Kompilatorteknikk TDT4205 - Problem set 3

Ola Kristoffer Hoff

21st February, 2023

1 Bottom/up parsing tables

Consider the following grammar:

$$\begin{aligned} E &\rightarrow E + T | T \\ T &\rightarrow T * F | F \\ F &\rightarrow x \end{aligned}$$

1.1 Augment the grammar, and construct its LR(0) automaton.

As I understand it augmenting the grammar is simply adding a "start"-state, $S' \rightarrow S$, that maps to the first actual element, here E. This is done so we can neatly see in our automaton where the program starts and ends: "$S \rightarrow .E$" and "$S \rightarrow E.$", respectively. The grammar is then denoted as:

$$\begin{aligned} E' &\rightarrow E \\ E &\rightarrow E + T | T \\ T &\rightarrow T * F | F \\ F &\rightarrow x \end{aligned}$$

Now that we have augmented our grammar we can construct the LR(0) automaton. We can see the automaton i figure 1.

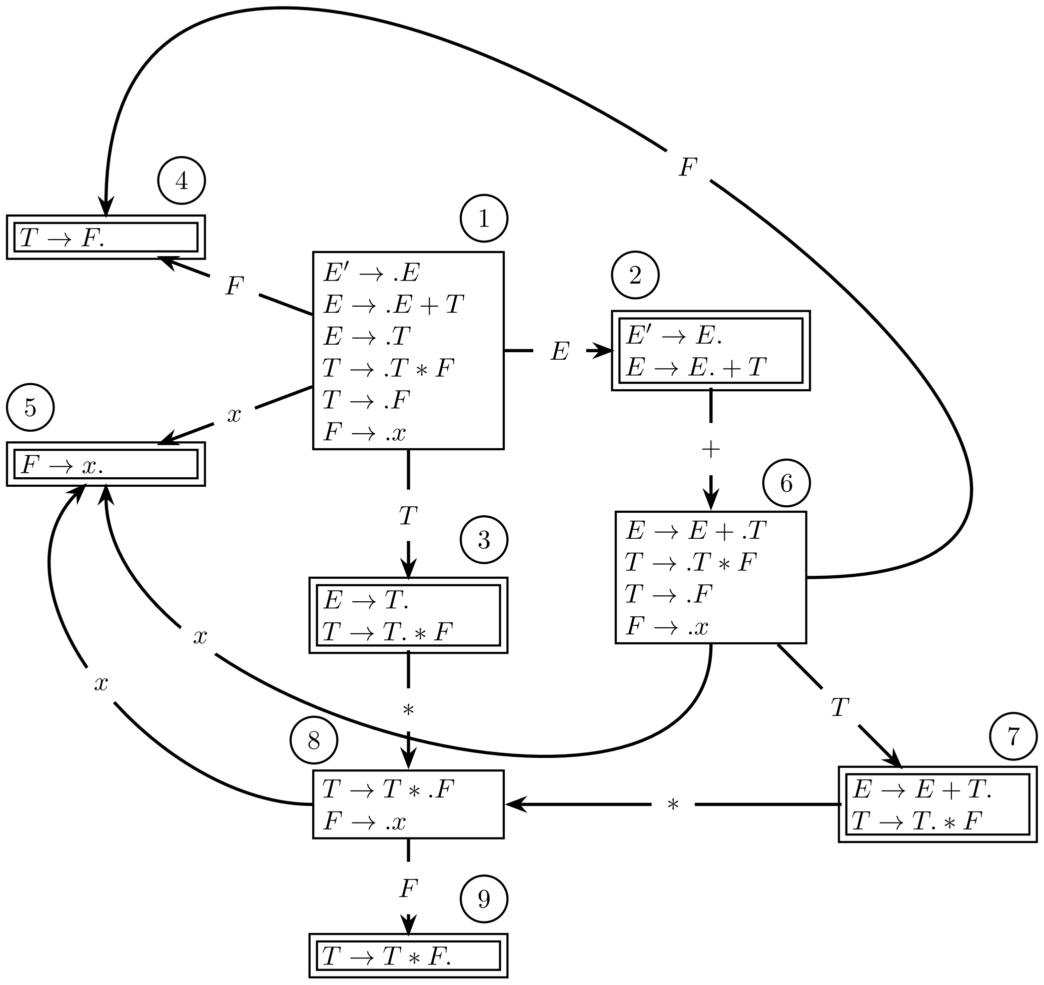

Figure 1: LR(0) automaton for the grammar.

1.2 Identify any states with shift/reduce conflicts in your automaton.

We have three states with shift/reduce conflicts. These are the accepting states with edges going out of them. These are: 2, 3 and 7, and can been seen in figure 1.

1.3 Is the grammar SLR parseable? Justify your answer by constructing the SLR parsing table.

Yes, the grammar is SLR parseable. This is beacuse with SLR we use one token of look ahead to solve the shift/reduce conflicts that we spotted in the previous subsection. I've added the grammar folded out and numbered below:

(1) $E' \to E$
(2) $E \to E + T$
(3) $E \to T$
(4) $T \to T * F$
(5) $T \to F$
(6) $F \to x$

	x	+	*	$	E	T	F
1	s5				g2	g3	g4
2		s6		a			
3			s8	r3			
4				r5			
5				r6			
6	s5					g7	g4
7			s8	r2			
8	s5						g9
9				r4			

Table 1: SLR parsing table for the given grammar.

As we can see in table 1 there are no double entries which we would get if we created the LR(0) parsing table.

2 Implementation — Tree simplification

I don't have much to comment on here. See the implementation in code.