CS412/CS413

Introduction to Compilers

Tim Teitelbaum

Lecture 9: LR Parsing

February 9, 2007

LR(k) Grammars

- LR(k) = Left-to-right scanning, Right-most derivation, k look-ahead characters
- Main cases: LR(0), LR(1), SLR(k), and LALR(1)
- Parsers for LR(0) Grammars:
  - Know whether to shift or reduce without consulting the lookahead symbol
  - Give intuition and techniques relevant for creating parsers for all grammar classes to be considered

Building LR(0) Parsing Tables

- To build the parsing table:
  - Define states of the parser
  - Build a DFA to describe the transitions between states
  - Use the DFA to build the parsing table

Viable Prefix

- γ is a viable prefix for G iff there is some derivation

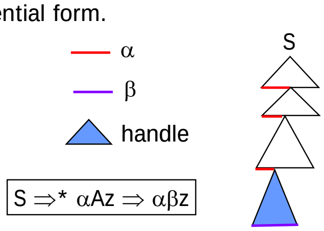
  $S \Rightarrow^* \alpha A z \Rightarrow \alpha\beta z$

  where γ is a prefix of $\alpha\beta$

- $\{\gamma \mid \gamma$ is a viable prefix of G$\}$ is a regular language, i.e., it can be recognized by a DFA known as the Canonical LR(0) Machine

Viable Prefix (Informally)

- γ is a viable prefix for G if it is a prefix of a sentential form derived from S that does not extend past the end of the handle of the sentential form.



LR(0) Items

- An LR(0) item for G is a triple $\langle A, \beta_1, \beta_2\rangle$ such that $A\rightarrow\beta_1\beta_2$ is a production of G. The item $\langle A, \beta_1, \beta_2\rangle$ is denoted by $[A\rightarrow\beta_1 . \beta_2]$

Validity of LR(0) Items

- The item $[A \to \beta_1 . \beta_2]$ is valid for viable prefix $\alpha\beta_1$ iff $S \Rightarrow^* \alpha A z \Rightarrow \alpha\beta_1\beta_2 z$
- Note:
  - β_1 may be ε
  - β_2 may be ε
- For any viable prefix α, let $V(\alpha)$ denote the set of LR(0) items that are valid for α.

Sets of Valid Items

- Observations
  - There are only finitely many distinct LR(0) items for a given G.
  - Thus, there are only finitely many sets of LR(0) items for G.
- Sets of valid items for viable prefixes of G will serve as the states of a DFA, i.e., the canonical LR(0) machine.

Relation $\downarrow$

- The relation $\downarrow$ on LR(0) items is defined by $I \downarrow I'$ iff $\exists$ A, B, β_1, β_2, β_3 such that

  $I = [A \to \beta_1 . B\beta_3]$

  $I' = [B \to .\beta_2]$
- Lemma. Let I, I' be as above. If $I \in V(\alpha\beta_1)$ and $I \downarrow I'$, then $I' \in V(\alpha\beta_1)$.
  - $I \in V(\alpha\beta_1)$ implies $S \Rightarrow^* \alpha A z \Rightarrow \alpha\beta_1 B\beta_3 z$
  - Assuming G has no useless productions, $\exists$y such that $\beta_3 \Rightarrow^* y$
  - Thus, $S \Rightarrow^* \alpha A z \Rightarrow \alpha\beta_1 B\beta_3 z \Rightarrow^* \alpha\beta_1 Byz \Rightarrow \alpha\beta_1\beta_2 yz$
  - Thus, I' (i.e., $[B \to .\beta_2]$) $\in V(\alpha\beta_1)$

Relation $\to_X$

- For any $X \in (V \cup \Sigma)$, the relation $\to_X$ is defined by $I \to_X I'$ iff $\exists$ A, β_1, β_3 such that

  $I = [A \to \beta_1 . X\beta_3]$

  $I' = [A \to \beta_1 X . \beta_3]$
- Lemma. Let I, I' be as above. If $I \in V(\alpha\beta_1)$ then $I' \in V(\alpha\beta_1 X)$.
  - $I = [A \to \beta_1 . X\beta_3] \in V(\alpha\beta_1)$ implies

