CS 164, Spring 2005 **CS 164: Homework #4** **P. N. Hilfinger**

Due: Wednesday, 23 February 2005

General instructions about homework. Use the command '`submit hw4`' to submit your homework. We will not accept homework in any other form. Unless the problem specifies otherwise, please put your solutions in a file named `hw4.txt`.

1. [From Aho, Sethi, and Ullman] Consider the following ambiguous grammar:

`E` → `E '+' E` | `E '*' E` | `'(' E ')'` | **id**

and this parsing table for it (end-of-input, ⊣, is never shifted):

State	**id**	'+'	'*'	'('	')'	⊣	E
0	s3			s2			s1
1		s4	s5			acc	
2	s3			s2			s6
3		r4	r4		r4	r4	
4	s3			s2			s7
5	s3			s2			s8
6		s4	s5		s9		
7		r1	s5		r1	r1	
8		r2	r2		r2	r2	
9		r3	r3		r3	r3	

In this table, `s`n denotes to a transition in the state machine ("go to state n on seeing this lookahead") and `r`n means "reduce the last symbols just scanned by the state machine (i.e., on top of the parsing stack) using production n." The productions are numbered left to right from 1; production 1 is $E \rightarrow E$ `'+'` E. Blank entries indicate errors. The start state is 0. Use the table to produce a reverse rightmost derivation of the string `id+id+id*(id+id)`. That is, give the sequence of reductions discovered by the parser.

2. Since the grammar of problem 1 is ambiguous, there are several possible parsing tables for it, depending on how we wish to resolve ambiguities. Show the modifications to the table in problem 1 that are necessary to

a. Give `'+'` and `'*'` equal precedence, keeping them left associative.

b. Give `'+'` higher precedence than `'*'`, keeping them left associative.

c. Give `'+'` lower precedence than `'*'`, and making `'+'` right associative (`'*'` stays left associative).

d. Make it illegal to mix different operators without parenthesization. For example, to make the example in Exercise 1, above, illegal.

3. Consider the string `id+id(id)`, which is illegal according to the grammar of the preceding two problems.

a. Show what happens when you try to parse it using the parsing table from problem 1. That is, show all the steps taken by the parser up to the point where the machine finds no valid transition.

b. Now modify the table as follows: for each row that contains at least one reduction (`r`n), replace all the empty (error) entries in the action table for that row (i.e., the part between the vertical lines) with that reduction. For example, all entries in row 9 (except for E) would become `r3`, and all entries except that for `'*'` (and E) in row 7 would become `r1`. Show what happens when you try to parse the illegal string with this revised table. (This optimization—introducing *default rules*—makes tables more compressible; the question is whether it causes the parser to recognize illegal sentences.)

4. [From Aho, Sethi, and Ullman] For the following grammar:

```
E  →  E '+' T  |  T
T  →  T  F  |  F
F  →  F '*'  |  'a'  |  'b'
```

a. Construct the LR(0) machine table for this grammar (that is, leave out all reductions).

b. Fill in reduction entries using FOLLOW sets (the SLR(1) construction). That is, when a reduction to symbol Q is possible in a state, do that reduction for all terminal symbols that are in FOLLOW(Q).

c. Fill in reduction entries using LALR(1) lookahead.