LALR Parsing Handout 1

Some, but not all, context-free languages can be parsed with an LALR parser. The input of the parser is a string in the language, while the output is a reverse rightmost derivation.

Here is a context-free grammar G for a "toy" algebraic language, whose start symbol is E (for *expression*), followed by the ACTION and GOTO tables for an LALR parser for G.

1. $E \to E + E$
2. $E \to E * E$
3. $E \to a$

The symbol a represents any variable.
The parser stack contains grammar symbols, and each of those symbols must have an associated *stack state* written as a subscript. The bottom of stack symbol has the subscript 0.
We annotate the right-hand sides of the production with stack symbols:

1. $E \to E_{1,3,5} +_2 E_3$
2. $E \to E_{1,3,5} *_4 E_5$
3. $E \to a_6$

	ACTION				GOTO
	a	$+$	$*$	\$	E
0	s6				1
1		s2	s4	HALT	
2	s6				3
3		r1	s4	r1	
4	s6				5
5		r2	r2	r2	
6		r3	r3	r3	

The LALR parser has three parts: the stack which grows and shrinks, the input file from which symbols are read one at a time, and the output file. We use \$ for both bottom of stack and end of file. We assume 1-lookahead, *i.e.,* the parser can peek at the next input symbol without necessarily reading it. The parser can also peek at the top stack state without necessarily poppling it.

Steps of the LALR Parser. The LALR parser operates in steps. At each step the parser peeks at the top stack state and the next input symbol, which may be either a terminal of the language or the end of file symbol.

Each row of the table is headed by a stack state, an number from 0 to 6 in this case. The columns of the ACTION table are labeled by the possible input symbols, including \$. Each column of the GOTO table is headed by a variable of the grammar; in our example, there is only one variable, the start symbol E.

A step operates as follows.

1. Peek at the top stack state and the next input symbol, and follow the instructions in the appropriate entry. A blank entry means that that combination of stack state and input symbol will never occur if the input string is a generated by G.

2. There are three kinds of actions, halt, shift, and reduce. HALT means that the parser if finished. The input file will be empty and the stack will have only one symbol, the start symbol with subscript 1: *i.e.,* E_1 in our example. We pop that E, which becomes the root of the parse tree.

3. If The entry s followed by a number is a *shift*. The number must be a stack state. The first

symbol of the input file is read, and then is pushed onto the stack, and the stack state is pushed on top of that.

4. If the action is r following by a number, that number must be the name of one of the productions. The top one or more symbols in the stack will be the right hand side of that production, along with stack states. That string is called a *handle*. The entire handle is popped, and then the left-hand side is pushed. The left hand side will be a variable. It will be given a stack state as determined by the GOTO table, which depends on the previous stack state. In this case, the name of the production is appended to the output file.

Example Computation. We show the computation of our LALR for one input string. Let the input string be $w = a + a * a * a + a$. At each step, we show the stack, the remaining input, and the output. The bottom of the stack is shown to the left, the top to the right.

$\$_0$	$a + a * a * a + a\$$		
$\$_0 a_6$	$+a * a * a + a\$$		$s6$
$\$_0 E_1$	$+a * a * a + a\$$	3	$r3$
$\$_0 E_1 +_2$	$a * a * a + a\$$	3	$s2$
$\$_0 E_1 +_2 a_6$	$*a * a + a\$$	3	$s6$
$\$_0 E_1 +_2 E_3$	$*a * a + a\$$	33	$r3$
$\$_0 E_1 +_2 E_3 *_4$	$a * a + a\$$	33	$s4$
$\$_0 E_1 +_2 E_3 *_4 a_6$	$*a + a\$$	33	$s6$
$\$_0 E_1 +_2 E_3 *_4 E_5$	$*a + a\$$	333	$r3$
$\$_0 E_1 +_2 E_3$	$*a + a\$$	3332	$r2$
$\$_0 E_1 +_2 E_3 *_4$	$a + a\$$	3332	$s4$
$\$_0 E_1 +_2 E_3 *_4 a_6$	$+a\$$	3332	$s6$
$\$_0 E_1 +_2 E_3 *_4 E_5$	$+a\$$	33323	$r3$
$\$_0 E_1 +_2 E_3$	$+a\$$	333232	$r2$
$\$_0 E_1$	$+a\$$	3332321	$r1$
$\$_0 E_1 +_2$	$a\$$	3332321	$s2$
$\$_0 E_1 +_2 a_6$	$\$$	3332321	$s6$
$\$_0 E_1 +_2 E_3$	$\$$	33323213	$r3$
$\$_0 E_1$	$\$$	333232131	$r1$
HALT			

1. Sketch the parse tree.

The output is the reverse rightmost derivation 333232131.

2. This grammar is ambigous, but the parser resolves all ambiguities, computing only one derivation for each string in the language. In any derivation produced by the parser, addition and multiplication are both left associative. Left associativity of addition is guaranteed by the entry r1 in row 3, in the column headed by the plus sign. Which entry of the action table guarantees that multiplication is left associative?

3. Which two entries in the action table cause multiplication to have precedence over addition?

4. Write the computation of the parser if the input is $a * a + a$. (Attach another page.)