Compilers

by
Marwa Yusuf

Lecture 8
Tues. 13-4-2021

Chapter 4 (4.6.2 to 4.6.5)

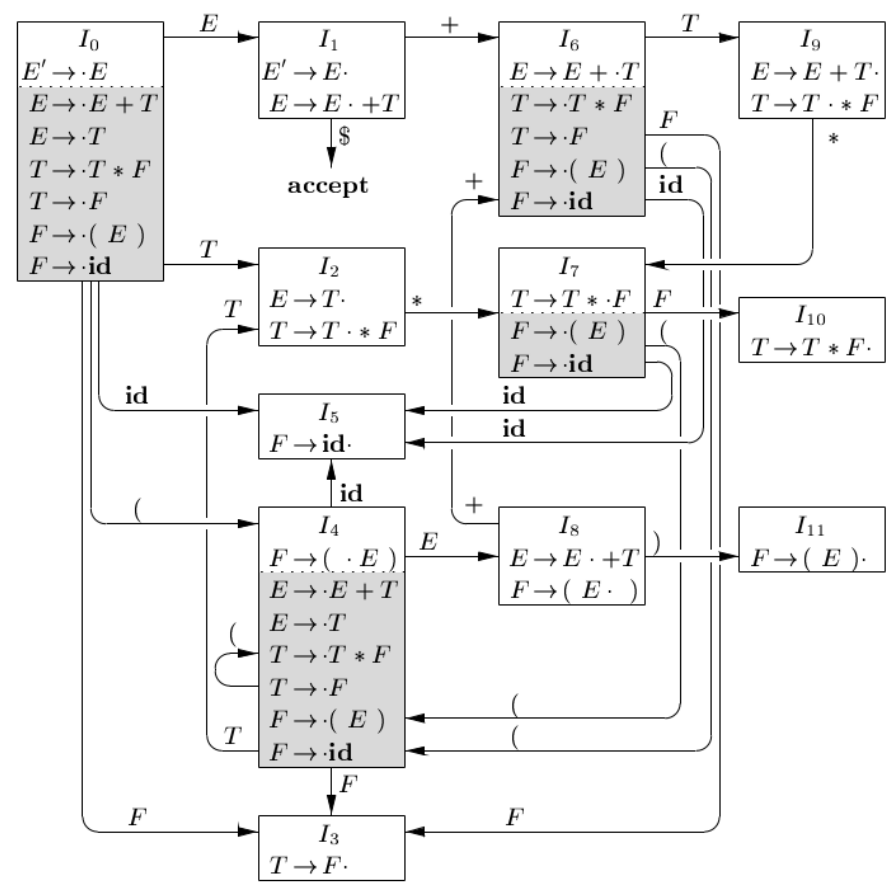
Syntax Analysis

Use of LR(0) AUtomaton

- SLR (Simple LR) parsing: construction from the grammar of the LR(0) automaton.
- States = sets of items, transitions = GOTO function.
- The start state is the CLOSURE({[S' → ·S]}).
- $State_j$ = set of items I_j.
- Shift-reduce parsing decisions: at some state, shift on next input symbol if there is a transition for that symbol, otherwise reduce.

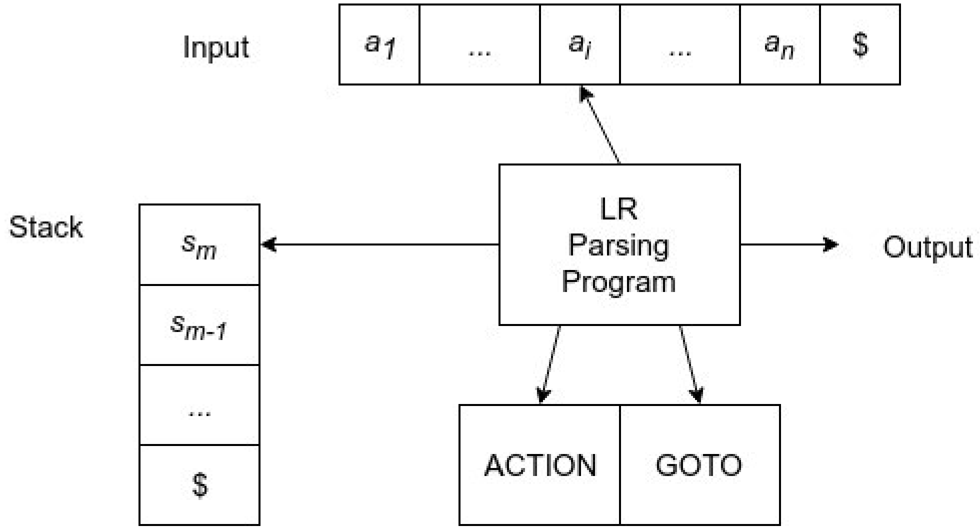
Example

LI N E	STACK	SYMB OLS	INPUT	ACTION
(1)	0	$	**id** * **id** $	shift to 5
(2)	0 5	$ **id**	* **id** $	reduce by $F \rightarrow$ **id**
(3)	0 3	$ F	* **id** $	reduce by $T \rightarrow F$
(4)	0 2	$ T	* **id** $	shift to 7
(5)	0 2 7	$ T *	**id** $	shift to 5
(6)	0 2 7 5	$ T * **id**	$	reduce by $F \rightarrow$ **id**
(7)	0 2 7 10	$ $T * F$	$	reduce by $T \rightarrow T * F$
(8)	0 2	$ T	$	reduce by $E \rightarrow T$
(9)	0 1	$ E	$	accept



The LR-Parsing Algorithm

- Driver: the same for all LR parsers, parsing table changes.
- Shift state (states from LR(0) automaton): each one represents info on the stack below, each has a corresponding grammar symbol.



