Paradigmas de Programación

Práctica 2

1. **Curry y uncurry**. Dada una función $f : X \times Y \to Z$, podemos siempre considerar una función $g : X \to (Y \to Z)$ tal que $f(x, y) = (g\,x)\,y$.

   A esta transformación se le denomina "currificación" (*currying*) y decimos que la función g es la forma "currificada" de la función f (y que la función f es la forma "descurrificada" de la función g). A la transformación inversa se le denomina "descurrificación" (*uncurrying*).

   Defina en OCaml una función

   ```
   curry : (('a * 'b) -> 'c) -> ('a -> ('b -> 'c))
   ```

   de forma que para cualquier función `f` cuyo origen sea el producto cartesiano de dos tipos, `curry f` sea la forma currificada de `f`.

   Y defina también la función inversa

   ```
   uncurry : ('a -> ('b -> 'c)) -> (('a * 'b) -> 'c)
   ```

   Una vez definidas estas dos funciones, prediga y compruebe (como en la práctica 1) el resultado de compilar y ejecutar las siguientes frases en OCaml:

   ```
   uncurry (+);;

   let sum = (uncurry (+));;

   sum 1;;

   sum (2,1);;

   let g = curry (function p -> 2 * fst p + 3 * snd p);;

   g (2,5);;

   let h = g 2;;

   h 1, h 2, h 3;;
   ```

   Realice todas estas tareas en el fichero de texto `curry.ml`.

2. **Composición**. Defina en OCaml la forma currificada de la composición de funciones:

   ```
   comp : ('a -> 'b) -> ('c -> 'a) -> ('c -> 'b)
   ```

   Una vez definida esta función, prediga y compruebe (como en la práctica 1) el resultado de compilar y ejecutar las siguientes frases en OCaml:

   ```
   let f = let square x = x * x in comp square ((+) 1);;

   f 1, f 2, f 3;;
   ```

   Realice todas estas tareas en el fichero de texto `comp.ml`.

3. **Ejercicio opcional**. Como sabemos, una expresión que contenga una definición local, de la forma

   ```
   let <x> = <eL> in <eG>
   ```

   puede siempre reescribirse, sin definiciones locales, utilizando la aplicación de funciones, como la expresión equivalente

   ```
   (function <x> -> <eG>) <eL>
   ```

   Reescriba el siguiente fragmento de código OCaml, de modo que no se empleen definiciones locales:

   ```
   let e1 =
     let pi = 2. *. asin 1. in pi *. (pi +. 1.);;

   let e2 =
     let lg2 = log 2. in
     let log2 = function x -> log x /. lg2
     in log2 (float (1024 * 1024));;

   let e3 =
     let pi_2 = 4. *. asin 1. in
     function r -> pi_2 *. r;;

   let e4 =
     let sqr = function x -> x *. x in
     let pi = 2. *. asin 1. in
     function r -> pi *. sqr r;;
   ```

   Realice todas estas tareas en el fichero de texto `ej23.ml`.

NOTA: Cuando se solicite la entrega de esta práctica, deberá enviar únicamente los ficheros `curry.ml`, `comp.ml` y `ej23.ml`. Sea muy cuidadoso a la hora de crear los ficheros, y **respete los nombres indicados**. En particular, fíjese que todos estos nombres sólo contienen letras en minúsculas, números y puntos. Además, **todos los ficheros deben compilar**.