LALR Parsing

LALR Parsing

- LR(1) parsing tables have many states
- LALR(1) parsing (Look-Ahead LR) combines LR(1) states to reduce table size
- Less powerful than LR(1)
  - Does not have shift-reduce conflicts, because shifts do not use lookaheads
  - May introduce reduce-reduce conflicts, but not much of a problem for grammars of programming languages

Example

- S → CC
- C → cC
- C → d

- Augmented
- S’ → S
- S → CC
- C → cC
- C → d

LR(1) items

- I_0 :

S' → .S, $

S → .CC, $

C → .cC, c/d (first(C$)

C → .d, c/d

- I_1 : goto(I_0, S)

  S' → S., $

- I_2 : goto(I_0, C)

  S → C.C, $

  C → .cC, $

  C → .d, $

- I_3 : goto(I_0 , c), goto(I_3 , c),
  C → c.C, c/d
  C → .cC, c/d
  C → .d, c/d
- I_4 : goto(I_0 , d) goto(I_3, d)
  C → d., c/d
- I_5 : goto(I_2 , C)
  S → CC., $
- I_6 : goto(I_2 , c) goto(I_6 ,c)
  C → c.C, $
  C → .cC, $
  C → .d, $

- I_7 : goto(I_2 , d) goto(I_6 , d)
  C → d., $
- I_8 : goto(I_3 , C)
  C → cC., c/d
- I_9 : goto(I_6 , C)
  C → cC., $

Parsing Table

- Construct C = {I_0 ,I_1 ,I_2 … I_n } the collection of LR(1) items for G’
- State I of the parser is from I_i

  if [A → α.aβ, b] is in I_i and goto(I_i, a) = I_j set action [i, a] = shift j, where a is a terminal

if [A → α . , a] is in I_i and A ≠ S', then set action[i, a] = reduce by A → α

// a conflict here implies the grammar is not CALR gramar

- If goto(I_i , A) = I_j then goto (i, A) = j
- [S' → .S, $] implies an accept action
- All other entries are error

LALR Parsing table

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

Merging

- I_3 : goto(I_0 , c), goto(I_3 , c),
  C → c.C, c/d
  C → .cC, c/d
  C → .d, c/d
- I_6 : goto(I_2 , c) goto(I_6 ,c)
  C → c.C, $
  C → .cC, $
  C → .d, $

- I_{36} : goto(I_0 , c), goto(I_{36} , c),
  C → c.C, c/d/$
  C → .cC, c/d/$
  C → .d, c/d/$

Merging

- I_4 : goto(I_0 , d) goto(I_3, d)
  C → d., c/d
- I_7 : goto(I_2 , d) goto(I_6 , d)
  C → d., $
- I_{47} : goto(I_2 , d) goto(I_6 , d)
  C → d., c/d/$
- I_8 : goto(I_3 , C)
  C → cC., c/d
- I_9 : goto(I_6 , C)
  C → cC., $
- I_{89} : goto(I_3 , C)
  C → cC., c/d/$

Parsing table - LALR

State	Action			goto	
	c	d	$	S	C
0	s36	s47		1	2
1					
2	s36	s47			5
36	s36	s47			89
47					
5					
89					

Parsing table - LALR

State	Action			goto	
	c	d	$	S	C
0	s36	s47		1	2
1			Accept		
2	s36	s47			5
36	s36	s47			89
47	r3	r3	r3		
5			r1		
89	r2	r2	r2		

Parsing algorithm

- Set input to point to the first symbol of w$
- Repeat
  - Let s be the state on the top of the stack
  - Let a be the symbol pointed to by ip
  - If action [s, a] = shift s’ then
    - Push a then s’ on top of the stack
    - Move input to the next input symbol

Parsing algorithm

- Else if action [s, a] = reduce A → β then
  - Pop 2 * | β | symbols off the stack
- Let s’ be the state now on the top of the stack
  - Push A then goto [s’, A] on top of the stack
  - Output the production A → β
- Else if action[s, a] = accept then return;
- Else error()

Parsing with CALR parser

Stack	Input	Action
0	ccdd$	[0, c] – shift 36
0 c 36	c d d $	[36, c] – shift 36
0 c 36 c 36	d d $	[36, d] – shift 47
0 c 36 c 36 d 47	d $	[47, d] – reduce 3, pop 2 symbols from stack, push C, goto(36, C) = 89
0 c 36 c 36 C 89	d $	[89, d] – reduce 2, pop 4 symbols from the stack, push C, goto(36, C) = 89
0 c 36 C 89	d $	[89, d] – reduce 2, pop 4 symbols from the stack, push C, goto(0, C) = 2

0 C 2

Stack	Input	Action
0 C 2	d $	[2, d] – shift 47
0 C 2 d 47	$	[47, $] – reduce 3, pop 2 symbols from the stack, goto(2, C) = 5
0 C 2 C 5	$	[5, $] – reduce 1, pop 4 symbols off the stack, goto(0, S) = 1
0 S 1	$	[1, $] – accept – successful parsing

