Compilers: Bottom-Up Parsing

a topic in

DM565 – Formal Languages and Data Processing

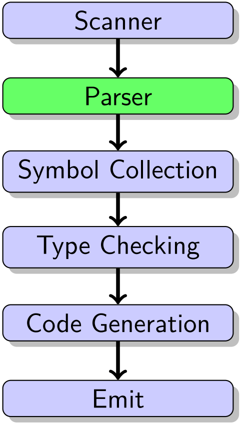
Kim Skak Larsen

Department of Mathematics and Computer Science (IMADA)
University of Southern Denmark (SDU)

kslarsen@imada.sdu.dk

October, 2023

Syntax Analysis: parsers



The Parsing Problem

We have a grammar G (partially) defining the programming language and a string s in the form of the user's program.

We want to know if the user program is correct, i.e., if $s \in L(G)$.

We have seen a *top-down* (predictive) parsing technique.

Now we consider a *bottom-up* parsing technique, focusing on the LR(1) techniques and the derived LALR(1).

Syntax Analysis: parsers

Input to phase

A stream of tokens (keywords, numbers, identifiers, symbols)

Output from phase

An abstract syntax tree (AST)

Organization of Presentation

1. The overall behavior we want.
2. How to use the parser we will make.
3. How to construct the parse table.

Crafting a Parser

Desired Functionality

Consider the following grammar:

1 $S \rightarrow S \; ; \; S$	4 $E \rightarrow \mathrm{id}$	8 $L \rightarrow E$
2 $S \rightarrow \mathrm{id} := E$	5 $E \rightarrow \mathrm{num}$	9 $L \rightarrow L \, , \, E$
3 $S \rightarrow \mathrm{print} \; (\; L \;)$	6 $E \rightarrow E + E$	
	7 $E \rightarrow (S \, , \, E)$	

GRAMMAR 3.1. A syntax for straight-line programs.

We will work towards getting the following parsing functionality:

