

CS412/CS413

Introduction to Compilers Tim Teitelbaum

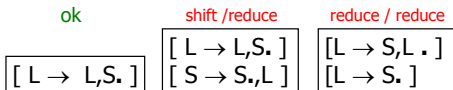
Lecture 10: LR Parsing
February 14, 2005

LR(0) Parsing Summary

- LR(0) state = set of LR(0) items
- LR(0) item = a production with a dot in RHS
- Compute LR(0) states and build DFA:
 - Use the closure operation to compute states
 - Use the goto operation to compute transitions between states
- Build the LR(0) parsing table from the DFA
- Use the LR(0) parsing table to determine whether to reduce or to shift

LR(0) Limitations

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



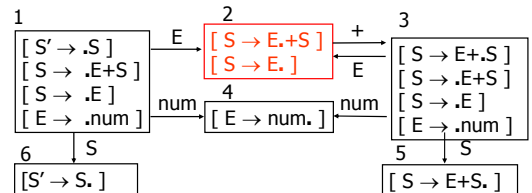
LR(0) Parsing Table

	()	id	,	ϵ	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:
 $S \rightarrow S + E \mid E$
 $E \rightarrow \text{num}$
- Left-associative version is LR(0)
- Right-associative version is **not** LR(0)
 $S \rightarrow E + S \mid E$
 $E \rightarrow \text{num}$

LR(0) Parsing Table



What to do in state 2:
shift or reduce?

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

SLR(1) Parsing

- SLR Parsing = easy extension of LR(0)
 - For each reduction $A \rightarrow \beta$, look at the next symbol c
 - Apply reduction only if c is in $FOLLOW(A)$, or $c = \epsilon$ and $S \Rightarrow^* \gamma A$
- SLR parsing table eliminates some conflicts
 - Same as LR(0) table except reduction rows
 - Adds reductions $A \rightarrow \beta$ only in the columns of symbols in $FOLLOW(A)$
- Example:

$FOLLOW(S) = \{ \}$
but $S \Rightarrow^* \gamma E$

	num	+	ϵ	E	S
1	s4			g2	g6
2	s3	$S \rightarrow E$			

SLR Parsing Table

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

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

SLR(k)

- Use the LR(0) machine states as rows of table
- Let Q be a state and u be a lookahead string
 - Action(Q, u) = shift $Goto(Q, b)$
 - if Q contains an item of the form $[A \rightarrow \beta, b\beta_1]$, with $u \in FIRST_k(b\beta_1, FOLLOW_k(A))$
 - Action(Q, u) = accept
 - if $Q = \{ [S' \rightarrow S,] \}$ and $u = \epsilon$
 - Action(Q, u) = reduce i
 - if Q contains the item $[A \rightarrow \beta, \cdot]$, where $A \rightarrow \beta$ is the ith production of G and $u \in FOLLOW_k(A)$, or $u = \epsilon$ and $S \Rightarrow^* \gamma A$
 - Action(Q, u) = error otherwise
- G is SLR(k) iff the Action function given above is single-valued for all Q and u , i.e., there are no shift-reduce or reduce-reduce conflicts.

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 following the production

LR(0) item : $[S \rightarrow \cdot S+E]$

LR(1) item : $[S \rightarrow \cdot S+E \quad +]$

LR(1) States

- LR(1) state = set of LR(1) items
- LR(1) item = $[A \rightarrow \alpha \cdot \beta \quad b]$, where $b \in \Sigma \cup \{ \epsilon \}$
- Meaning: α already matched at top of the stack; next expect to see βb
- Shorthand notation

$[A \rightarrow \alpha \cdot B \quad b_1, \dots, b_n]$
means:
 $[A \rightarrow \alpha \cdot \beta \quad b_1]$
...
 $[A \rightarrow \alpha \cdot \beta \quad b_n]$
- Extend closure and goto operations

$[S \rightarrow S, +E$	$+ , \epsilon]$
$[S \rightarrow S + E$	num

LR(1) Closure

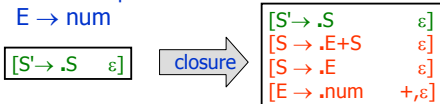
- LR(1) closure operation:
 - Start with $Closure(S) = S$
 - For each item in S :

$[A \rightarrow \alpha \cdot B\beta \quad b]$
and for each production $B \rightarrow \gamma$, add the following item to the closure of S :
 $[B \rightarrow \cdot \gamma \quad FIRST(\beta b)]$
or $[B \rightarrow \cdot \gamma \quad \epsilon]$ if $FIRST(\beta b) = \{ \}$
 - Repeat until nothing changes
- Similar to LR(0) closure, but also keeps track of the look-ahead symbol

LR(1) Start State

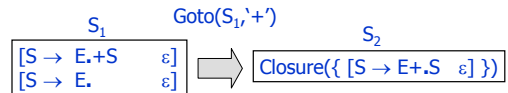
- Initial state: start with $[S' \rightarrow .S \ \epsilon]$, then apply the closure operation
- Example: sum grammar

$S' \rightarrow S$
 $S \rightarrow E+S \mid E$
 $E \rightarrow \text{num}$



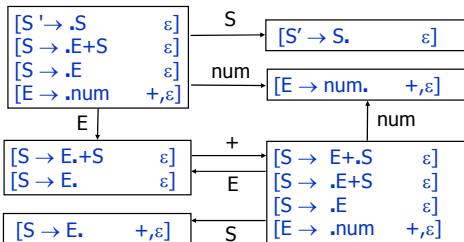
LR(1) Goto Operation

- LR(1) goto operation = describes transitions between LR(1) states
- Algorithm: for a state S and a symbol Y
 - $S' = \{ [A \rightarrow \alpha Y \beta \ b] \mid [A \rightarrow \alpha Y \beta \ b] \in S \}$
 - $\text{Goto}(S, Y) = \text{Closure}(S')$



