| CS 431 Translation of Computer Languages | Spring 2002 |
|---|---|

# Exam 2

Given: 18 April 2002 Due: 24 April 2002, 4 PM, Bryan 509

You may use any materials for this exam. Solutions will be graded on a curve; partial credit will be given where sufficient detail is provided.

Where possible, answer questions on the pages of the exam.

By turning in this exam, you agree that you have read, signed, and abided by the Statement of Student Academic Integrity.

This exam is due precise by the time stated. **This deadline will be strictly enforced.**

You may use *CUP*, and your answers involving *CUP* can involve printouts you produce and annotate using *CUP*. To set up your workspace for the exam *CUP* files:

```
mkdir exam2
cd exam2
make -f (tilde)cs431/exam2/Makefile Setup
make states
```

The last command performs the grammar analysis.

| Name: | | |
|---|---|---|
| Problem Number | Possible Points | Received Points |
| 1 | 25 | |
| 2 | 25 | |
| 3 | 25 | |
| 4 | 25 | |
| Total | 100 | |

1. (25 points) While teaching the nuances of programming in C, you have discovered that students tend to launch into parenthesized expressions of such depth, complexity, and grandeur, that by the time they have reached the end of the expression, they have forgetten the symbol used to open the expression,[1] and therefore tended to close expressions with an aribtrary closing symbol. In other words, expressions are opened with either a "(" or a "[", but then the expression may be closed with either a ")" or a "]". For example, you common see mistakes such as:

   - A[3+5-12+13/x*3)
   - 6*(10+4]
   - x-(10+3*(a-1)+f[13*(8+1))]

   Known for your generousity as well as your brown-belt third-degree in LR parsing, you decide to modify the expression grammar for the programming language, so that "any closer will do". However, to remind the students of your graciousness (as well as to point out their errors), you insist that the grammar report transgressions of the form shown above.

   You formulated the following grammar solution, which you can find in *CUP* form as `prob1.cup`:

```
S    →  ( Good )
     |  [ Good ]
     |  ( Bad ]
     |  [ Bad )
Good →  e
        {: System.out.println("Nice"); :}
Bad  →  e
        {: System.out.println("Naughty"); :}
```

   where "e" normally would be the nonterminal "E" that derives standard infix expressions; here, we shall represent such things as the terminal "e".

*Continued on next page...*

[1] Mark Twain, in describing the German language, claimed that by the time a speaker reached the end of the sentence, where the postponed verbs unstack, that the audience was either on the edge of their seats in anticipation of the verbs, or else had forgotten the drift of the sentence entirely.

(a) **(5 points)** Show the $LALR(1)$ collection of items for this grammar.

*Continued on next page...*

(b) **(10 points)** Would more lookahead help $LALR(1)$ to resolve the inadequate states? Why or why not?

(c) **(10 points)** Is the grammar $LR(1)$? Justify your answer by showing the $LR(1)$ collection of items below.

2. (25 points) While you are at it, you discover that students also tend to exaggerate the number of opening or closing parentheses needed to balance a formula. For example, you commonly see:

   - (3+4))
   - ((4 * 7)
   - ((3+1)))

   You wisely devise the following grammar solution, which you can find in *CUP* form as `prob2.cup`:

$$
\begin{array}{lcl}
\text{E} & \rightarrow & \text{E + T} \\
 & | & \text{T} \\
\text{T} & \rightarrow & \text{T * F} \\
 & | & \text{F} \\
\text{F} & \rightarrow & \text{digit} \\
 & | & \text{(E)} \\
 & | & \text{((E)} \\
 & | & \text{(E))}
\end{array}
$$

*Continued on next page...*

(a) **(5 points)** Show the $LALR(1)$ collection of items for this grammar.

*Continued on next page...*

(b) (5 points) Would more lookahead (LALR(2), LALR(3), etc.) resolve the inadequate states? Why or why not?

(c) (5 points) Would $LR(1)$ help? Why or why not?

(d) **(10 points)** Consider the following grammar:

$$
\begin{array}{lcl}
\text{S} & \rightarrow & \text{B x x x y z} \\
 & | & \text{C x x x x z} \\
 & | & \text{x B x x x x y} \\
\text{B} & \rightarrow & \text{w} \\
\text{C} & \rightarrow & \text{w}
\end{array}
$$

For some values of $i, j, k$ the above grammar is $LR(i)$ and $SLR(j)$ but not $SLR(k)$.

i. **(3 points)** What is the smallest correct value for $i$?

ii. **(3 points)** What is the smallest correct value for $j$?

iii. **(4 points)** What is the largest correct value for $k$?

3. (25 points) Suppose a language (like Java in fact) does not allow an inner scope to hide any variables declared in an outer scope. For example, consider the program (a) shown below

```
{
   int x;
   {
      int x;
   }
}
```

(a)

```
{
   int x;
}
{
   int x;
}
```

(b)

The inner declaration of `x` hides the outer declaration, and so this is considered illegal. Howver, (b) shown above has scopes that do not hide any variables.

Describe in detail the changes you would make to lab 4 so that declarations of variables that hide outer declarations are flagged as illegal. More specifically, provide code for handling `LocalDeclaring` in the symbol table visitor below. Assume `sfi` is your symbol table reference.

```
SymTabInterface sfi;  // already set up for my use
public void visit(LocalDeclaring ld) {









}
```

4. (25 points) Consider the following Java class:

```
public class Item {
   public Item thing1;
   public Item thing2;
}
```

and a program below that uses that class:

```
static Item vampire = new Item();  // lives forever
public void foo() {
   Item
       a = new Item(),
       b = new Item(),
       c = new Item(),
       d = new Item(),
       e = new Item();

    a.thing1 = b;
    b.thing1 = a;
    c.thing1 = d;
    d.thing1 = c;
    d.thing1 = e;
    d.thing2 = e;

    vampire.thing1 = e;
    vampire.thing1 = d;
    vampire.thing1 = c;
}
```

Once `foo` returns, how can the objects instantiated by `foo` be collected? To answer this question, fill in the following table. Each square deserves a "T" or "F". Write "T" if the specified garbage-collection method can determine that object's death after `foo` returns. Otherwise, write "F".

| Object referenced by | Collected by | | | |
|---|---|---|---|---|
| | Mark Sweep | Copy Collect | Ref Count | Contam-inated |
| a | | | | |
| b | | | | |
| c | | | | |
| d | | | | |
| e | | | | |