Crafting a Parser

Stack	*Input*	*Action*
$_1$	a := 7 ; b := c + (d := 5 + 6 , d) $	*shift*
$_1$ id$_4$	:= 7 ; b := c + (d := 5 + 6 , d) $	*shift*
$_1$ id$_4$:=$_6$	7 ; b := c + (d := 5 + 6 , d) $	*shift*
$_1$ id$_4$:=$_6$ num$_{10}$	; b := c + (d := 5 + 6 , d) $	*reduce* $E \to$ num
$_1$ id$_4$:=$_6$ E_{11}	; b := c + (d := 5 + 6 , d) $	*reduce* $S \to$ id:=E
$_1$ S_2	; b := c + (d := 5 + 6 , d) $	*shift*
$_1$ S_2 ;$_3$	b := c + (d := 5 + 6 , d) $	*shift*
$_1$ S_2 ;$_3$ id$_4$	:= c + (d := 5 + 6 , d) $	*shift*
$_1$ S_2 ;$_3$ id$_4$:=$_6$	c + (d := 5 + 6 , d) $	*shift*
$_1$ S_2 ;$_3$ id$_4$:=$_6$ id$_{20}$	+ (d := 5 + 6 , d) $	*reduce* $E \to$ id
$_1$ S_2 ;$_3$ id$_4$:=$_6$ E_{11}	+ (d := 5 + 6 , d) $	*shift*
$_1$ S_2 ;$_3$ id$_4$:=$_6$ E_{11} +$_{16}$	(d := 5 + 6 , d) $	*shift*
$_1$ S_2 ;$_3$ id$_4$:=$_6$ E_{11} +$_{16}$ ($_8$	d := 5 + 6 , d) $	*shift*
$_1$ S_2 ;$_3$ id$_4$:=$_6$ E_{11} +$_{16}$ ($_8$ id$_4$	:= 5 + 6 , d) $	*shift*
$_1$ S_2 ;$_3$ id$_4$:=$_6$ E_{11} +$_{16}$ ($_8$ id$_4$:=$_6$	5 + 6 , d) $	*shift*
$_1$ S_2 ;$_3$ id$_4$:=$_6$ E_{11} +$_{16}$ ($_8$ id$_4$:=$_6$ num$_{10}$	+ 6 , d) $	*reduce* $E \to$ num
$_1$ S_2 ;$_3$ id$_4$:=$_6$ E_{11} +$_{16}$ ($_8$ id$_4$:=$_6$ E_{11}	+ 6 , d) $	*shift*
$_1$ S_2 ;$_3$ id$_4$:=$_6$ E_{11} +$_{16}$ ($_8$ id$_4$:=$_6$ E_{11} +$_{16}$	6 , d) $	*shift*
$_1$ S_2 ;$_3$ id$_4$:=$_6$ E_{11} +$_{16}$ ($_8$ id$_4$:=$_6$ E_{11} +$_{16}$ num$_{10}$	, d) $	*reduce* $E \to$ num
$_1$ S_2 ;$_3$ id$_4$:=$_6$ E_{11} +$_{16}$ ($_8$ id$_4$:=$_6$ E_{11} +$_{16}$ E_{17}	, d) $	*reduce* $E \to E + E$
$_1$ S_2 ;$_3$ id$_4$:=$_6$ E_{11} +$_{16}$ ($_8$ id$_4$:=$_6$ E_{11}	, d) $	*reduce* $S \to$ id:=E
$_1$ S_2 ;$_3$ id$_4$:=$_6$ E_{11} +$_{16}$ ($_8$ S_{12}	, d) $	*shift*
$_1$ S_2 ;$_3$ id$_4$:=$_6$ E_{11} +$_{16}$ ($_8$ S_{12} ,$_{18}$	d) $	*shift*
$_1$ S_2 ;$_3$ id$_4$:=$_6$ E_{11} +$_{16}$ ($_8$ S_{12} ,$_{18}$ id$_{20}$	) $	*reduce* $E \to$ id
$_1$ S_2 ;$_3$ id$_4$:=$_6$ E_{11} +$_{16}$ ($_8$ S_{12} ,$_{18}$ E_{21}	) $	*shift*
$_1$ S_2 ;$_3$ id$_4$:=$_6$ E_{11} +$_{16}$ ($_8$ S_{12} ,$_{18}$ E_{21})$_{22}$	$	*reduce* $E \to (S, E)$
$_1$ S_2 ;$_3$ id$_4$:=$_6$ E_{11} +$_{16}$ E_{17}	$	*reduce* $E \to E + E$
$_1$ S_2 ;$_3$ id$_4$:=$_6$ E_{11}	$	*reduce* $S \to$ id:=E
$_1$ S_2 ;$_3$ S_5	$	*reduce* $S \to S; S$
$_1$ S_2	$	*accept*

FIGURE 3.18. Shift-reduce parse of a sentence. Numeric subscripts in the *Stack* are DFA state numbers; see Table 3.19.