C → d

S → CC

Example

- $S' \rightarrow S$
- $S \rightarrow L = R$
- $S \rightarrow R$
- $L \rightarrow * R$
- **L → id**
- $R \rightarrow L$

Another Example

- I_0

  $[S' \rightarrow \bullet S,$ **$**] goto($I_0$,$S$)=$I_1$
  $[S \rightarrow \bullet L$**=**$R,$ **$**] goto($I_0$,$L$)=$I_2$
  $[S \rightarrow \bullet R,$ **$**] goto($I_0$,$R$)=$I_3$
  $[L \rightarrow \bullet$***$R,$ **=/$**] goto($I_0$,***)=$I_4$
  $[L \rightarrow \bullet$**id**, **=/$**] goto($I_0$,**id**)=$I_5$
  $[R \rightarrow \bullet L,$ **$**] goto($I_0$,$L$)=$I_2$

- I_1: goto(I_0,S)
  $[S' \rightarrow S\bullet,$ **$**]

- I_2 : goto(I_0,L)

  $[S \rightarrow L\bullet$**=**$R,$ **$**] goto($I_2$,**=**)=$I_6$
  $[R \rightarrow L\bullet,$ **$**]

- I_3 : goto(I_0,R)
  $[S \rightarrow R\bullet,$ **$**]

- I_4 : goto(I_0,*) goto(I_4,*)
  $[L \rightarrow$ ***$\bullet R,$ **=/$**] goto($I_4$,$R$)=$I_7$
  $[R \rightarrow \bullet L,$ **=/$**] goto($I_4$,$L$)=$I_8$
  $[L \rightarrow \bullet$***$R,$ **=/$**] goto($I_4$,*)=$I_4$
  $[L \rightarrow \bullet$**id**, **=/$**] goto($I_4$,**id**)=$I_5$

- I_5 : goto(I_0,id) goto(I_4,id)

  $[L \rightarrow$ **id**$\bullet,$ **=/$**]

- I_6 : goto(I_2,=)

  [$S \rightarrow L{=}\bullet R$, $] goto($I_6$,$R$)=$I_9$
  [$R \rightarrow \bullet L$, $] goto($I_6$,$L$)=$I_{10}$

  [$L \rightarrow \bullet{*}R$, $] goto($I_6$,*)=$I_{11}$
  [$L \rightarrow \bullet$id, $] goto($I_6$,id)=$I_{12}$

- I_7 : goto(I_4,R)

  [$L \rightarrow {*}R\bullet$, =/$]

- I_8 : goto(I_4,L)

  [$R \rightarrow L\bullet$, =/$]

- I_9 : goto(I_6,R)

  [$S \rightarrow L{=}R\bullet$,$]

- I_{10} : goto(I_6,L) goto(I_{11},L)

  [$R \rightarrow L\bullet$, $]

- I_{11} : goto(I_6,*) goto(I_{11},*)

  [$L \rightarrow {*}\bullet R$, **$**] goto(I_{11},R)=I_{13}
  [$R \rightarrow \bullet L$, **$**] goto($I_{11}$,$L$)=$I_{10}$
  [$L \rightarrow \bullet{*}R$, **$**] goto(I_{11},*)=I_{11}
  [$L \rightarrow \bullet$**id**, **$**] goto($I_{11}$,**id**)=$I_{12}$

- ***I*₁₂** *:* goto(I_6,id) goto(I_{11},id)
  [$L \rightarrow$ **id**•, **$**]
- ***I*₁₃** *:* goto(I_{11},*R*)
  [$L \rightarrow$ ***R**•, **$**]

- I_4 and I_{11}
- I_5 and I_{12}
- I_7 and I_{13}
- I_8 and I_{10}

State	Action				goto		
	id	*	=	$	S	L	R
0	s512	s411			1	2	3
1				accept			
2			s6	r5			
3				r2			
411	s512	S411				810	713
512			r4	r4			
6	s512	s411				810	9

State	Action				goto		
	id	*	=	$	S	L	R
713			r3	r3			
810			r5	r5			
9				r1			

LL, LR parsers

- 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 $\times$ non-terminals $\rightarrow$ goto state transitions

LL and LR parsers

- A grammar is
  - LL(1) if its LL(1) parse table has no conflicts
  - SLR if its SLR parse table has no conflicts
  - LALR(1) if its LALR(1) parse table has no conflicts
  - CALR(1) if its CALR(1) parse table has no conflicts

Resolving conflicts

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

Error Detection

- Canonical LR parser uses full LR(1) parse tables and will never make a single reduction before recognizing 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

- Panic mode
  - Pop until state with a goto on a nonterminal *A* is found, where *A* represents a major programming construct
  - 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 that allows parsing to continue

Summary

- LALR parsing table construction based on CALR parsing table
- LALR parsing action
- Error recovery in LR parsers

Thank you