THE UNIVERSITY OF CALGARY

FACULTY OF SCIENCE

FINAL EXAMINATION

COMPUTER SCIENCE 411

April, 2002 Time: 2 hrs.

Instructions

The exam contains questions totaling 110 points. Answer all questions.

This exam is closed book.

Please note that all grammars are provided using the CFG format: non-terminals are in lower-case and terminals are in upper-case. The productions associated with a non-terminal are separated by bars and ended with a period.

20 marks

1. Consider the following grammar:

```
stmt -> lexpr ASS expr.

lexpr -> lexpr STAR
       | lexpr index
       | ID.

index -> LPAR sign_num RPAR.

sign_num -> sign num.

sign -> NEG
      |.

num -> num DIGIT
     | PDIGIT.

expr -> ID.
```

10 marks

(a) Calculate the *first* and *follow* sets for the above grammar. Indicate which non-terminals are nullable and which are endable.

2 marks

(b) Explain why this grammar is not LL(1).

8 marks

(c) Transform the grammar to remove left recursion. Is the result LL(1)?

40 marks

2. For each of these grammars you are expected to calculate the vital statistics (that is nullable, endable, first, and follow properties) and to give the appropriate item automaton.

10 marks

(a) Is the following grammar LL(1), SLR(1), or LALR(1):

```
s -> a s
   | B.
a -> s a
   | A.
```

What language does this grammar recognize? Is this grammar ambiguous?

10 marks

(b) Show that the following grammar is LL(1) but not SLR(1):

```
s -> a A a B | b B b A.
a -> .
b -> .
```

Show that this grammar is LALR(1) and give the transformation of it to an SLR(1) grammar.

10 marks

(c) Is the following grammar LL(1), SLR(1), or LALR(1):

```
s -> x y A.
x -> A
   | y B.
y -> C
   |.
```

10 marks

(d) Show that the following grammar is LR(1) but neither LL(1) nor LALR(1):

```
s -> A a D | B b D | c.
c -> A b E | B a E.
a -> C.
b -> C.
```

Is this grammar ambiguous?

25 marks

3. This question is concerned with the runtime organization of a stack-based environment:

   (a) Explain what the term *activation record* means. Describe the organization of an activation record for a function.

   (b) Explain the difference between the *dynamic link* and the *static link*.

   (c) Explain how one uses an activation record to access *local variables* and *non-local variables*.

   (d) Describe the *caller* and *callee* responsibilities when a function is called.

   (e) Describe what happens on a *return* from a function call.

25 marks

4. Given the following abstract syntax tree for expressions with "let" declarations draw the plumbing diagrams and develop pseudo-code to determine the type of the expression given an initial symbol table:

```
Expr = ADD(Expr,Expr)
     | OR(Expr,Expr)
     | ID String
     | LET(Decls,Expr)
Decls = DCL(Decl)
     | DCLS(Decls,Decl)
Decl = DEC(string,expr)
```

A typical abstract syntax tree (which will not type) could then be:

```
ADD(ID x
   ,LET(DCLS(DCL(DEC(y,OR(ID a,ID b)))
                ,DEC(z,ADD(ID c,ID b)))
        ,OR(ID y,ID z)))
```

You may assume that you have the following operations on your symbol table and types:

```
lookup(st:symtab,str:string ; t:type)
insert(st:symtab,str:string,t:type ; st':symtab)
eq(t:type,t':type ; )
```

for respectively looking up the type of a variable (as most recently inserted), inserting the name of a variable and its type into the symbol table, and testing the equality of types. Note that both the first and last operation can fail and raise an exception. There are just two types in this language:

```
Type = INT | BOOL
```

and `OR` is an operation on booleans while `ADD` is an operation on integers:

```
OR: (BOOL,BOOL) -> BOOL   ADD: (INT,INT) -> INT.
```