SDU

Stack	Input	Action
$_1$	`a := 7 ; b := c + ( d := 5 + 6 , d ) $`	*shift*
$_1$ id$_4$	`:= 7 ; b := c + ( d := 5 + 6 , d ) $`	*shift*
$_1$ id$_4$:=$_6$	`7 ; b := c + ( d := 5 + 6 , d ) $`	*shift*
$_1$ id$_4$:=$_6$ num$_{10}$	`; b := c + ( d := 5 + 6 , d ) $`	*reduce* $E \rightarrow$ num
$_1$ id$_4$:=$_6$ E_{11}	`; b := c + ( d := 5 + 6 , d ) $`	*reduce* $S \rightarrow$ id:=E
$_1$ S_2	`; b := c + ( d := 5 + 6 , d ) $`	*shift*
$_1$ S_2 ;$_3$	`b := c + ( d := 5 + 6 , d ) $`	*shift*
$_1$ S_2 ;$_3$ id$_4$	`:= c + ( d := 5 + 6 , d ) $`	*shift*
$_1$ S_2 ;$_3$ id$_4$:=$_6$	`c + ( d := 5 + 6 , d ) $`	*shift*
$_1$ S_2 ;$_3$ id$_4$:=$_6$ id$_{20}$	`+ ( d := 5 + 6 , d ) $`	*reduce* $E \rightarrow$ id
$_1$ S_2 ;$_3$ id$_4$:=$_6$ E_{11}	`+ ( d := 5 + 6 , d ) $`	*shift*
$_1$ S_2 ;$_3$ id$_4$:=$_6$ E_{11} +$_{16}$	`( d := 5 + 6 , d ) $`	*shift*
$_1$ S_2 ;$_3$ id$_4$:=$_6$ E_{11} +$_{16}$ ($_8$	`d := 5 + 6 , d ) $`	*shift*
$_1$ S_2 ;$_3$ id$_4$:=$_6$ E_{11} +$_{16}$ ($_8$ id$_4$	`:= 5 + 6 , d ) $`	*shift*
$_1$ S_2 ;$_3$ id$_4$:=$_6$ E_{11} +$_{16}$ ($_8$ id$_4$:=$_6$	`5 + 6 , d ) $`	*shift*

Crafting a Parser

$_1\ S_2\ ;_3\ \mathrm{id}_4\ :=_6\ E_{11}\ +_{16}\ (_8\ \mathrm{id}_4\ :=_6$	`5 + 6 , d ) $`	*shift*
$_1\ S_2\ ;_3\ \mathrm{id}_4\ :=_6\ E_{11}\ +_{16}\ (_8\ \mathrm{id}_4\ :=_6\ \mathrm{num}_{10}$	`+ 6 , d ) $`	*reduce* $E \to \mathrm{num}$
$_1\ S_2\ ;_3\ \mathrm{id}_4\ :=_6\ E_{11}\ +_{16}\ (_8\ \mathrm{id}_4\ :=_6\ E_{11}$	`+ 6 , d ) $`	*shift*
$_1\ S_2\ ;_3\ \mathrm{id}_4\ :=_6\ E_{11}\ +_{16}\ (_8\ \mathrm{id}_4\ :=_6\ E_{11}\ +_{16}$	`6 , d ) $`	*shift*
$_1\ S_2\ ;_3\ \mathrm{id}_4\ :=_6\ E_{11}\ +_{16}\ (_8\ \mathrm{id}_4\ :=_6\ E_{11}\ +_{16}\ \mathrm{num}_{10}$	`, d ) $`	*reduce* $E \to \mathrm{num}$
$_1\ S_2\ ;_3\ \mathrm{id}_4\ :=_6\ E_{11}\ +_{16}\ (_8\ \mathrm{id}_4\ :=_6\ E_{11}\ +_{16}\ E_{17}$	`, d ) $`	*reduce* $E \to E + E$
$_1\ S_2\ ;_3\ \mathrm{id}_4\ :=_6\ E_{11}\ +_{16}\ (_8\ \mathrm{id}_4\ :=_6\ E_{11}$	`, d ) $`	*reduce* $S \to \mathrm{id} := E$
$_1\ S_2\ ;_3\ \mathrm{id}_4\ :=_6\ E_{11}\ +_{16}\ (_8\ S_{12}$	`, d ) $`	*shift*
$_1\ S_2\ ;_3\ \mathrm{id}_4\ :=_6\ E_{11}\ +_{16}\ (_8\ S_{12}\ ,_{18}$	`d ) $`	*shift*
$_1\ S_2\ ;_3\ \mathrm{id}_4\ :=_6\ E_{11}\ +_{16}\ (_8\ S_{12}\ ,_{18}\ \mathrm{id}_{20}$	`) $`	*reduce* $E \to \mathrm{id}$
$_1\ S_2\ ;_3\ \mathrm{id}_4\ :=_6\ E_{11}\ +_{16}\ (_8\ S_{12}\ ,_{18}\ E_{21}$	`) $`	*shift*
$_1\ S_2\ ;_3\ \mathrm{id}_4\ :=_6\ E_{11}\ +_{16}\ (_8\ S_{12}\ ,_{18}\ E_{21}\ )_{22}$	`$`	*reduce* $E \to (S, E)$
$_1\ S_2\ ;_3\ \mathrm{id}_4\ :=_6\ E_{11}\ +_{16}\ E_{17}$	`$`	*reduce* $E \to E + E$
$_1\ S_2\ ;_3\ \mathrm{id}_4\ :=_6\ E_{11}$	`$`	*reduce* $S \to \mathrm{id} := E$
$_1\ S_2\ ;_3\ S_5$	`$`	*reduce* $S \to S; S$
$_1\ S_2$	`$`	*accept*

