8.1. LL(1) Parsers

A top-down parser that uses a one-token lookahead is called an LL(1) parser.

- The first L indicates that the input is read from left to right.
- The second L says that it produces a left-to-right derivation.
- And the 1 says that it uses one lookahead token. (Some parsers look ahead at the next 2 tokens, or even more than that.)

The LL(1) parsing table

The parser needs to find a production to use for nonterminal N when it sees lookahead token t.

To select which production to use, it suffices to have a table that has, as a key, a pair (N, t) and gives the number of a production to use.

Let's illustrate with an LL(1) parsing table for the expression grammar that we used earlier, which looks like this.

1. $E \rightarrow T\ R$
2. $R \rightarrow \varepsilon$
3. $R \rightarrow +\ E$
4. $T \rightarrow F\ S$
5. $S \rightarrow \varepsilon$
6. $S \rightarrow *\ T$
7. $F \rightarrow$ **n**
8. $F \rightarrow (\ E\ )$

Parsing table D below does the job. Each row is labeled by a nonterminal and each column by a lookahead token, or the special symbol $ that indicates the end of the input.

D(N, t) is the production to use to expand N when the lookahead is t. Blank entries mean syntax error.

Table D

	n	+	*	**(**	**)**	**$**
E	1			1		
R		3	2		2	2
T	4			4		

S		5	6		5	5
F	7			8		

Now it is easy to use the table to control a top-down parse. Parsing **n * n** goes as follows. Start with *E*.

```
    E
```

D(*E*, **n**) = 1, so expand *E* using production 1.

```
     E
    / \
   T   R
```

Since D(*T*, **n**) = 4, we continue by expanding *T* to *F S*.

```
     E
    / \
   T   R
  / \
 F   S
```

Now D(*F*, **n**) = 7, and production 7 is *F* → **n**.

```
     E
    / \
   T   R
  / \
 F   S
 |
 n
```

The lookahead changes to * and D(S, *) = 6. Since production 5 is S → * *T*, the table tells us to replace *S* by * *T*.

```
     E
    / \
   T   R
  / \
 F   S
 |  / \
 n *   T
```

Now the lookahead is **n**, and D(*T*, **n**) = 4. After using production 4 (*T* → *F S*), the table will tell us to use production 7 (*F* → **n**), giving

```
     E
    / \
   T   R
  / \
 F   S
 |  / \
 n *   T
      / \
     F   S
```

```
          |
          n
```

The parse is almost finished. Since there are no more tokens, the lookahead is \$. D(S, \$) = 5 and D(R, \$) = 2, which says to replace S and R by ε.

```
           E
          / \
         /   \
       T       R
      / \      |
     F   S     ε
     |  / \
     n *   T
          / \
         F   S
         |   |
         n   ε
```

A stack-based approach

Instead of building a parse tree, it can be preferable to construct a derivation.

We use a two stacks, called *Match* and *Todo*. Stack Matched only holds tokens.

At any given point, the string that has been derived is *m t* where *m* is the contents of the Matched stack (from bottom to top) and *t* is the contents of the Todo stack (from top to bottom).

The *action* tells the production that is used, or, when a token is moved to the Matched stack and removed from the input, a *match* action.

Here is a parse of **n + n * n** using the same expression grammar.

Matched		**Todo**		**Input**		**Action**
		E **\$**		**n** + **n** * **n** **\$**		
		$T\ R$ **\$**		**n** + **n** * **n** **\$**		$E \rightarrow T\ R$
		$F\ S\ R$ **\$**		**n** + **n** * **n** **\$**		$T \rightarrow F\ S$
		n $S\ R$ **\$**		**n** + **n** * **n** **\$**		$F \rightarrow$ **n**
n		$S\ R$ **\$**		+ **n** * **n** **\$**		match **n**
n		R **\$**		+ **n** * **n** **\$**		$S \rightarrow \varepsilon$
n		+ E **\$**		+ **n** * **n** **\$**		$R \rightarrow + E$
n +		E **\$**		**n** * **n** **\$**		match +
n +		$T\ R$ **\$**		**n** * **n** **\$**		$E \rightarrow T\ R$
n +		$F\ S\ R$ **\$**		**n** * **n** **\$**		$T \rightarrow F\ S$

n +		**n** S R **$**		**n** * **n** **$**		$F \to$ **n**
n + **n**		S R **$**		* **n** **$**		match **n**
n + **n**		* T R **$**		* **n** **$**		$S \to$ * T
n + **n** *		T R **$**		**n** **$**		match *
n + **n** *		F S R **$**		**n** **$**		$T \to F$ S
n + **n** *		**n** S R **$**		**n** **$**		$F \to$ **n**
n + **n** * **n**		S R **$**		**$**		match **n**
n + **n** * **n**		R **$**		**$**		$S \to \varepsilon$
n + **n** * **n**		**$**		**$**		$R \to \varepsilon$

← • HOME • UP • →

Top/Top-down parsing/LL(1)