Paradigmas de Programación

Práctica 8

Ejercicios:

1. Una de las operaciones más frecuentes con listas es la iteración a través de todos sus elementos para actualizar, con cada uno de ellos, el valor de un acumulador. Puede utilizarse, por ejemplo, para contar su número de elementos (incrementando el valor del acumulador una unidad por cada elemento), para sumar sus elementos (añadiendo al acumulador el valor de cada uno de ellos), o para obtener su valor máximo (substituyendo en cada paso el valor del acumulador por el del elemento, si éste es mayor). Y, naturalmente, para realizar muchas más operaciones.

   La función `fold_left`, definida en el módulo `List` de OCaml y una de las más útiles de este módulo, supone una abstracción que generaliza todas esas operaciones. Para realizar el proceso, aparte de la lista de valores, se necesita conocer la operación que actualiza, en cada paso, el valor del acumulador a partir del valor de cada elemento de la lista, y el valor inicial del acumulador. Eso es lo que hay que proporcionar a `fold_left` en sus argumentos al aplicarla. El tipo polimórfico de `fold_left`

   ```
   fold_left : ('a -> 'b -> 'a) -> 'a -> 'b list -> 'a
   ```

   indica que el primer argumento (cuyo tipo es `'a -> 'b -> 'a`) representa la operación que, a partir de un acumulador de tipo `'a` y uno de los elementos de la lista (que serán de tipo `'b`), calcula el nuevo valor del acumulador. Por eso, el segundo argumento, que representa el valor inicial del acumulador, debe ser de tipo `'a`, y la lista de valores, sobre los que se quiere iterar, de tipo `'b list`. El resultado de la operación (esto es, el valor del acumulador después de iterar sobre todos los elementos de lista) tiene que ser, claro, de tipo `'a`.

   Si ya ha implementado esta función en la práctica anterior, debería haber comprobado que su definición resulta recursiva terminal de modo natural. Así lo es también la del módulo `List`. Esto aporta la ventaja adicional de que su uso no supone el riesgo de provocar un error por agotamiento de la pila de recursividad (*stack overflow*).

   Analice hasta su total comprensión los siguientes ejemplos, donde se utiliza `fold_left` para definir, de modo recursivo terminal, las funciones que dan el número de elementos de una lista, la suma de sus valores (para una lista de enteros), y el máximo elemento de una lista:

   ```
   let length l = List.fold_left (fun n _ -> succ n) 0 l

   let sum l = List.fold_left (+) 0 l

   let lmax = function
       [] -> raise (Failure "lmax")
     | h::t -> List.fold_left max h t
   ```

   Defina ahora usted utilizando la función `fold_left` del módulo `List`, las funciones descritas en el archivo `folding.mli` proporcionado. En este ejercicio no está permitido utilizar ninguna otra de las funciones del módulo `List` (aparte de `fold_left`). Tampoco se permite usar ninguna de las funciones que hay que definir, en la definición de las otras, ni la palabra reservada `rec`. Las definiciones de este ejercicio deben guardarse en un archivo con nombre `folding.ml`.

2. Aparte de la función `fold_left`, el módulo `List` contiene otras funciones muy útiles (como `init`, `map`, `filter`, etc.). Las mencionadas están implementadas, además, de modo recursivo terminal[1].

   Defina las funciones descritas en el archivo `listing.mli`. Ahora sí puede utilizar las funciones del módulo `List`. De hecho, se pide que intente utilizarlas siempre que esto pueda simplificar la definición en cuestión.

   Sólo puede utilizarse recursividad terminal.

   Las funciones que vaya definiendo pueden utilizarse (si conviene) en las definiciones de las siguientes. Las definiciones de este ejercicio deben guardarse en un archivo con nombre `listing.ml`.

[1] Tenga en cuenta que `List.map` (como `List.append`) sólo está implementada de modo recursivo terminal desde la versión 5.1 de OCaml. Si usted tiene instalada una versión anterior, tenga en cuenta que su uso sí puede provocar (en su equipo) un error de *stack overflow*. En cualquier caso, puede usar `List.map` en este ejercicio como si fuese recursiva terminal.