CSE 428, Spring 2000. Final exam, 4 May 2000

The exam consists of 7 exercises, totalling 100 points. Write your answers directly on these pages. **Please remember to write your name and SSN at the top of each sheet.**

Exercise 1 (15 pts)

The following grammar generates the language of some of the Types of ML:

```
Type ::= TVar | Type -> Type | Type list | (Type)
```

Where: `Type` and `TVar` are non-terminals, and `->`, `list`, `(`, and `)` are terminals. `TVar` is assumed to generate a set of names (the Type variables) α, β, γ, etc. The productions for `TVar` are not represented here.

1. (4 pts) Show that this grammar is ambiguous by giving two derivation trees for each of the following type expressions:
   (a) α `->` β `->` γ

   (b) α `->` β `list`

2. (11 pts) Modify the grammar so to obtain a unambiguous grammar, generating the same language, and enforcing the following properties:
   (a) `->` is right associative
   (b) `list` has higher priority than `->`

Exercise 2 (12 pts)

For each of the following procedure declarations in C++, say whether its call leaves (after returning) dangling pointers and/or memory leaks. If it leaves dangling pointers, specify whether they are to the heap or to the stack. Assume that `p` is a global variable declared as a pointer to `int` and initialized to `NULL`.

1. `int* foo() { int x; return &x; }`.
   Assume that the call is `p = foo();`

2. `int foo() { int *q; q = new int; *q = 2; return *q; }`.
   Assume that the call is `p = new int; *p = foo();`

3. `void foo() { int *q; p = new int; q = p; }`

4. `void foo() { int *q; q = new int; p = new int; *q = *p; }`

5. `void foo(int* r) { int *q; q = new int; r = q; delete q;}`.
   Assume that the call is `foo(p);`

6. `void foo(int* &r) { int *q; q = new int; r = q; delete q;}`.
   Assume that the call is `foo(p);`

Exercise 3 (10 pts)

Consider the following class declaration in Java:

```
class C {
   private int x;
   private int y;

   public C(int n) { x = n; y = 0; }
   public int inc_and_get_x() { x++; return x; }
   public synchronized int get_y() { if (y==0) wait(); return y; } //ignore synchronizedExceptions
   public synchronized void inc_y() { y++; notifyAll(); }
}
```

This class is meant to be used in a multithread context, and its definition is be complete, i.e. there are no methods other than those defined above.

For each of the following statements, say whether it is true or false

1. There is a lock associated to the class `C`

2. There is a lock associated to each objects of class `C`

3. There is a lock associated to each thread using an object of class `C`

4. We don't need to declare synchronized `inc_and_get_x()`, beacuse it is the only method where `x` occurs

5. If we want that `get_y()` returns only values different from `0`, then we need to replace `if` with `while`

6. When a thread executes an instruction of the form `x.inc_y()` (where `x` is an object of class `C`), the execution of `notifyAll()` sends a notification to all threads which have executed `get_y()` on any object of class `C`, including `x`. If we want to "wake up" only those threads that have executed `get_y()` on `x`, then we should use `notify()` instead.

7. In the body of `inc_y()` we can exchange the order of `notifyAll()` and `y++`

8. When a thread executes the `wait()` instruction it becomes suspended

9. When a thread executes the `wait()` instruction, it releases the lock

10. The notification transfers a thread to the running state, and gives it back the lock on the object.

Exercise 4 (20 pts)

1. (5 pts) Define in ML a datatype `'a tree` representing trees whose nodes contain information of type `'a`, and have an arbitrary number of children. Assume that a tree cannot be empty; the smallest tree consists of a single node with no children.

2. (15 pts) Define in ML the function `sum_tree: int tree -> int` which gives the sum of all the nodes of the tree.

   Note: `int tree` is (the integer instance of) the data type defined above.

   Warning: write your solution in ML. Solutions in other languages (C++, Pascal etc.) will not be accepted.

Exercise 5 (10 pts)

Consider the function `reduce : ('a * 'b -> 'b) -> 'b -> 'a list -> 'b`:

```
fun reduce f v [] = v
  | reduce f v (x::l) = f(x, reduce f v l);
```

For each of the following expressions, say what is the result of its evaluation in ML.

- `reduce (fn (x,y) => if x=[] then y else x::y) [] [[5],[],[7,3]];`

- `reduce (fn (x,y) => if x>y then x else y) 0 [5,7,3];`

Exercise 6 (12 pts)

For each of the following ML definitions of the function `f`, say whether is it typeable and, in the adfirmative case, what is its type.

Warning: If it is typeable, you have to give the type. The answer "typeable" alone will not receive points.

- `fun f g h x = (g x, h x)`

- ```
  fun f g x 0 = x
    | f g x n = g (f g x (n-1))
  ```

- ```
  fun f x [] = []
    | f x (y::L) = (x+y) :: (f x L)
  ```

Exercise 7 (21 pts)

A binary tree, in Prolog, can be represented by a term, like in ML. For instance, we can stipulate to use the name `nod` for the node constructor, and `emp` for the empty tree. With this convention, the following terms

```
T1 = nod(nod(emp,1,emp),2,nod(emp,3,emp))
T2 = nod(nod(emp,a,emp),b,nod(nod(emp,c,emp),d,emp))
T3 = nod(nod(nod(nod(emp,a,emp),b,emp),c,emp),d,nod(nod(emp,e,emp),f,nod(emp,g,emp)))
```

represent the following trees::

```
     T1                 T2                   T3

     2                  b                    d
    / \                / \                  / \
   1   3              a   d                c   f
                         /                /   / \
                        c                b   e   g
                                        /
                                       a
```

Define a predicate `balanced(T)` which holds if `T` is balanced. Remember that a tree is balanced if, for each node, the difference between the height of the left subtree and the height of the right subtree is at most one. For instance, of the tree above, `T1` and `T2` are balanced, while `T3` is not. Feel free to define auxiliary predicates if you wish.

Note: the Prolog predicate for "smaller or equal" (on integers) is `=<`. The one for "greater or equal" is is `>=`. They are both infix.

Warning: write your solution in Prolog. Solutions in other languages (ML, C++ etc.) will not be accepted.