University of Nevada, Las Vegas Computer Science 456/656 Fall 2023

Answers to Assignment 6: Due Friday November 10, 2023

The problems on this assignment deal with LALR parsing. There will be no homework problem on undecidability this week, but here is a quick proof you should read now, to get started thinking about undecidability.

The Halting Problem is Undecidable.

A language $L \subseteq \Sigma^*$ is called *decidable*, or *recursive*, if there is a machine M_L such that, given input $w \in \Sigma^*$, halts and outputs 1, meaning "yes," if $w \in L$ and 0, meaning "no," if $w \notin L$. There is no time limit on M_L's computation; it just must halt eventually for for any input string. A language which is not decidable is called *undecidable.*

An instance of the halting problem is a string $\langle M \rangle w$ where M is a machine and w is a string. The language L_{HALT} written simply as HALT) consists of those instances such that, given input w, M halts.

Theorem: L_{HALT} is undecidable.

Our proof of the theorem is by contradiction. Suppose HALT is decidable. Consider the machine M_{DIAG} defined by the following program:

```
Read a machine description $\langle M \rangle$.
If ( $M$ halts with input $\langle M \rangle$ ) enter an infinite loop.
Else write "1" and halt.
```

That is M_{DIAG} accepts all descriptions of machines which do not halt given their own descriptions as input. Note that The program cannot get stuck at the If statement since the halting problem is decidable.

Now we get the contradiction. Does M_{DIAG} halt if the input string is $\langle M_{\text{DIAG}} \rangle$? There are two cases.

Let M_{DIAG} be given the input string

Case 1: M_{DIAG} halts. In that case, the If condition is true, which means that it must enter an infinite loop, contradiction.

Case 2: M_{DIAG} does not halt. In that case, the If condition is false, which means that it executes the Else branch, hence must halt, contradiction.

Our conclusion is that HALT is undecidable.

LALR Parsing

An LALR (Look Ahead Left-to-right Rightmost derivation) parser,[1] is a DPDA with output. Languages that can be parsed by an LALR parser include most of the context-free languages we've discussed this semester, such as algebraic languages, including the language of regular expressions over a given alphabet, and simple programming language models. The input of the parser is a string of the language, and the output is a reverse rightmost derivation of that string. LALR parsing can work with some ambiguous grammars by removing the ambiguity during the computation.

[1]Invented by Frank DeRemer in his 1969 PhD dissertation

LALR parsing is quickly explained in LALR Parsing Handout 1. LALR Parsing Handouts 2 and 3 cover the same material with many examples and explanations.

An LALR parser reads an *input* string and write an *output* string. The last symbol of the input string is the end-of-file symbol, which I write as the dollar sign. The stack has three kinds of symbols:

Variables of the grammar,
Terminals of the grammar, members of the alphabet Σ of the language,
Stack states, integers starting at 0.

Page 1 of Handout 1 first shows a CFG grammar for a simple algebraic language with start symbol E. Then, the grammar is repeated with subscripts which denote stack states. The ACTION and GOTO tables of the parser are then given. A complete computation of the parser with input $a + a * a + a * a + a$ is given on page 2. The first column is the stack, the second column is the remaining input, the third column is the output, and the fourth column designates the action at that step.

Here are the homework problems.

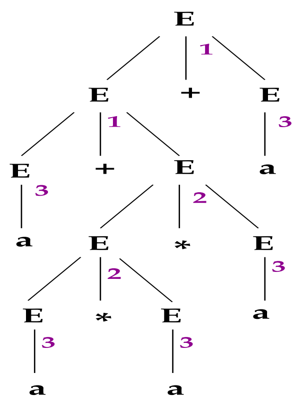
Exercises 1, 2, 3, and 4, on the second page of Handout 1.

Recall the subscripted grammar, and the ACTION and GOTO tables.
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	

1.1: Sketch the parse tree of $a + a * a * a + a$.



The leaves of the tree in postorder give the input string. Each internal node is labeled with the number of a production. Reading those numbers in postorder gives the output string of the LALR parser, the reverse rightmost derivation.

