Paradigmas de Programación
Parcial 2

14 de Junio de 2006

1. Ejercicios

1. (2 puntos) Usamos concurrencia para crear un contenedor que se puede modificar. Creamos un hilo que usa un procedimiento recursivo para leer un stream. El stream tiene dos comandos posibles: `access(X)`, el cual bindea `X` al contenido del contenedor y `assign(X)`, que asigna `X` como el nuevo contenido.

```
fun {MakeState Init}
  proc {Loop S V}
     case S of access(X)|S2 then
        X=V {Loop S2 V}
     []assign(X)|S2 then {Loop S2 X}
     else skip end
   end S
  in
    thread {Loop S Init} end S
end
```

La llamada a `S={MakeState 0}` crea un nuevo contenedor con contenido inicial 0. Usamos el contenedor poniendo commandos en el stream.

```
declare S1 X Y in
S=access(X) | assign(3) | access(Y) | S1
```

Esto bindea X a 0 (el contenido inicial), pone 3 en el contenedor, y despues bindea Y a 3.

- Escriba una funcion SumList que sume el contenido de una lista, y que ademas use una celda para saber cuantas veces a sido llamada
- Rescriba la funcion SumList de manera que esta *no* use celdas, y en que use el contenedor definido antes para llevar cuenta de cuantas veces ha sido llamada.

2. (3 puntos)

Segun el teorico, esta función construye la representacion de la clase C1 heredando de C2 y C3, donde C1, C2, C3 son definiciones de clases.

```
fun {From C1 C2 C3}
  c(methods:M1 attrs:A1)={Unwrap C1}
  c(methods:M2 attrs:A2)={Unwrap C2}
  c(methods:M3 attrs:A3)={Unwrap C3}
  MA1={Arity M1}
  MA2={Arity M2}
  MA3={Arity M3}
  ConfMeth={Minus {Inter MA2 MA3} MA1}
```

```
   ConfAttr={Minus {Inter A2 A3} A1}
in if ConfMeth \=nil then raise illegalInhe end end
   if ConfAttr \=nil then raise illegalInhe end end
   c(methods:{Adjoin {Adjoin M2 M3} M1}
     attrs:{Union {Union A2 A3} A1})
end
```

La siguiente función crea un instancia de una clase. Esta función es independiente de que la clase haya sido construida con herencia.

```
fun {New Class InitialMethod}
  State Obj
  in State={MakeRecord s Class.attrs}
     {Record.forAll State proc {\S A } {NewCell _ A} end }
     proc {Obj M}
        {Class.methods.{Label M} M State Obj}
     end
    {Obj InitialMethod}
    Obj
end
```

Claramente la definición de herencia definida en From C1 C2 C3 funciona solamente si C1 hereda de dos clases.

a) (1 punto) Redefina From para que tome herencia Multiple

b) (2 puntos) Redefina la definicion de clase para que soporte "static binding" (hint: redefinir la estructura de contiene una definición de clase).

3. (2 puntos) Dado el siguiente par de hilos, determinar cuántos *interleaving* posibles existen.
   Dar la respuesta en función de N y M, las cuales solo denotan constantes numéricas distintas.

```
declare X1 X2 ... Xn Y1 Y2 ... Ym in
thread X1=1 X2=2 ... Xn=N end
thread Y1=1 Y2=2 ... Ym=M end
```

4. (3 puntos) Dadas las siguientes declaraciones en pseudo-codigo:

```
int a[2] = {1,0}; // arreglo
int i = 0;

swap(int x, int y) {
    int z;
    z = x;
    x = y;
    y = z;
}

int f(int a, int b) {
    a = 0;
    b = b+1;
    return b;
}
```

Justificar en cada caso.

a) Dar los valores finales para `i` y `a[i]`; si se realiza la llamada `swap(i,a[i])` por *nombre*.

b) Dar el valor final para `a[i]`; si se realiza la llamada `f(a[i],a[i])` por *referencia*.

c) Dar el valor final para `a[i]`; si se realiza la llamada `f(a[i],a[i])` por *resultado-valor*.