بسم اللّٰه الرّحمن الرّحیم

دانشگاه صنعتی اصفهان ــ دانشکدهٔ مهندسی برق و کامپیوتر
(نیم‌سال تحصیلی ۴۰۲۲)

کامپایلر

حسین فلسفین

Example 4.45: Figure 4.37 shows the ACTION and GOTO functions of an LR-parsing table for the expression grammar (4.1), repeated here with the productions numbered:

$$\begin{array}{llll} (1) & E \to E + T & (4) & T \to F \\ (2) & E \to T & (5) & F \to (E) \\ (3) & T \to T * F & (6) & F \to \mathbf{id} \end{array}$$

The codes for the actions are:

1. si means shift and stack state i,
2. rj means reduce by the production numbered j,
3. acc means accept,
4. blank means error.

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			

Figure 4.37: Parsing table for expression grammar

	STACK	SYMBOLS	INPUT	ACTION
(1)	0		$\mathbf{id} * \mathbf{id} + \mathbf{id}\,\$$	shift
(2)	0 5	$\mathbf{id}$	$* \mathbf{id} + \mathbf{id}\,\$$	reduce by $F \to \mathbf{id}$
(3)	0 3	F	$* \mathbf{id} + \mathbf{id}\,\$$	reduce by $T \to F$
(4)	0 2	T	$* \mathbf{id} + \mathbf{id}\,\$$	shift
(5)	0 2 7	$T *$	$\mathbf{id} + \mathbf{id}\,\$$	shift
(6)	0 2 7 5	$T * \mathbf{id}$	$+ \mathbf{id}\,\$$	reduce by $F \to \mathbf{id}$
(7)	0 2 7 10	$T * F$	$+ \mathbf{id}\,\$$	reduce by $T \to T * F$
(8)	0 2	T	$+ \mathbf{id}\,\$$	reduce by $E \to T$
(9)	0 1	E	$+ \mathbf{id}\,\$$	shift
(10)	0 1 6	$E +$	$\mathbf{id}\,\$$	shift
(11)	0 1 6 5	$E + \mathbf{id}$	$\$$	reduce by $F \to \mathbf{id}$
(12)	0 1 6 3	$E + F$	$\$$	reduce by $T \to F$
(13)	0 1 6 9	$E + T$	$\$$	reduce by $E \to E + T$
(14)	0 1	E	$\$$	accept

Figure 4.38: Moves of an LR parser on $\mathbf{id} * \mathbf{id} + \mathbf{id}$

4.6.4 Constructing SLR-Parsing Tables

The ACTION and GOTO entries in the parsing table are constructed using the following algorithm. It requires us to know FOLLOW(A) for each nonterminal A of a grammar

Algorithm 4.46 : Constructing an SLR-parsing table.

INPUT: An augmented grammar G'.

OUTPUT: The SLR-parsing table functions ACTION and GOTO for G'.

1. Construct $C = \{I_0, I_1, \ldots, I_n\}$, 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 $[A \to \alpha{\cdot}a\beta]$ is in I_i and $\text{GOTO}(I_i, a) = I_j$, then set $\text{ACTION}[i, a]$ to "shift j." Here a must be a terminal.

   (b) If $[A \to \alpha\cdot]$ is in I_i, then set $\text{ACTION}[i, a]$ to "reduce $A \to \alpha$" for all a in $\text{FOLLOW}(A)$; here A may not be S'.

   (c) If $[S' \to S\cdot]$ is in I_i, then set $\text{ACTION}[i, \$]$ to "accept."

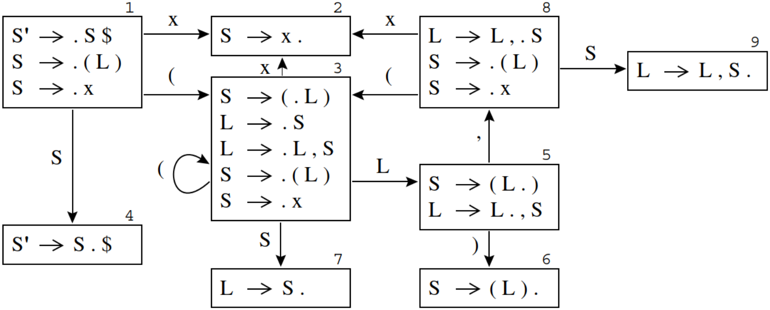
If any conflicting actions result from the above rules, we say the grammar is not SLR(1). The algorithm fails to produce a parser in this case.

3. The goto transitions for state i are constructed for all nonterminals A using the rule: If $\text{GOTO}(I_i, A) = I_j$, then $\text{GOTO}[i, A] = j$.

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 $[S' \to \cdot S]$.

یک مثال از کتاب اپل: یک پارسر $LR(0)$ است و نه یک پارسر $SLR(1)$



	(	)	x	,	$	S	L
1	s3		s2			g4	
2	r2	r2	r2	r2	r2		
3	s3		s2			g7	g5
4					a		
5		s6		s8			
6	r1	r1	r1	r1	r1		
7	r3	r3	r3	r3	r3		
8	s3		s2			g9	
9	r4	r4	r4	r4	r4		

0 $S' \to S\$$

1 $S \to (\,L\,)$ 3 $L \to S$

2 $S \to x$ 4 $L \to L\,,\ S$

تمایز یک پارسر LR(0) با یک پارسر SLR(1)

A simple way of constructing better-than-LR(0) parsers is called SLR, which stands for simple LR. Parser construction for SLR is almost identical to that for LR(0), except that we put reduce actions into the table only where indicated by the FOLLOW set.