1.2: 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, column "+" . If addition were right associative, that entry would be $s2$.

Which entry of the action table guarantees that multiplication is left associative? Similarly, the entry r2 in row 5 column "$*$" . If multiplication were right associative, that entry would be $s4$.

1.3: Which two entries in the action table cause multiplication to have precedence over addition?

The entry r2 in row 5 column "+" guarantees that multiplication will be executed before addition for the string $a * a + a*$. The entry s4 in row 3 column "$*$" guarantees that multiplication will be executed before addition for the string $a + a * a*$. If addition had precedence over multiplication, those entries would be s2 and r1, respectively.

1.4: Write the computation of the parser if the input is $a * a + a$.

$\$_0$	$a * a + a\$$		
$\$_0 a_6$	$*a + a\$$		$s6$
$\$_0 E_1$	$*a + a\$$	3	$r3$
$\$_0 E_1 *_4$	$a + a\$$	3	$s4$
$\$_0 E_1 *_4 a_6$	$+a\$$	3	$s6$
$\$_0 E_1 *_4 E_5$	$+a\$$	33	$r3$
$\$_0 E_1$	$+a\$$	332	$r2$
$\$_0 E_1 +_2$	$a\$$	332	$s2$
$\$_0 E_1 +_2 a_6$	$\$$	332	$s6$
$\$_0 E_1 +_2 E_3$	$\$$	3323	$r3$
$\$_0 E_1$	$\$$	33231	$r1$
$\$_0 E_1$	$\$$	33231	HALT

Consider the grammar G given in Figure 1 of Handout 2, which uses parentheses. The grammar is repeated on page 3 with subscripts indicating stack states. The ACTION and GOTO tables are completed on page 5. Work exercises 1, 2, 3 on page 5 of Handout 2.

Recall the annotated grammar and the parser tables:

1. $E \longrightarrow E_{1,7} +_2 E_3$
2. $E \longrightarrow E_{1,3,7} *_4 E_5$
3. $E \longrightarrow (_6 E_7)_8$
4. $E \longrightarrow a_9$

	ACTION						GOTO
	a	+	$*$	(	)	\$	E
0	s9			s6			1
1		s2	s4			**halt**	
2	s9			s6			3
3		r1	s4		r1	r1	
4	s9			s6			5
5		r2	r2		r2	r2	
6	s9			s6			7
7		s2	s4		s8		
8		r3	r3		r3	r3	
9		r4	r4		r4	r4	

2.1: Give the complete computation of the parser for G with input $(a + a) * a$.
Hint: There are 13 steps. (I may have miscounted.)
Hint: At one step, the stack is $_0(_6E_7 +_2 a_9$, the input file is $) * a\$$, and the output file is 4.
Hint: At the end of the computation, the stack is $_0E_1$, the input file is \$, and the output file is 441342.

2.2: Row 3 column "+" .

2.3: Row 3 column "*" .

2.4: Row 5 column "+" and row 3 column "*" .

3. Give the complete computation of the parser for input string $(a + a) * a$.

STACK	INFILE	OUTFILE	ACTION
$\$_0$	$(a + a) * a\$$		
$\$_0(_6$	$a + a) * a\$$		$s6$
$\$_0(_6a_9$	$+a) * a\$$		$s9$
$\$_0(_6E_7$	$+a) * a\$$	4	$r4$
$\$_0(_6E_7+_2$	$a) * a\$$	4	$s2$
$\$_0(_6E_7 +_2 a_9$	$) * a\$$	4	$s9$
$\$_0(_6E_7 +_2 E_3$	$) * a\$$	44	$r4$
$\$_0(_6E_7$	$) * a\$$	441	$r1$
$\$_0(_6E_7)_8$	$*a\$$	441	$s8$
$\$_0E_1$	$*a\$$	4413	$r3$
$\$_0E_1*_4$	$a\$$	4413	$s4$
$\$_0E_1 *_4 a_9$	\$	4413	$s9$
$\$_0E_1 *_4 E_5$	\$	44134	$r4$
$\$_0E_1$	\$	441342	$r2$
$\$_0E_1$	\$	441342	HALT