    $S \Rightarrow^* \alpha A z \Rightarrow \alpha\beta_1 X\beta_3 z$

    which by definition means I' (= $[A \to \beta_1 X . \beta_3]$) $\in V(\alpha\beta_1 X)$

Technical Details

- Start symbol never appears on RHS
  - It is convenient if the start symbol never appears on the RHS of any production.
  - Given $G = \langle V,\Sigma,S,\to\rangle$, let $S' \notin V$ and

    $G' = \langle V,\Sigma,S',\to \cup \{S' \to S\}\rangle$
  - Assume that the grammars we work with have the form of G'.
- If S is a set and R is a relation, then

  $SR = \{y \mid x \in S \text{ and } \langle x,y\rangle \in R\}$

  SR is called S mapped by R

$V(\varepsilon)$, the base case

- Let S' be the start symbol of G. Then
  - $V(\varepsilon) = \{ [S' \to .S] \}\downarrow^*$

  (i.e., the "initial item" of G $\{[S' \to .S]\}$ mapped by the reflexive transitive closure of the $\downarrow$ relation.)
- If Q is a set of items, we call $Q\downarrow^*$ the closure(Q).

V(αX), the inductive case

- For any α and X, $V(\alpha X) = V(\alpha)\rightarrow_X\downarrow^*$
- For any set Q of items, we call $Q\rightarrow_X\downarrow^*$ the X-successor of Q, or Goto(Q,X).

Canonical LR(0) Machine

- <u>States</u>: Sets of valid items
- <u>Transition function</u>: Goto, as defined above.
- <u>Algorithm</u>: To compute all sets of valid items

  STATES := $V(\varepsilon)$
  while $\exists$ Q $\in$ STATES, X$\in$(V$\cup\Sigma$) such that
  Goto(Q,X) $\notin$ STATES
  do STATES := STATES $\cup$ { Goto(Q,X) }

- Clearly, this terminates, as STATES is bounded above by the Powerset(LR(0) items)

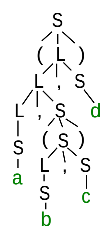
LR(0) Grammar

- Nested lists:

  $S \rightarrow$ (L) | id
  $L \rightarrow$ S | L,S

- Sample strings
  - (a,b,c)
  - ((a,b),(c,d),(e,f))
  - (a,(b,c,d),((f,g)))

Parse tree for (a, (b,c), d)



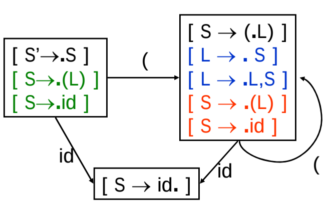
Start State

Grammar
$S \rightarrow$ (L) | id
$L \rightarrow$ S | L, S

- Start state
  - $V(\varepsilon)$ = { [S'→.S] }$\downarrow^*$
    = { [S'→.S] [S→.(L)], [S→.id] }
- Closure of a parser state Q:
  - Start with Closure(Q) = Q
  - Then for each item in Q:
    $A \rightarrow \alpha.B\beta$
    add the items for all the productions $B \rightarrow \gamma$ to the closure of Q:
    $B \rightarrow .\gamma$

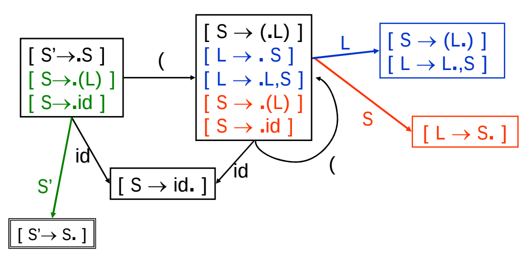
Goto: Terminal Symbols

Grammar
$S \rightarrow$ (L) | id
$L \rightarrow$ S | L,S



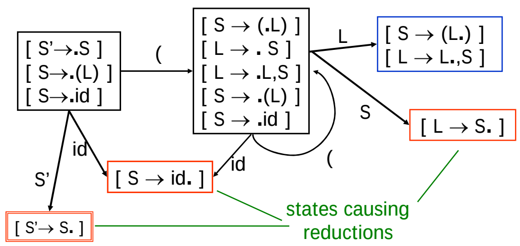
In new state, include all items that have appropriate input symbol just after dot, advance dot in those items, and take closure.

