

Bottom-Up Parsing

Compiler Construction, Lecture 9

K.R. Chowdhary, Professor of CS

September 23, 2019 | Fall 2019

Reductions and Handle Pruning

Bottom-up parsing **reduces** the input to the start symbol: a rightmost derivation in reverse. Key decisions: **when** to reduce and **which** production to use.

Grammar

$$\begin{aligned} E &\rightarrow E + T \mid T \\ T &\rightarrow T * F \mid F \\ F &\rightarrow (E) \mid \text{num} \mid \text{id} \end{aligned}$$

A **handle** is a substring matching a production body whose reduction is one step of the reverse rightmost derivation. If $S \Rightarrow_{rm}^* \alpha A w \Rightarrow_{rm} \alpha \beta w$, then $A \rightarrow \beta$ is a handle of $\alpha \beta w$; w has only terminals. An unambiguous grammar has exactly one handle per right-sentential form.

Right-sentential form	Handle	Reduction
$id_1 + id_2$	id_1	$F \rightarrow id$
$F + id_2$	F	$T \rightarrow F$
$T + id_2$	T	$E \rightarrow T$
$E + id_2$	id_2	$F \rightarrow id$
$E + F$	F	$T \rightarrow F$
$E + T$	$E + T$	$E \rightarrow E + T$

Handle pruning: locate the handle, replace it by its head, repeat until only E remains.

Shift-Reduce Parsing

A stack holds grammar symbols (top at the right) and the input buffer holds the rest. Start: stack \$, input w \$. Accept: stack \$\$, input \$.

Four operations

1. **Shift** the next input symbol onto the stack
2. **Reduce**: the handle is on top of the stack; replace it by the head
3. **Accept**
4. **Error**: call error recovery

The handle always appears on top of the stack, never inside. For n input symbols and r reductions, there are $n + r$ handle-finding steps, one per parse-tree node.

Stack	Input	Action
\$	$id_1 + id_2$ \$	shift
\$ id_1	$+ id_2$ \$	reduce $F \rightarrow id$
\$ F	$+ id_2$ \$	reduce $T \rightarrow F$
\$ T	$+ id_2$ \$	reduce $E \rightarrow T$
\$ E	$+ id_2$ \$	shift
\$ $E +$	id_2 \$	shift
\$ $E + id_2$	\$	reduce $F \rightarrow id$
\$ $E + F$	\$	reduce $T \rightarrow F$
\$ $E + T$	\$	reduce $E \rightarrow E + T$
\$ E	\$	accept

LR(k): Left-to-right scan, Rightmost derivation in reverse, k look-ahead symbols. LR covers virtually all programming-language constructs and detects errors as early as possible; its parsers are built by tools such as YACC.

LR(0) items

$A \rightarrow XYZ$ gives four items:

$A \rightarrow \cdot XYZ$, $A \rightarrow X \cdot YZ$, $A \rightarrow XY \cdot Z$, $A \rightarrow XYZ \cdot$.

$A \rightarrow \varepsilon$ gives only $A \rightarrow \cdot$. The dot marks how much of the body has been seen.

Augmented grammar: add $S' \rightarrow S$; accept when about to reduce by it. The **canonical collection** starts from $\text{CLOSURE}(\{S' \rightarrow \cdot S\})$ and adds GOTO sets until none are new.

CLOSURE(I) and GOTO(I, X)

- **CLOSURE**: if $A \rightarrow \alpha \cdot B \beta$ is in the set, add $B \rightarrow \cdot \gamma$ for every B -production, until nothing new
- **GOTO(I, X)**: CLOSURE of all $A \rightarrow \alpha X \cdot \beta$ such that $A \rightarrow \alpha \cdot X \beta \in I$

Example: $I = \{E' \rightarrow E \cdot, E \rightarrow E \cdot + T\}$.

$\text{GOTO}(I, +) = \{E \rightarrow E + \cdot T, T \rightarrow \cdot T * F, T \rightarrow \cdot F, F \rightarrow \cdot (E), F \rightarrow \cdot id\}$.

Kernel items: $S' \rightarrow \cdot S$ and items with the dot not at the left end. **Non-kernel** items are regenerated by closure.

