CALR Parsing

Conflict in SLR parsers

- Shift / reduce conflict arises
- Follow information alone is not sufficient to decide when to reduce.
- Hence, powerful parser is required

Conflicts in SLR parsers

- In SLR, if there is a production of the form $A \rightarrow \alpha\blacksquare$, then a reduce action takes place based on follow(A)
- There would be situations, where when state i appears on the TOS, the viable prefix $\beta\alpha$ on the stack is such that βA cannot be followed by terminal ‘a’ in a right sentential form
- Hence, the reduction $A \rightarrow \alpha$ would be invalid on input ‘a’

CALR parsers motivation

- If it is possible to do more in the states that allow us to rule out some of the invalid reduction, introduce more states
- Introduce exactly which input symbols to follow a particular non-terminal

CALR parsers

- Construct LR(1) items
- Use these items to construct the CALR parsing table involving action and goto
- Use this table, along with input string and stack to parse the string

CALR motivation

- Extra symbol is incorporated in the items to include a terminal symbol as a second component
- A →[α .β, a] where A →αβ is a production and 'a' is a terminal or the right end marker $ - LR(1) item

$A \rightarrow \alpha . \beta , a$

terminal and $

$ a

LR(1) item

- 1 – refers to the length of the second component – lookahead of the item
- Lookahead has no effect in A →[α .β , a] where β is not ε, but A →[α . , a] calls for a reduction A → α if the next input symbol is 'a', 'a' will be subset of follow(A)

LR(1) item

- $A \rightarrow [\alpha .\beta , a]$ is a valid item for a viable prefix γ if there is a derivation $S \Rightarrow \delta A w \Rightarrow \delta\alpha\beta w$ where $\gamma = \delta\alpha$ and either ‘a’ is the first symbol of ‘w’ or ‘w’ is ε and ‘a’ is $

LR(1) item algorithm

- Closure (I)

  {repeat for each item [$A \rightarrow \alpha \cdot B\beta$, a] in I,

  for each production $B \rightarrow \gamma$ in G'

  and each terminal b in First(βa) such that [$B \rightarrow .\gamma$, b] is not in I do

  add [$B \rightarrow .\gamma$, b]

  until no more items can be added to I

  end }

$G \Rightarrow G'$

$S' \rightarrow S$

first (βa)

b

$A \rightarrow \alpha . B\beta , a$

$B \rightarrow .\gamma , b$

first(a)

$A \rightarrow \alpha . B , a$

$B \rightarrow .\gamma , a$

Goto(I, X)

Begin

Let J be the set of items [A$\rightarrow$αX.β , a] such that

[A$\rightarrow$α.Xβ , a] is in I;

Return closure(J)

end

Items(G')

Begin C:= closure ({S' → .S, $});

repeat for each set of items I in C and each grammar symbol X such that goto(I,X) is not empty and not in C

add goto(I, X) to C

until no more set of items can be added to C

end

Example

1 • S → CC
2 • C → cC
3 • C → d

FIRST(C) = {c, d}

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

LR(1) items

- I_0 :

$A \rightarrow \alpha . B\beta, a$

$S' \rightarrow .S, \$$ $Fi(\beta a) = Fi(\$)$

$A \rightarrow \alpha . B \beta \quad a$

$S \rightarrow .CC, \$$ $Fi(C\$) = \{c, d\}$

$C \rightarrow .cC, c/d$ (first(C$))

$C \rightarrow .d, c/d$

- I_1 : goto(I_0, S)

$S' \rightarrow S., \$$

- I_2 : goto(I_0, C)

$\alpha \quad B\beta, a$

$S \rightarrow C.C, \$$ $Fi(\$)$

$C \rightarrow .cC, \$$

$C \rightarrow .d, \$$

$C \rightarrow c.C, \$$

$C \rightarrow .cC, \$$

$C \rightarrow .d, \$$

- 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)

  $C \rightarrow d., \$$

- I_8 : goto(I_{3}, C)

  $C \rightarrow cC., c/d$

- I_9 : goto(I_{6}, C)

  $C \rightarrow cC., \$$

Parsing Table

- Construct C = $\{I_0, I_1, I_2 \ldots I_n\}$ 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 grammar
- If goto(I_i , A) = I_j then goto (i, A) = j
- [S' → .S, $] implies an accept action
- All other entries are error

Parsing table - CALR

State	Action			goto	
	c	d	$	S	C
0	s3	S4		1	2
1					
2	s6	s7			5
3	s3	s4			8
4					
5					
6	s6	s7			9

Parsing table - CALR

State	Action			goto	
	c	d	$	S	C
0	s3	s4		1	2
1			accept		
2	s6	s7			5
3	s3	s4			8
4	r3	r3			
5			r1		
6	s6	s7			9

State	Action			goto	
	c	d	$	S	C
7			r3		
8	r2	r2			
9			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 3
0 c 3	c d d $	[3, c] – shift 3
0 c 3 c 3	d d $	[3, d] – shift 4
0 c 3 c 3 d 4	d $	[4, d] – reduce 3, pop 2 symbols from stack, push C, goto(3, C) = 8 C → d
0 c 3 c 3 C 8	d $	[8, d] – reduce 2, pop 4 symbols from the stack, push C, goto(3, C) = 8 C → cC
0 c 3 C 8	d $	[8, d] – reduce 2, pop 4 symbols from the stack, push C, goto(0, C) = 2 C → cC

0C2

Stack	Input	Action
0 C 2	d $	[2, d] – shift 7
0 C 2 d 7	$	[7, $] – reduce 3, pop 2 symbols from the stack, goto(2, C) = 5 $C \to d$
0 C 2 C 5	$	[5, $] – reduce 1, pop 4 symbols off the stack, goto(0, S) = 1 $S \to CC$
0 S 1	$	[1, $] – accept – successful parsing

Example

- $S' \rightarrow S$
- 1 $S \rightarrow L = R$
- 2 $S \rightarrow R$
- 3 $L \rightarrow * R$
- 4 **L → id**
- 5 $R \rightarrow L$

I_0:

$S' \rightarrow .S, \$$

$S \rightarrow .L = R, \$$

$S \rightarrow .R, \$$

$L \rightarrow .*R, =/\$$

$L \rightarrow .id, =/\$$

$R \rightarrow .L, \$$

$FI(=R\$) = \{=\}$

$FI(\$) = \$$

$L \rightarrow .*R, \$$

$L \rightarrow .id, \$$

Another Example

- I_0

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

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

- I_2 : goto(I_0,L)

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

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

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

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

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

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

  [$L \rightarrow$ *****R•, **$**]

Parsing Table

State	Action				goto		
	id	*	=	$	S	L	R
0	s5	s4			1	2	3
1				accept			
2			s6	r5			
3				r2			
4	s5	s4				8	7
5			r4	r4			
6	s12	s11				10	9

State	Action				Goto		
	id	*	=	$	S	L	R
7			r3	r3			
8			r5	r5			
9				r1			
10				r5			
11	s12	s11				10	13
12				r4			
13				r3			

Summary

- CALR – most powerful parser
- Have so many items and states
- No conflicts

Thank you