Structure of the LR Parsing Table

- ACTION function: ACTION[i, a] =
  - Shift j: shift a but use state as a representative.
  - Reduce $A \rightarrow \beta$: β is on top of stack.
  - Accept.
  - Error.
- GOTO function: GOTO[I_i, A] = I_j.

LR-Parser Configurations

- A *configuration* (stack states, remaining input:

  $(s_0 s_1 \ldots s_m, a_i a_{i+1} \ldots a_n \$)$

- Recall that states (except s_0) represent symbols. Hence this pair represents a right sentential form:

  $X_1 X_2 \ldots X_m \, a_i \, a_{i+1} \ldots a_n$

Behavior of the LR Parser

- Given s_m on top of stack, a_i as next input symbol, if ACTION[s_m, a_i] =
  - shift s: ($s_0 s_1 \ldots s_m s$, $a_{i+1} \ldots a_n\$$)
  - reduce $A \rightarrow \beta$: ($s_0 s_1 \ldots s_{m-r} s$, $a_i a_{i+1} \ldots a_n\$$)
    - r=length of β, s=GOTO[s_{m-r}, A]
    - output = semantic action of the production, for now just print it.
  - accept: parsing complete.
  - error: call error recovery routine.

LR-Parser Algorithm

- **INPUT** : An input string *w* and an LR-parsing table with functions ACTION and GOTO for a grammar *G*.
- **OUTPUT** : If *w* is in *L*(*G*), the reduction steps of a bottom-up parse for *w*; otherwise, an error indication.
- **METHOD** : Initially, the parser has s_0 on its stack, where s_0 is the initial state, and *w*$ in the input buffer. The parser then executes the following program.

LR-Parser Algorithm

```
let a be the first symbol of w$;
while (1) { /* repeat forever */
  let s be the state on top of the stack;
  if ( ACTION[s, a] = shift t ) {
    push t onto the stack;
    let a be the next input symbol;
  } else if ( ACTION [s, a] = reduce A → β ) {
    pop |β| symbols off the stack;
    let state t now be on top of the stack;
    push GOTO[t, A] onto the stack;
    output the production A → β;
  } else if ( ACTION[s, a] = accept ) break; /*
  parsing is done */
  else call error-recovery routine;
}
```

Example

GOTO[0, **id**]

shift and stack 5

reduce by prod. 2

- Grammar:
  1) $E \rightarrow E + T$
  2) $E \rightarrow T$
  3) $T \rightarrow T * F$
  4) $T \rightarrow F$
  5) $F \rightarrow (E)$
  6) $F \rightarrow \mathbf{id}$

STATE	ACTION						GOTO		
	id	+	*	(	)	$	*E*	*T*	*F*
0	s5			s4			1	2	3
1		s6				acc			
2		r2	s7		r2	r2			
3		r4	r4		r4	r4			
4	s5			s4			8	2	3
5		r6	r6		r6	r6			
6	s5			s4				9	3
7	s5			s4					10
8		s6			s11				
9		r1	s7		r1	r1			
10		r3	r3		r3	r3			
11		r5	r5		r5	r5			

error

accept

Example

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

	STACK	SYMs	INPUT	ACTION
1	0		**id** * **id** + **id** $	s5
2	0 5	**id**	* **id** + **id** $	r $F \rightarrow$ **id**
3	0 3	F	* **id** + **id** $	r $T \rightarrow F$
4	0 2	T	* **id** + **id** $	s7
5	0 2 7	T *	**id** + **id** $	s5
6	0 2 7 5	T * **id**	+ **id** $	r $F \rightarrow$ **id**
7	0 2 7 10	$T * F$	+ **id** $	r $T \rightarrow T * F$
8	0 2	T	+ **id** $	r $E \rightarrow T$
9	0 1	E	+ **id** $	s6
10	0 1 6	E +	**id** $	s5
11	0 1 6 5	E + **id**	$	r $F \rightarrow$ **id**
12	0 1 6 3	$E + F$	$	r $T \rightarrow F$
13	0 1 6 9	$E + T$	$	r $E \rightarrow E + T$
14	0 1	E	$	acc

S	ACTION						GOTO		
	id	+	*	(	)	$	E	T	F
0	s5			s4			1	2	3
1		s6				acc			
2		r2	s7		r2	r2			
3		r4	r4		r4	r4			
4	s5			s4			8	2	3
5		r6	r6		r6	r6			
6	s5			s4				9	3
7	s5			s4					10
8		s6			s11				
9		r1	s7		r1	r1			
10		r3	r3		r3	r3			
11		r5	r5		r5	r5			