Goto: Nonterminal Symbols

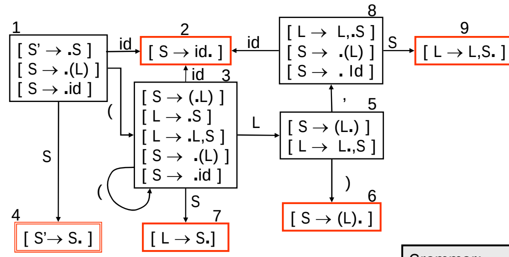


(same algorithm for transitions on nonterminals)

Reduce States



Full LR(0) Machine



Grammar:
S → (L) | id
L → S | L, S

Parsing Example: ((a),b)

Grammar:
S → (L) | id
L → S | L, S

derivation	stack	input	action
((a),b) ⇐	1	((a),b)	shift, goto 3
((a),b) ⇐	13	(a),b)	shift, goto 3
((a),b) ⇐	133	a),b)	shift, goto 2
((a),b) ⇐	1332	),b)	reduce S→id
((S),b) ⇐	1337	),b)	reduce L→S
((L),b) ⇐	1335	),b)	shift, goto 6
((L),b) ⇐	13356	,b)	reduce S→(L)
(S,b) ⇐	137	,b)	reduce L→S
(L,b) ⇐	135	,b)	shift, goto 8
(L,b) ⇐	1358	b)	shift, goto 9
(L,b) ⇐	13582	)	reduce S→id
(L,S) ⇐	13589	)	reduce L→L , S
(L) ⇐	135	)	shift, goto 6
(L) ⇐	1356		reduce S→(L)
S	14		done

Reductions

- On reducing $B \to \beta$ with stack $\alpha\beta_2$:
  - pop $|\beta|$ states off stack
  - This reveals topmost state Q, which contains an item $[A \to \beta_1 . B\beta_3]$
  - push state Goto(Q,B) onto the stack

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		

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

LR(0) Limitations

- An LR(0) machine only works if each state with a reduce action has only one possible reduce action and no shift action
- With more complex grammars, construction gives states with shift/reduce or reduce/reduce conflicts
- Need to use look-ahead to choose

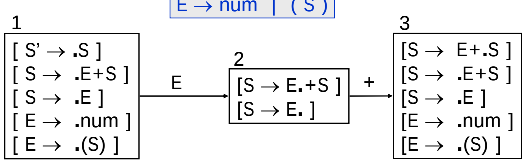
ok	shift /reduce	reduce / reduce
[L→ L,S.]	[L → L,S.] [S → S.,L]	[L → S,L.] [L → S.]

A Non-LR(0) Grammar

- Grammar for addition of numbers:
  S → S + E | E
  E → num | (S)
- Left-associative is LR(0)
- Right-associative version is not LR(0)
  S → E + S | E
  E → num | (S)

LR(0) Parsing Table

Grammar
S → E + S | E
E → num | (S)



What to do in state 2?

	+	ε	E
1			2
2	s3/S→E	S→E	

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→ β_1.bβ_3], with u ∈ FIRST$_k$(bβ_3 FOLLOW$_k$(A))
  - Action(Q,u) = accept
    if Q = { [S'→S] } and u=ε
  - Action(Q,u) = reduce i
    if Q contains the item [A→ β.], where A→ β is the ith production of G and u ∈ FOLLOW$_k$(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.

Next Time

- Learn about other kinds of LR parsing:
  - SLR = improved LR(0)
  - LR(1) = 1 character lookahead
  - LALR(1) = Look-Ahead LR(1)
- Basic ideas are the same as for LR(0)
  - Parser states with LR items
  - DFA with transitions between parser states
  - Parser table with shift/reduce/goto actions