Bottom-Up Parsing

We can use bottom-up parsing to find the derivation of a string provided the CFG has the right form.

Parsing

Parsing means getting from input string to derivation tree, and rejecting those strings not generated by the grammar.

There are two competing approaches. Easiest is ***top-down*** parsing: look at the first symbol of the input to determine which production was used first. Only works for some CFGs.

Bottom-up Parsing

In ***bottom-up*** parsing, we read symbols until have a group that matches RHS of a production. Then replace the group with LHS of production. And repeat. (This produces a rightmost derivation.)

Shifts & Reductions

Bottom-up parsing consists of steps:

- ***Shift*** the next input terminal onto the stack; or
- ***Reduce:*** replace stacked symbols that are RHS of production by LHS variable.

The main question is: should one reduce now, or read the next symbol?

Example CFG

The grammar for arithmetic expressions, slightly modified:

$$E \rightarrow E\texttt{+}T \mid T$$
$$T \rightarrow T\times F \mid F$$
$$F \rightarrow \texttt{(}\,E\,\texttt{)} \mid \texttt{n}$$

The parsing of $\texttt{(n+n)}\times\texttt{n}$ is given next. Note that this *assumes* we know what to do at each step. . .

Type	*Stack*	*To be read*	*Comment*
		(n+n)*n	
s	(	n+n)*n	
s	(n	+n)*n	
R	(F	+n)*n	
R	(T	+n)*n	
R	(E	+n)*n	
s	(E+	n)*n	
s	(E+n	)*n	
R	(E+F	)*n	
R	(E+T	)*n	
R	(E	)*n	not reduction T to E
s	(E)	*n	
R	F	*n	
R	T	*n	
s	T*	n	not reduction T to E
s	T*n		
R	T*F		
R	T		not reduction F to T
R	E		

Table-Driven Parser for LR(1) Grammars

We consider special type of CFG called a LR(1) grammar. This has a table that tells one what the next operation should be.

The parser has a current state and a stack. Entries on the stack alternate between states and symbols (terminal or variable).

Four Actions

For each state and possible input symbol, the table specifies one of four possible actions:

- *Shift.*
- *Reduce by the specified production.*
- *Error.* (Indicated by blank entry.)
- *Halt.* And accept.

Details of Shifting

(1) Push current state onto stack.

(2) Read next symbol and push onto stack.

(3) Change state as specified in table.

For example: entry "s4" means shift and change to state 4.

Details of Reducing

(1) Pop symbols and intervening states and discard.

(2) Update state: Say one has reduced to variable X. Look at, but do not pop, the stack-top state, say q. Then look up in table new state based on q and X.

(3) Push X.

Example Reduction

For example: entry "R1" means use production 1. Say this is $C \rightarrow \mathsf{x}B$ and the stack is:

Then the reduction pops the B, 7, and x, and discards. It notes stack-top is 4, pushes the C, and updates state to that given in row 4 column C.

Example Grammar

1: $S \rightarrow \texttt{r}\, L$
2: $L \rightarrow L\, \texttt{,}\, I$
3: $L \rightarrow I$
4: $I \rightarrow \texttt{v}$

As discussed later, each state of parser corresponds to partial RHSs of some productions, summarized here as "progress". Eos stands for end-of-string:

The Table

State	*Progress*	r	,	v	eos	S	I	L
0		s1				2		
1	r			s3			4	5
2	S				*acc*			
3	v		R4		R4			
4	I		R3		R3			
5	r L		s6		R1			
6	L ,			s3			7	
7	L , I		R2		R2			

Example Parsing: `rv,v`

Curr state	*Stack*	*To read*	*Operation*
0		`rv,v`	s1
1	0 `r`	`v,v`	s3
3	0 `r` 1 `v`	`,v`	R4
4	0 `r` 1 I	`,v`	R3
5	0 `r` 1 L	`,v`	s6
6	0 `r` 1 L 5 `,`	`v`	s3
3	0 `r` 1 L 5 `,` 6 `v`		R4
7	0 `r` 1 L 5 `,` 6 I		R2
5	0 `r` 1 L		R1
2	0 S		acc

Example: Arithmetic Again

1: $E \rightarrow E + T$
2: $E \rightarrow T$
3: $T \rightarrow T \times F$
4: $T \rightarrow F$
5: $F \rightarrow (E)$
6: $F \rightarrow \mathsf{n}$

The table. . .

State	*Progress*	+	×	(	)	n	eos	E	T	F
0				s1		s2		3	4	5
1	(			s1		s2		6	4	5
2	n	R6	R6		R6		R6			
3	E	s7					*acc*			
4	T	R2	s8		R2		R2			
5	F	R4	R4		R4		R4			
6	(E	s7			s9					
7	E+			s1		s2			10	5
8	T×			s1		s2				11
9	(E)	R5	R5		R5		R5			
10	E+T	R1	s8		R1		R1			
11	T×F	R3	R3		R3		R3			

Summary

Parsing can be performed top down or bottom up. In the latter, the string is processed through a series of shifts (pushing input symbol) and reductions (replacing right-hand side of production by left-hand variable). A table can be produced for efficient parsing.