Constructing SLR Parsing Tables

- **INPUT** : An augmented grammar G'.
- **OUTPUT** : The SLR-parsing table functions ACTION and GOTO for G'.
- **METHOD**:

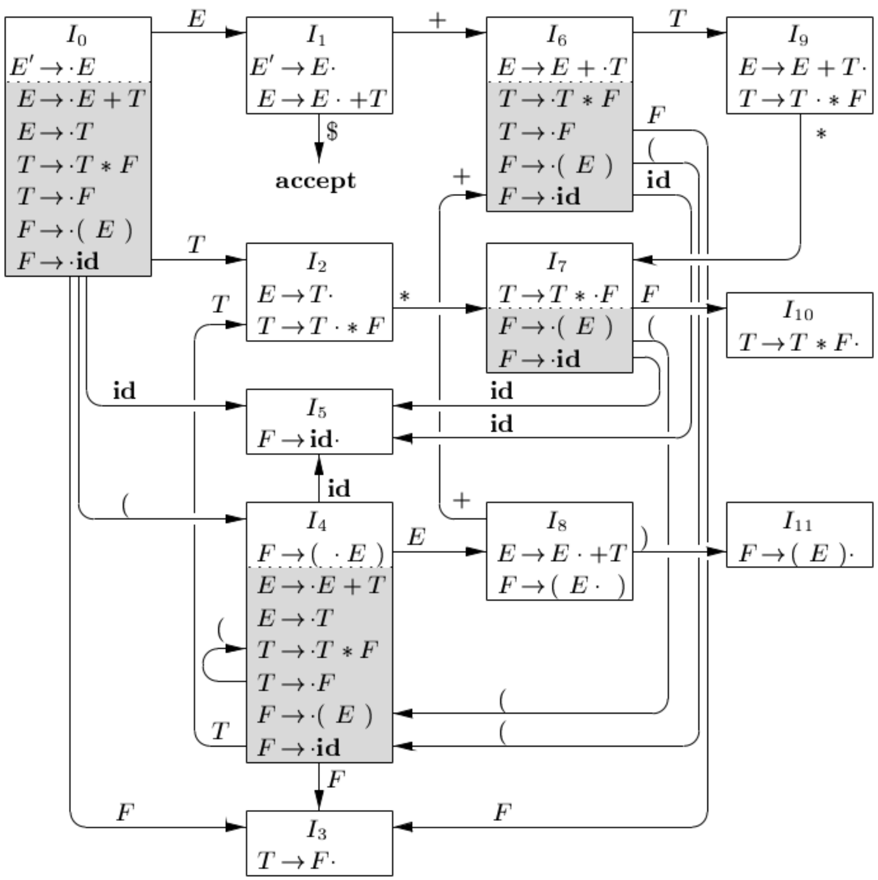
1) Construct $C = \{I_o, I_1, \ldots, I_n\}$, the collection of sets of LR(0) items for G'.
2) State i is constructed from I_i. The parsing actions for state I are determined as follows:
   a) If $[A \rightarrow \alpha \cdot a\beta]$ is in I_i and GOTO(I_i, a) = I_j, then set ACTlON[i, a] to "shift j." Here a must be a terminal.
   b) If $[A \rightarrow \alpha \cdot]$ is in I_i, then set ACTION[i,a] to "reduce $A \rightarrow \alpha$" for all a in FOLLOW(A); here A may not be S'.
   c) If $[S' \rightarrow S \cdot]$ is in I_i, then set ACTION[i, \$] to "accept."

   If any conflicting actions result from the above rules, we say the grammar is not SLR (l) . The algorithm fails to produce a parser in this case.
3) The goto transitions for state i are constructed for all non-terminals A using the rule: If GOTO(I_i, A) = I_j, then GOTO[i, A] = j.
4) All entries not defined by rules (2) and (3) are made "error."
5) The initial state of the parser is the one constructed from the set of items containing $[S' \rightarrow \cdot S]$.

Constructing SLR Parsing Tables

- This algorithm produces *SLR(1) table for G*.
- An LR parser using SLR(1) table for G is *SLR(1) parser for G*.
- A grammar having SLR(1) table is SLR(1).
- Usually (1) is omitted.

Example



STATE	ACTION						GOTO		
	id	+	*	(	)	\$	*E*	*T*	*F*
0	s5			s4			1	2	3
1		s6				acc			
2		r2	s7		r2	r2			
3		r4	r4		r4	r4			
4	s5			s4			8	2	3
5		r6	r6		r6	r6			
6	s5			s4				9	3
7	s5			s4					10
8		s6			s11				
9		r1	s7		r1	r1			
10		r3	r3		r3	r3			
11		r5	r5		r5	r5			

Note

- Every SLR(1) grammar is unambiguous, but not every unambiguous grammar is SLR(1).

Skipped

- From 4.6.5 till the end of chapter.