(5.4) For each complete item $A \to \alpha_1 \ldots \alpha_n \centerdot$ for some $n \geq 0$, where the α_i's are symbols in $V_T \cup V_N$, in the label of a state q_i,
we insert in each entry of row q_i in the *action* part of the table T, the action *red* p (short for *reduce production* p), where p is the number identifying the production $A \to \alpha_1 \ldots \alpha_n$ of the grammar G'. This means that the parsing automaton should replace the string $\alpha_1 q_{\alpha_1} \ldots \alpha_n q_{\alpha_n}$ which is on the stack (the top element being state q_{α_n}) by the nonterminal A. This action of the parsing automaton is called a *reduce action*.

(5.4*) for every state q of the deterministic finite automaton M,

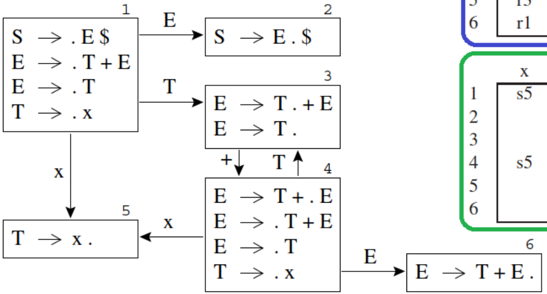
if there exists in q a complete item $A \to \alpha \centerdot$, for some $A \in V_N$ and $\alpha \in V^*$,
then for every terminal symbol a of the set $Follow_1(A)$, we insert in the entry $T[q, a]$ of the parsing table the *reduce* action *red* p, where p identifies the production $A \to \alpha$ of the augmented grammar G'.

Let's revisit the assumption that if the item is complete, the parser must choose to reduce. Is that always appropriate? If the sequence on top of the stack could be reduced to the non-terminal A, what tokens do we expect to find as the next input? What tokens would tell us that the reduction is not appropriate? Perhaps $\mathrm{FOLLOW}(A)$ could be useful here! The simple improvement that $\mathrm{SLR}(1)$ makes on the basic $\mathrm{LR}(0)$ parser is to reduce only if the next input token is a member of the follow set of the non-terminal being reduced. When filling in the table, we don't assume a reduce on all inputs as we did in $\mathrm{LR}(0)$, we selectively choose the reduction only when the next input symbols in a member of the follow set.

مثالی از یک گرامر که LR(0) نیست اما SLR(1) هست (از کتاب اپل)

0	$S \rightarrow E\ \$$	2	$E \rightarrow T$
1	$E \rightarrow T + E$	3	$T \rightarrow x$

GRAMMAR 3.23.



LR(0) Parsing

	x	+	$	E	T
1	s5			g2	g3
2			a		
3	r2	s4,r2	r2		
4	s5			g6	g3
5	r3	r3	r3		
6	r1	r1	r1		

	x	+	$	E	T
1	s5			g2	g3
2			a		
3		s4	r2		
4	s5			g6	g3
5		r3	r3		
6			r1		

SLR(1) Parsing

تمایز یک گرامر LR(0) با یک گرامر SLR(1): گرامر ۳.۲۳

In state 3, on symbol $+$, there is a duplicate entry: The parser must shift into state 4 and also reduce by production 2. This is a conflict and indicates that the grammar is not $\mathrm{LR}(0)$ – it cannot be parsed by an $\mathrm{LR}(0)$ parser. We will need a more powerful parsing algorithm.

We can put fewer reduce actions into the SLR table. The SLR class of grammars is precisely those grammars whose SLR parsing table contains no conflicts (duplicate entries). Grammar 3.23 belongs to this class, as do many useful programming-language grammars.

مثال از یک گرامر نامناسب برای تجزیهٔ SLR (مثال ۴.۴۸ از کتاب آهو)

$$
\begin{array}{llll}
I_0: & S' \to \cdot S & I_5: & L \to \mathbf{id}\cdot \\
 & S \to \cdot L = R & & \\
 & S \to \cdot R & I_6: & S \to L = \cdot R \\
 & L \to \cdot * R & & R \to \cdot L \\
 & L \to \cdot \mathbf{id} & & L \to \cdot * R \\
 & R \to \cdot L & & L \to \cdot \mathbf{id} \\
I_1: & S' \to S\cdot & I_7: & L \to *R\cdot \\
I_2: & S \to L\cdot = R & I_8: & R \to L\cdot \\
 & R \to L\cdot & & \\
 & & I_9: & S \to L = R\cdot \\
I_3: & S \to R\cdot & & \\
I_4: & L \to *\cdot R & & \\
 & R \to \cdot L & & \\
 & L \to \cdot * R & & \\
 & L \to \cdot \mathbf{id} & &
\end{array}
$$

$$
\boxed{
\begin{array}{lll}
S & \to & L = R \mid R \\
L & \to & *R \mid \mathbf{id} \\
R & \to & L
\end{array}
}
$$

Consider the set of items I_2. The first item in this set makes $\mathrm{ACTION}[2,=]$ be "shift 6." Since $\mathrm{FOLLOW}(R)$ contains $=$ (to see why, consider the derivation $S \Rightarrow L = R \Rightarrow *R = R$), the second item sets $\mathrm{ACTION}[2,=]$ to "reduce $R \rightarrow L$." Since there is both a shift and a reduce entry in $\mathrm{ACTION}[2,=]$, state 2 has a shift/reduce conflict on input symbol $=$.

در واقع یک درایه از جدول دارای بیش از یک عضو خواهد بود.