CS412/413

Introduction to Compilers

Radu Rugina

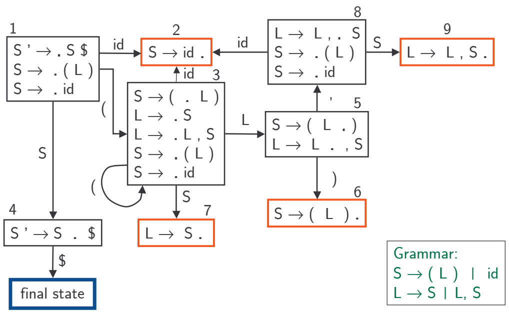
Lecture 9: LR, SLR, and LALR

10 Feb 06

LR Parsing Engine

- Basic mechanism:
  - A set of parser states
  - Use parser stack with symbols and states
    - E.g: $_1$ ($_6$ S $_{10}$ + $_5$
  - Use parsing table to:
    - Determine what action to apply (shift/reduce)
    - Determine the next state
- Table constructed from a DFA of LR states
  - LR state = set of LR items
  - LR item = production with a dot in the RHS

Example LR(0) DFA



LR Parsing Table Example

	(	)	id	,	$	S	L
1	s3		s2			g4	
2	S→id	S→id	S→id	S→id	S→id		
3	s3		s2			g7	g5
4					accept		
5		s6		s8			
6	S→(L)	S→(L)	S→(L)	S→(L)	S→(L)		
7	L→S	L→S	L→S	L→S	L→S		
8	s3		s2			g9	
9	L→L,S	L→L,S	L→L,S	L→L,S	L→L,S		

Build the Parsing Table

- States in the table = states in the DFA
- For a transition S → S' on terminal "a":
  Shift(S') ⊆ Table[S,a]
- For a transition S → S' on non-terminal A:
  Goto(S') ⊆ Table[S,A]
- If S is a reduction state A → γ then:
  Reduce(A → γ) ⊆ Table[S,*]

Parsing Example: ((a),b)

S → (L) | id
L → S | L, S

derivation	stack	input	action
((a),b) ←	$_1$	((a),b)	shift, goto 3
((a),b) ←	$_1$ ($_3$	(a),b)	shift, goto 3
((a),b) ←	$_1$ ($_3$ ($_3$	a),b)	shift, goto 2
((a),b) ←	$_1$ ($_3$ ($_3$ a_2	),b)	reduce S→id
((S),b) ←	$_1$ ($_3$ ($_3$ S_7	),b)	reduce L→S
((L),b) ←	$_1$ ($_3$ ($_3$ L_5	),b)	shift, goto 6
((L),b) ←	$_1$ ($_3$ ($_3$ L_5)$_6$	,b)	reduce S→(L)
(S,b) ←	$_1$ ($_3$ S_7	,b)	reduce L→S
(L,b) ←	$_1$ ($_3$ L_5	,b)	shift, goto 8
(L,b) ←	$_1$ ($_3$ L_5 ,$_8$	b)	shift, goto 9
(L,b) ←	$_1$ ($_3$ L_5 ,$_8$ b_2	)	reduce S→id
(L,S) ←	$_1$ ($_3$ L_5 ,$_8$ S_9	)	reduce L→L , S
(L) ←	$_1$ ($_3$ L_5	)	shift, goto 6
(L) ←	$_1$ ($_3$ L_5)$_6$		reduce S→(L)
S	$_1$ S_4	$	done

Shift Operations

- When shifting terminal “a” onto the stack:
  - Let S be the current DFA state
  - follow DFA transition from S on symbol a
    - S must contain an item of the form $B \rightarrow \gamma\,.\,a\,\delta$
    - Actions[S,a] is of the form shift(S’)
    - Push a, then S’ onto the stack

- Example:

((a),b) ← $_1\,(_3\,(_3$ a),b) shift, goto 2
((a),b) ← $_1\,(_3\,(_3\,a_2$),b)

Reductions

- When reducing $A \rightarrow \beta$ with stack $\alpha\beta$:
  - pop β off stack, revealing prefix α and top state S
  - follow DFA transition from S on symbol A
    - S must contain an item of the form $B \rightarrow \gamma\,.\,A\,\delta$
    - push A, then push Goto[S,A] onto stack