LR(0) Automaton for the Expression Grammar

State	Kernel items	GOTO on
0	$E' \rightarrow \cdot E$	$E:1, T:8, F:5, (: 3, id:4$
1	$E' \rightarrow E \cdot, E \rightarrow E \cdot + T$	$+:2$
2	$E \rightarrow E + \cdot T$	$T:6, F:5, (: 3, id:4$
3	$F \rightarrow (\cdot E)$	$E:7, T:8, F:5, (: 3, id:4$
4	$F \rightarrow id \cdot$	none
5	$T \rightarrow F \cdot$	none
6	$E \rightarrow E + T \cdot, T \rightarrow T \cdot * F$	$*:10$
7	$F \rightarrow (E \cdot), E \rightarrow E \cdot + T$	$):11, +:2$
8	$E \rightarrow T \cdot, T \rightarrow T \cdot * F$	$*:10$
9	$T \rightarrow T * F \cdot$	none
10	$T \rightarrow T * \cdot F$	$F:9, (: 3, id:4$
11	$F \rightarrow (E) \cdot$	none

Reading the automaton

- States are sets of items; edges are GOTO
- Shift if the state has a transition on the next symbol; otherwise reduce by a completed item
- States 4, 5, 9, 11 have no transitions: pure reduce states
- Each state except 0 has a unique grammar symbol

SLR Parsing Table

	ACTION						GOTO		
	id	+	*	(	)	\$	E	T	F
0	s4			s3			1	8	5
1		s2				acc			
2	s4			s3				6	5
3	s4			s3			7	8	5
4		r6	r6		r6	r6			
5		r4	r4		r4	r4			
6		r1	s10		r1	r1			
7		s2			s11				
8		r2	s10		r2	r2			
9		r3	r3		r3	r3			
10	s4			s3					9
11		r5	r5		r5	r5			

Productions: (1) $E \rightarrow E + T$, (2) $E \rightarrow T$, (3) $T \rightarrow T * F$, (4) $T \rightarrow F$, (5) $F \rightarrow (E)$, (6) $F \rightarrow id$

Building the table

- $A \rightarrow \alpha \cdot a \beta \in I_i$, $GOTO(I_i, a) = I_j$:
ACTION[i, a] = shift j
- $A \rightarrow \alpha \cdot \in I_i$: reduce $A \rightarrow \alpha$ for all
 $a \in FOLLOW(A)$
- $S' \rightarrow S \cdot$: ACTION[$i, \$$] = accept
- $GOTO(I_i, A) = I_j$: $GOTO[i, A] = j$
- Empty entries are errors

Parser moves

Configuration: (stack of states, remaining input). **Shift** pushes s . **Reduce** $A \rightarrow \beta$ pops $|\beta|$ states, then pushes $GOTO[s_{m-r}, A]$.

Example: Parsing $id * id$ and Review

	State stack	Input	Action
1	0	$id * id\$$	shift 4
2	0 4	$*id\$$	reduce $F \rightarrow id$
3	0 5	$*id\$$	reduce $T \rightarrow F$
4	0 8	$*id\$$	shift 10
5	0 8 10	$id\$$	shift 4
6	0 8 10 4	$\$$	reduce $F \rightarrow id$
7	0 8 10 9	$\$$	reduce $T \rightarrow T * F$
8	0 8	$\$$	reduce $E \rightarrow T$
9	0 1	$\$$	accept

Each reduce pops one state per body symbol, then pushes GOTO[exposed state, head].

Review and exercises

1. What is “derivation in reverse”?
What is handle pruning?
2. Shift-reduce vs. reduce-reduce conflicts?
3. Why augment the grammar? Kernel vs. non-kernel items?
4. Handles in $S \rightarrow 0S1 \mid 01$ for 000111; parse it bottom-up
5. Parse $id * id + id$ by shift-reduce
6. SLR table for $T \rightarrow B \mid \{L\}$,
 $L \rightarrow TL \mid B$, $B \rightarrow a \mid b$; parse $\{a\{ba\}\}$
7. Canonical LR vs. SLR: recognition power?