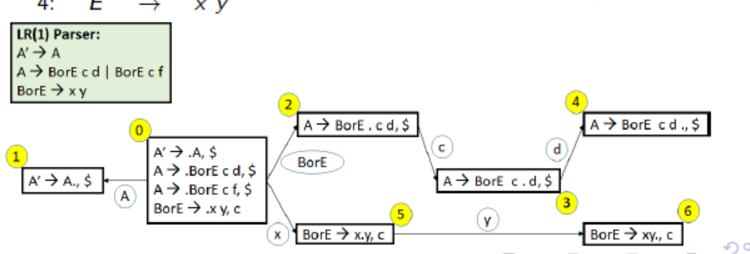
LR(k) Parser: Shift-Reduce Conflict

Grammar G_{12}

1:	A	$\rightarrow$	$B\ C\ d$
2:	A	$\rightarrow$	$E\ C\ f$
3:	B	$\rightarrow$	$x\ y$
4:	E	$\rightarrow$	$x\ y$
5:	C	$\rightarrow$	$C\ c$
6:	C	$\rightarrow$	c

- The grammar would confuse any LR(k) or LL(k) parser with a fixed amount of look-ahead
- To workaround, rewrite

1:	A	$\rightarrow$	$B\ C\ d$	as	1:	A	$\rightarrow$	$BorE\ c\ d$
2:	A	$\rightarrow$	$E\ C\ f$		2:	A	$\rightarrow$	$BorE\ c\ f$
3:	B	$\rightarrow$	$x\ y$		3:	$BorE$	$\rightarrow$	$x\ y$
4:	E	$\rightarrow$	$x\ y$					



Practice Example

Construct an LR(0) parser for G_7:

$$
\begin{array}{llll}
1: & S & \rightarrow & A\ A \\
2: & A & \rightarrow & a\ A \\
3: & A & \rightarrow & b
\end{array}
$$

Practice Example

Determine the LR Class (LR(0), SLR(1), LR(1) or LALR(1)) for the following grammars:

- G: $S \rightarrow aSb \mid b$
- G: $S \rightarrow Sa \mid b$
- G: $S \rightarrow (S) \mid SS \mid \epsilon$
- G: $S \rightarrow (S) \mid SS \mid ()$
- G: $S \rightarrow ddX \mid aX \mid \epsilon$
- G: $S \rightarrow E$; $E \rightarrow T + E \mid T$; $T \rightarrow int * T \mid int \mid (E)$
- G: $S \rightarrow V = E \mid E$; $E \rightarrow V$; $V \rightarrow x \mid *E$
- G: $S \rightarrow AB$; $A \rightarrow aAb \mid a$; $B \rightarrow d$

Practice Example

Construct an SLR(1) parser for G_8:

$$
\begin{array}{llll}
1: & S & \rightarrow & E \\
2: & E & \rightarrow & E + T \\
3: & E & \rightarrow & T \\
4: & T & \rightarrow & T * F \\
5: & T & \rightarrow & F \\
6: & F & \rightarrow & \textbf{id}
\end{array}
$$