Crafting a Parser

If we succeed, we have found a derivation, i.e., $s \in L(G)$.

The parser tree is developed bottom-up, ending with the root (the start symbol of the grammar).

Notice that before we understand the method, it seems some “guessing” is involved in choosing when to reduce.

We will construct a parser table (not the same kind as for top-down) where no guessing is involved.

Example

Parse the string (user program)

```
x := 40+2; print(x)$
```

using the following grammar and parsing table:

Crafting a Parser

1 $S \to S \; ; \; S$	4 $E \to \text{id}$	8 $L \to E$
2 $S \to \text{id} := E$	5 $E \to \text{num}$	9 $L \to L \; , \; E$
3 $S \to \text{print} \; (\; L \;)$	6 $E \to E + E$	
	7 $E \to (S \; , \; E \;)$	

GRAMMAR 3.1. A syntax for straight-line programs.

Typos on next slide from the book:

- In State 9, add "s20" under 'id', "s10" under 'num', "s8" under '('.
- In State 15, add "s16" under '+'.

Crafting a Parser

	id	num	print	;	,	+	:=	(	)	$	*S*	*E*	*L*
1	s4		s7								g2		
2				s3						a			
3	s4		s7								g5		
4							s6						
5				r1	r1					r1			
6	s20	s10						s8				g11	
7								s9					
8	s4		s7								g12		
9												g15	g14
10				r5	r5	r5			r5	r5			
11				r2	r2	s16				r2			
12				s3	s18								
13				r3	r3					r3			
14					s19				s13				
15					r8				r8				
16	s20	s10						s8				g17	
17				r6	r6	s16			r6	r6			
18	s20	s10						s8				g21	
19	s20	s10						s8				g23	
20				r4	r4	r4			r4	r4			
21									s22				
22				r7	r7	r7			r7	r7			
23					r9	s16			r9				

TABLE 3.19. LR parsing table for Grammar 3.1.

Crafting a Parser

LR(1) Parser Table Construction

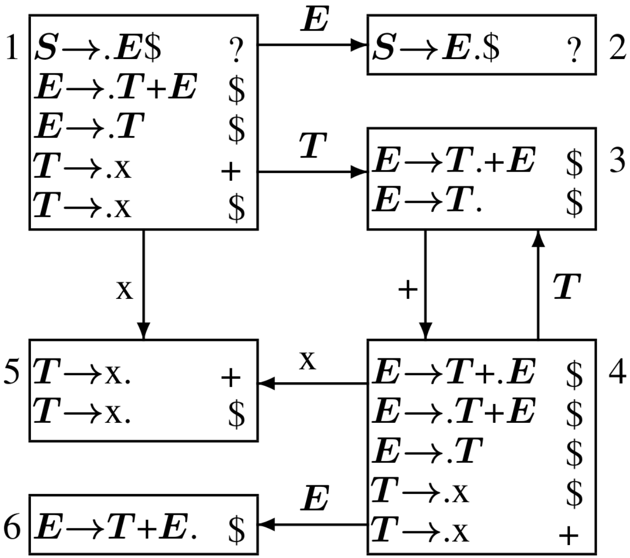
Using the grammar:

