CS 164, Fall 2006 **CS 164: Homework #4** **P. N. Hilfinger**

Due: Monday, 9 October 2006

New instructions about homework. Check out the homework framework with the command:

`svn checkout svn+ssh://cs61b-tb@`*HOST*`/_hw/`*LOGIN*

where *LOGIN* is your instructional login. Submit homework by committing the `hw4` subdirectory of this directory.

Unless the problem specifies otherwise, please put your solutions in a file named `hw4.txt` in your `hw4` subdirectory.

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, sn denotes to a transition in the state machine ("go to state n on seeing this lookahead") and rn 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 $E \to E$ '+' 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, we had to add information to get a table out of it, since otherwise, some of the entries would be unresolved. 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 make `'+'` 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. Here's another LR parsing table.

STATE	⊣	'/'	':'	'<'	'>'	'i'	'v'	E	F	P	S
0	r1	r1	r1	r1	r1	r1	r1				s1
1	acc						s3		s4		
3			s5								
4	r2	r2	r2	r2	r2	r2	r2				
5						s7	s6	s8			
6	r7	r7	r7	s9	r7	r7	r7			s10	
7	r4	r4	r4	r4	r4	r4	r4				
8		s11									
9						s7	s6	s12			
10	r3	r3	r3	r3	r3	r3	r3				
11	r5	r5	r5	r5	r5	r5	r5				
12					s13						
13	r6	r6	r6	r6	r6	r6	r6				

The reductions here have the following properties:

```
r1: 0 symbols  →  S
r2: 2 symbols  →  S
r3: 2 symbols  →  E
r4: 1 symbols  →  E
r5: 4 symbols  →  F
r6: 3 symbols  →  P
r7: 0 symbols  →  P
```

For example, production #2 has nonterminal `S` on the left-hand side and two symbols on the right-hand side (but I won't tell you what they are).

By considering the parse of the following sentence:

```
v:v<v>/v:i/⊣
```

reconstruct the grammar (that is, determine what symbols appear on the right-hand sides of the seven productions).

5. [From Aho, Sethi, Ullman] A grammar is called ϵ-*free* if there are either no ϵ productions, or exactly one ϵ production of the form $S \rightarrow \epsilon$, where S is the start symbol of the grammar, and does not appear on the right side of any productions. (We write ϵ productions either as '$A \rightarrow$ ' or '$A \rightarrow \epsilon$'; both mean the same thing: there are no terminals or non-terminals to the right of the arrow). Describe an algorithm to change a grammar into an equivalent ϵ-free grammar (i.e., one recognizing the same language). By "describe," I mean "give sufficient detail that a programmer could probably figure out what you meant and convert it into a program." Apply your algorithm to the grammar:

$$\texttt{S} \rightarrow \texttt{aSbS} \mid \texttt{bSaS} \mid \epsilon$$