Simple LALR Parsers

We use "\$" as both the bottom of stack symbol and the end of file symbol. The instantaneous description, **id**, is a string consisting of the stack, from bottom to top, followed by the current (remaining) input file starting with the next symbol, followed by the current output file. The symbols in the stack above the bottom are alternating stack states and grammar symbols, where the stack states are written as subscripts for clarity. The last symbol in the input file will be \$.

In all of the LALR parsers given below, there will be two special stack states, 0, the state of the empty stack, and 1, the state when the start symbol is just above the bottom. The stack is initially $\$_0$, and the last configuration of the stack is always $\$_0 S_1$, where S is the start symbol. We give several examples of simple LALR parsers. When we write a grammar, we include stack states as subcripts.

Example 1: Dangling else

The following LALR parser demonstrates how the "dangling else" can be resolved. Let L be the language generated by the ambiguous CF grammar below, with start symbol S.

1. $S \to a_2$
2. $S \to w_3 S_4$
3. $S \to i_5 S_6$
4. $S \to i_5 S_6 e_7 S_8$

Here are the ACTION and GOTO tables.

	a	w	i	e	\$	E
0	$s2$	$s3$	$s5$			1
1					**halt**	
2				$r1$	$r1$	
3	$s2$	$s3$	$s5$			4
4				$r2$	$r2$	
5	$s2$	$s3$	$s5$			6
6				$s7$	$r3$	
7	$s2$	$s3$	$s5$			8
8				$r4$	$r4$	

Which entry of the ACTION table resolves the dangling else problem?

We now show the action of our parser on the input string *iiwaea*.

$\$_0$	$iiwaea\$$	
$\$_0 i_5$	$iwaea\$$	
$\$_0 i_5 i_5$	$waea\$$	
$\$_0 i_5 i_5 w_3$	$aea\$$	
$\$_0 i_5 i_5 w_3 a_2$	$ea\$$	
$\$_0 i_5 i_5 w_3 S_4$	$ea\$$	1
$\$_0 i_5 i_5 S_6$	$ea\$$	12
$\$_0 i_5 i_5 S_6 e_7$	$a\$$	12
$\$_0 i_5 i_5 S_6 e_7 a_2$	\$	12
$\$_0 i_5 i_5 S_6 e_7 S_8$	\$	121
$\$_0 i_5 S_6$	\$	1214
$\$_0 S_1$	\$	12143

halt

Example 2: Left Associativity of an Operator

The following grammar generates an algebraic language with one operator, subtraction, and one variable, x. We use E (for expression) as the start symbol. Subtraction is left-associative. For example, $8 - 4 - 2$ is 2, not 6.

1. $E \to x_2$
2. $E \to E -_3 E_4$

Here are the ACTION and GOTO tables.

	x	$-$	\$	E
0	$s2$			1
1		$s3$	**halt**	
2		$r1$	$r1$	
3	$s2$			4
4		$r2$	$r2$	

Which entry of the ACTION table guarantees that subtraction is left-associative?

Example 3: Binary and Unary Minus Sign

In computer languages, $- - 4$ means 4, although your algebra teacher would not like it. How does an LALR parser distinguish between the two operators, and enforce the priority of the unary operator?

1. $E \to x_2$
2. $E \to E -_3 E_4$
3. $E \to -_5 E_6$

Here are the ACTION and GOTO tables. **Error Fixed Friday November 27. Header of column of GOTO table is S, not E.**

	x	$-$	\$	S
0	$s2$	$s5$		1
1		$s3$	**halt**	
2		$r1$	$r1$	
3	$s2$	$s5$		4
4		$r2$	$r2$	
5	$s2$	$s5$		6
6		$r3$	$r3$	

Example 4: Right Associativity of an Operator

Exponentiation is right associative. For example, $2^{3^2} = 512$, not 64. We'll use "$\wedge$" for exponentiation.

1. $E \to x_2$
2. $E \to E \wedge_3 E_4$

Here are the ACTION and GOTO tables.

	x	$\wedge$	\$	E
0	$s2$			1
1		$s3$	**halt**	
2		$r1$	$r1$	
3	$s2$			4
4		$s3$	$r2$	

Which entry of the ACTION table guarantees that exponentiation is right-associative?

Example 5: Precedence of Operators

Multiplication has precedence over subtraction. For example, $7 - 3 * 2$ is 1, not 8. Consider the language generated by the CF grammar:

1. $E \to x_2$
2. $E \to E -_3 E_4$
3. $E \to E *_5 E_6$

Here are the ACTION and GOTO tables.

	x	$-$	$*$	$\$$	E
0	$s2$	$s3$			1
1		$s3$	$s5$	**halt**	
2		$r1$	$r1$	$r1$	
3	$s2$				4
4		$r2$	$s5$	$r2$	
5	$s2$				6
6		$r3$	$r3$	$r3$	

Which entry of the ACTION table guarantees that multiplication has precedence over subtraction?

Example 6: Parentheses

1. $E \to x_2$
2. $E \to E -_3 E_4$
3. $E \to (_5 E_6)_7$

	x	$-$	(	)	$\$$	E
0	$s2$		$s5$			1
1		$s3$			**halt**	
2		$r1$		$r1$	$r1$	
3	$s2$		$s5$			4
4		$r2$		$r2$	$r2$	
5	$s2$		$s5$			6
6		$s3$		$s7$		
7		$r3$		$r3$	$r3$	

Unlike the first four, this grammar is unambiguous, so there is no ambiguity to resolve

Example 7: Combining Examples 2, 3, 4, and 5.

How would you create an LALR parser for a grammar which had all of the above operators? Let's throw in addition, just for fun! How many productions do you need? How many stack states?

Example 8: Generating Identifiers

Suppose an identifier must start with a letter and can contain any combination of letters and numerals. Let's say identifiers are case-insensitive. How many productions do you need to generate all identifiers? Note that the language of all identifiers is regular.