$$_0\; S \rightarrow E\$ \qquad\qquad _2\; E \rightarrow T$$

$$_1\; E \rightarrow T + E \qquad\qquad _3\; T \rightarrow \mathrm{x}$$

we construct the following DFA and then a parser table:

Crafting a Parser



Crafting a Parser

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

The Algorithmic Components

Closure(I) =
  repeat
    for any item $(A \to \alpha.X\beta, z)$ in I
      for any production $X \to \gamma$
        for any $w \in \text{FIRST}(\beta z)$
          $I \leftarrow I \cup \{(X \to .\gamma,\ w)\}$
  until I does not change
  return I

Goto(I, X) =
  $J \leftarrow \{\}$
  for any item $(A \to \alpha.X\beta,\ z)$ in I
    add $(A \to \alpha X.\beta,\ z)$ to J
  return Closure(J).

SDU

LALR(1) Parser Table Construction

LR(1) parser tables are space consuming.

An LALR(1) DFA is defined to be the LR(1) DFA where states, identical except for lookahead, are merged (recursively), giving rise to smaller parser tables:

Typos on next slide from the book:

- The arrow from State 1 to State 2 should be marked 'S'.
- The arrow from State 1 to State 6 should be marked '$*$'.
- The 'T' in State 6 should be a 'V'.

Crafting a Parser

SDU

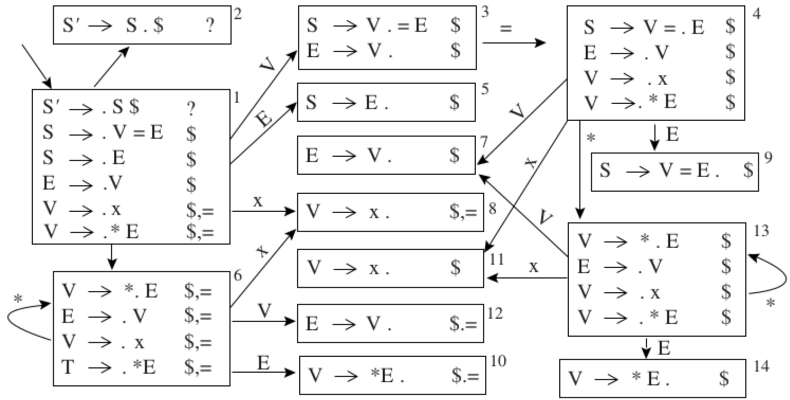


FIGURE 3.27. LR(1) states for Grammar 3.26.

Crafting a Parser

	x	*	=	$	*S*	*E*	*V*
1	s8	s6			g2	g5	g3
2				a			
3			s4	r3			
4	s11	s13				g9	g7
5				r2			
6	s8	s6				g10	g12
7				r3			
8			r4	r4			
9				r1			
10			r5	r5			
11				r4			
12			r3	r3			
13	s11	s13				g14	g7
14				r5			

(a) LR(1)

	x	*	=	$	*S*	*E*	*V*
1	s8	s6			g2	g5	g3
2				a			
3			s4	r3			
4	s8	s6				g9	g7
5				r2			
6	s8	s6				g10	g7
7			r3	r3			
8			r4	r4			
9				r1			
10			r5	r5			

(b) LALR(1)

TABLE 3.28. LR(1) and LALR(1) parsing tables for Grammar 3.26.

SDU

Consequences of Using LALR(1)

- The same strings are accepted (because we find a derivation).
- Error messages may be delayed (because we may continue for a while in situations where LR(1) would have terminated).