Paradigmas de la Programación
Primer Parcial

Gabriel Infante Lopez
Laura Alonso Alemany
Ezequiel Orbe
Edgardo Hames

28 de Abril de 2011

Apellido y Nombre: _____

Instrucciones:

- Coloque nombre y nro de página en todas las hojas.
- Resuelva cada uno de los ejercicios en hojas distintas.
- Lea todos los ejercicios antes de iniciar la resolución del exámen.
- Sea conciso y claro. Se quitarán puntos cuando las respuestas sean confusas o irrelevantes.
- **El exámen se aprueba con 4 (50 %).**
- Buena Suerte!

Ejercicio	Pts. Otorgados	Pts. Obtenidos
1	20	
2	13	
3	34	
4	20	
5	13	
TOTAL	100	
NOTA		
REGULAR	SI	NO
PROMOCION	SI	NO

1. *(20 Puntos)*

   a) *(10 Puntos)* Defina un ADT para árboles binarios que sea protegido.

   b) *(10 Puntos)* Implemente una función {Height T} recursiva a la cola que tome como argumento un árbol binario y retorne su altura.

2. *(13 Puntos)* En los puntos indicandos con <----, indique cuáles son las variables que pueden ser recolectadas si el recolector de basura (garbage collector) se ejecuta.

```
fun {Mergesort Xs}
 case Xs
      of nil then nil
         [] [X] then [X]
       else Ys Zs in
           {Split Xs Ys Zs}
                        <----
           {Merge {Mergesort Ys}
                      {Mergesort Zs}}
  end
end

Z = [1 2 3 4 5]
XS = {Mergesort  1 | 2 | 3 | Z }
{Browse Z}
         <----
{Browsw XS}
         <----
```

3. *(34 Ptos)* Haskell posee una construcción llamada *list comprehension*, la cual provee una forma concisa de crear listas. Cada *list comprehension* consiste de una expresión seguida de una barra vertical (|), y luego, una secuencia no vacía de

   - generadores de la forma v <- e, donde v es una variable de tipo T y e es una expresión de tipo $[T]$, o
   - expresiones arbitrarias de tipo booleano.

   El resultado de una *list comprehension* será la lista resultante de evaluar la expresión en el contexto de los entornos creados por la evaluación de cada uno de los generadores.

   A modo de ejemplo, el siguiente fragmento de código muestra la utilización de esta construcción en Haskell:

```
Hugs> [ 3 * x |  x <- [1,2,3] ]
[3,6,9]
Hugs> [ 3 * x |  x <- [1,2,3,4], x > 2 ]
[9,12]
```

   Agregue al lenguaje Oz el soporte sintáctico para *list comprehensions* e implemente la semántica de esta construcción a través de su traducción en función de las construcciones existentes en el lenguaje de kernel. La traducción que realice debe ser lo más eficiente posible.

4. *(20 Ptos)* Las siguientes funciones escritas en ML poseen idéntico comportamiento computacional,

```
fun f(x) = not (f(x));

fun g(y) = g(y) * 2;
```

dado que podemos reemplazar una función por otra en cualquier programa sin cambiar el comportamiento observable del mismo.

a) ¿Cuál es el tipo de `f(x)`?

b) ¿Cuál es el tipo de `g(y)`?

c) Explique porque ambas funciones tienen el mismo comportamiento computacional en tiempo de ejecución.

5. ***(13 Puntos)*** Considere el siguiente fragmento de código:

```
local X = 2 in
   local fun{F Y} X + Y end in
      local X = 7 in
         local Z = X + {F X} in
            {Browse Z}
         end
      end
   end
end
```

a) Traduzca a lenguaje de kernel.

b) Considerando *scoping estático*, ¿ cuál es el valor de Z?

c) Considerando *scoping dinámico*, ¿cuál es el valor de Z?