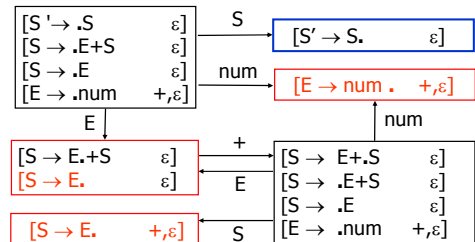
LR(1) DFA Construction

- If $S' = \text{Goto}(S, X)$ then add an edge labeled X from S to S'



LR(1) Reductions

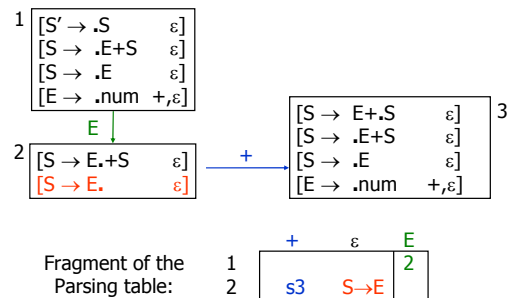
- Reductions correspond to LR(1) items of the form $[A \rightarrow \beta \ \cdot \ x]$



LR(1) Parsing Table Construction

- Same as construction of LR(0) parsing table, except for reductions
- If $[A \rightarrow \beta \ \cdot \ b] \in \text{state } Q$, then:
 - Action(Q, b) is Reduce($A \rightarrow \beta$)

LR(1) Parsing Table Example



LR(1) but not SLR(1)

- Let G have productions

$S \rightarrow aAb \mid Ac$

$A \rightarrow a \mid \epsilon$

- $V(a) = \{$

$[S \rightarrow a.Ab]$ $FOLLOW(A) = \{b,c\}$
 $[A \rightarrow a.]$
 $[A \rightarrow .a]$
 $[A \rightarrow .]$
 $\}$

} **reduce-reduce conflict**

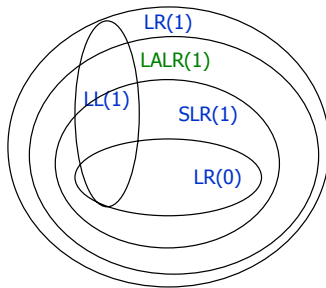
LALR(1) Grammars

- Problem with LR(1): too many states
- LALR(1) Parsing** (Look-Ahead LR)
 - Construct 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)

$[S \rightarrow id. +]$ + $[S \rightarrow id. \epsilon]$
 $[S \rightarrow E. \epsilon]$ + $[S \rightarrow E. +]$ = ?

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

Classification of Grammars



$LR(k) \subseteq LR(k+1)$
 $LL(k) \subseteq LL(k+1)$
 $LL(k) \subseteq LR(k)$
 $LR(0) \subseteq SLR(1)$
 $LALR(1) \subseteq LR(1)$

Automate the Parsing Process

- Can automate:
 - The construction of LR parsing tables
 - The construction of shift-reduce parsers based on these parsing tables
- Automatic parser generators: **yacc, bison, CUP**
- LALR(1) parser generators
 - No much difference compared to LR(1) in practice
 - Smaller parsing tables than LR(1)
 - Augment LALR(1) grammar specification with declarations of precedence, associativity
- output: LALR(1) parser program

Associativity

$S \rightarrow S + E \mid E$
 $E \rightarrow \text{num}$

➔

$E \rightarrow E + E$
 $E \rightarrow \text{num}$

What happens if we run this grammar through LALR construction?

Shift/Reduce Conflict

$E \rightarrow E + E$
 $E \rightarrow \text{num}$

$[E \rightarrow E+E. +]$
 $[E \rightarrow E.+E +, \epsilon]$

➔

shift/reduce conflict

shift: $1+(2+3)$
 reduce: $(1+2)+3$

$1+2+3$
 \wedge

Grammar in CUP

nonterminal E; terminal PLUS, LPAREN...
precedence **left PLUS**;

“when shifting '+' conflicts with
reducing a production, choose reduce”

```
E ::= E PLUS E
    | LPAREN E RPAREN
    | NUMBER ;
```

Precedence

- CUP can also handle operator precedence

```
E → E + E | T
T → T × T | num | ( E )
```



```
E → E + E | E × E
    | num | ( E )
```

Conflicts without Precedence

```
E → E + E | E × E
    | num | ( E )
```

[E → E.+E ...]	[E → E+E. ×]
[E → E×E. +]	[E → E.×E ...]

Precedence in CUP

```
precedence left PLUS;
precedence left TIMES; // TIMES > PLUS
E ::= E PLUS E | E TIMES E | ...
```

RULE: in conflict, choose **reduce** if production symbol
higher precedence than shifted symbol; choose **shift** if
vice-versa

[E → E.+E ...]	[E → E+E. ×]
[E → E×E. +]	[E → E.×E ...]

reduce E→E×E

Shift ×

Summary

- Look-ahead information makes SLR(1), LALR(1), LR(1) grammars expressive
- Automatic parser generators support LALR(1) grammars
- Precedence, associativity declarations simplify grammar writing