- Example:

((a),b) ← $_1\,(_3\,(_3$ a),b) shift, goto 2
((a),b) ← $_1\,(_3\,(_3\,a_2$),b) reduce S→id
((S),b) ← $_1\,(_3\,(_3\,S_7$),b)

LR(0) Summary

- LR(0) parsing recipe:
  Start with an LR(0) grammar
  Compute LR(0) states and build DFA:
  Build the LR(0) parsing table from the DFA

- This process can be automated, i.e. we can build parser generator tools

LR(0) Limitations

- An LR(0) machine only works if states with reduce actions have a single reduce action -- in those states, always reduce ignoring lookahead
- With more complex grammar, construction gives states with shift/reduce or reduce/reduce conflicts
- Need to use look-ahead to choose

ok	shift /reduce	reduce / reduce
L → L , S .	L → L , S . S → S . , L	L → S , L . L → S .

LR(0) Parsing Table

	(	)	id	,	$	S	L
1	s3		s2			g4	
2	S→id	S→id	S→id	S→id	S→id		
3	s3		s2			g7	g5
4					accept		
5		s6		s8			
6	S→(L)	S→(L)	S→(L)	S→(L)	S→(L)		
7	L→S	L→S	L→S	L→S	L→S		
8	s3		s2			g9	
9	L→L,S	L→L,S	L→L,S	L→L,S	L→L,S		

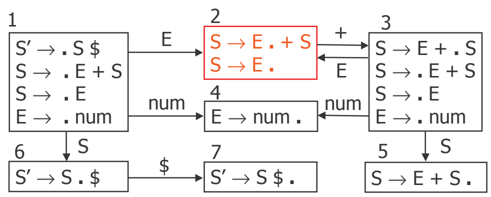
A Non-LR(0) Grammar

- Grammar for addition of numbers:
  S → S + E | E
  E → num

- Left-associative is LR(0)

- Right-associative version is not LR(0)
  S → E + S | E
  E → num

LR(0) Parsing Table



What to do in state 2: shift or reduce?

	num	+	$	E	S
1	s4			g2	g6
2	S→E	s3/S→E	S→E		

SLR Parsing

- SLR Parsing = easy extension of LR(0)
  - For each reduction A → γ: look at the next symbol "c"
  - Apply reduction only if "c" is in FOLLOW(A)
- SLR parsing table eliminates some conflicts
  - Same as LR(0) table except reduction rows
  - Adds reductions A → γ only in the columns of symbols in FOLLOW(A)
- Example:

  FOLLOW(S)={$}

	num	+	$	E	S
1	s4			g2	g6
2		s3	S→E		

SLR Parsing Table

- Reductions do not fill entire rows
- Otherwise, same as LR(0)

	num	+	$	E	S
1	s4			g2	g6
2		s3	S→E		
3	s4			g2	g5
4			S→E		
5			S→E+S		
6			s7		
7			accept		

LR(1) Parsing

- Get as much power as possible out of 1 look-ahead symbol parsing table
- LR(1) grammar = recognizable by a shift/reduce parser with 1 look-ahead
- LR(1) parsing uses similar concepts as LR(0)
  - Parser states = sets of items
  - LR(1) item = LR(0) item + look-ahead symbol possibly following production

LR(0) item : S → . S + E

LR(1) item : S → . S + E   +

LR(1) States

- LR(1) state = set of LR(1) items
- LR(1) item = (X → α . β , y)
- Meaning: α already matched at top of the stack; next expect to see β y
- Shorthand notation

  (X → α . β , {x_1, ..., x_n})

  means:

  (X → α . β , x_1)

  ...

  (X → α . β , x_n)

  S → S . + E   +,$
  S → S + . E   num
- Extend closure and goto operations

LR(1) Closure

- LR(1) closure operation:
  - Start with Closure(S) = S
  - For each item in S:

    X → α . Y β ,   z

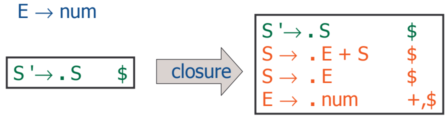
    and for each production Y → γ, add the following item to the closure of S:

    Y → . γ ,   FIRST(βz)
  - Repeat until nothing changes
- Similar to LR(0) closure, but also keeps track of the look-ahead symbol

LR(1) Start State

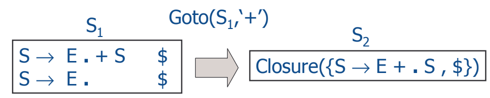
- Initial state: start with (S' → . S , $), then apply the closure operation
- Example: sum grammar

S' → S $
S → E + S | E
E → num



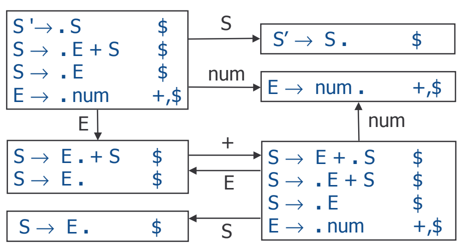
LR(1) Goto Operation

- LR(1) goto operation = describes transitions between LR(1) states
- Algorithm: for a state S and a symbol Y
  - S' = {(X → αY.β, z) | (X → α.Yβ, z) ∈ S}
  - Goto(S, Y) = Closure(S')



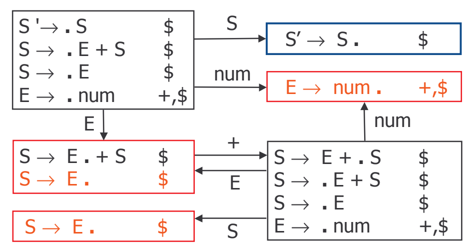
LR(1) DFA Construction

- If S' = goto (S,x) then add an edge labeled x from S to S'



LR(1) Reductions

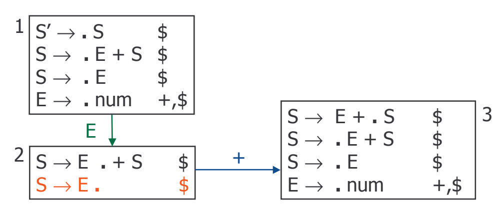
- Reductions correspond to LR(1) items of the form (X → γ . , y)



LR(1) Parsing Table Construction

- Same as construction of LR(0) parsing table, except for reductions
- For a transition S → S' on terminal x:
  Shift(S') ⊆ Table[S,x]
- For a transition S → S' on non-terminal N:
  Goto(S') ⊆ Table[S,N]
- If (X → γ . , y) ∈ S, then:
  Reduce(X → γ) ⊆ Table[S,y]

LR(1) Parsing Table Example



Fragment of the Parsing table:

	+	$	E
1			2
2	s3	S→E	

LALR(1) Grammars

- Problem with LR(1): too many states
- LALR(1) Parsing (Look-Ahead LR)
  - Constructs LR(1) DFA and then merge any two LR(1) states whose items are identical except look-ahead
  - Results in smaller parser tables
  - Theoretically less powerful than LR(1)

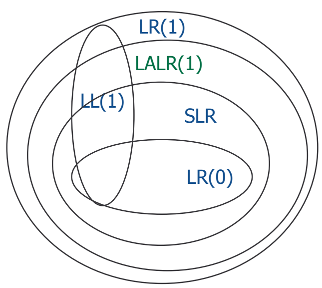
$$\begin{array}{|l|}\hline S \to id\ .\ \ + \\ S \to E\ .\ \ \$ \\ \hline\end{array} + \begin{array}{|l|}\hline S \to id\ .\ \ \$ \\ S \to E\ .\ \ + \\ \hline\end{array} = ?$$

- LALR(1) Grammar = a grammar whose LALR(1) parsing table has no conflicts

LL/LR Grammar Summary

- LL parsing tables
  - Nonterminals x terminals → productions
  - Computed using FIRST/FOLLOW
- LR parsing tables
  - LR states x terminals → shift/reduce
  - LR states x non-terminals → goto
  - Computed using closure/goto operations on LR states
  - LR(0), LR(1) = basic approaches
  - SLR, LALR(1) = variations

Classification of Grammars



$LR(k) \subseteq LR(k+1)$
$LL(k) \subseteq LL(k+1)$
$LL(k) \subseteq LR(k)$

$LR(0) \subseteq SLR$

$LALR(1) \subseteq LR(1)$