Proyecto 3

Algoritmos y Estructuras de Datos II
Laboratorio

17 de septiembre de 2006

El proyecto consta de cuatro ejercicios (el último es el más divertido). Los ejercicios no se pueden especificar en el formalismo del teórico pero pueden ser resueltos aplicando el razonamiento inductivo como se hizo en el ejercicio de las torres de Hanoi.

1. Programar la función

```
intercal :: a -> [a] -> [[a]]
```

   donde `intercal x ys` devuelve todas las posibles intercalaciones de `x` en la lista `ys`. Por ejemplo:

```
intercal 100 [6,3] = [[100,6,3],[6,100,3],[6,3,100]]
intercal 1 [3,4,5] = [[1,3,4,5],[3,1,4,5],[3,4,1,5],
                      [3,4,5,1]]
intercal 'u' "clo" = ["uclo","culo","cluo","clou"]
intercal 3 [] = [[3]]
```

   Ayuda. Pensar inducción sobre la lista. O sea, ver que devuelve la función para el caso en que la lista sea vacía y para el caso en que no lo sea. Para el caso no vacía es posible que necesite crear una función auxiliar. Para ella aplique también el razonamiento inductivo.

2. Una función muy útil que está definida en el prelude de Haskell es

$$\texttt{map} :: (\texttt{a} \mapsto \texttt{b}) \mapsto [\texttt{a}] \mapsto [\texttt{b}]$$

   Esta toma un función y una lista, y devuelve la lista original pero aplicando la función a cada elemento. O sea,

$$\texttt{map f}\ [\texttt{a}_1, \texttt{a}_2, \cdots, \texttt{a}_\texttt{n}] = [\texttt{f}\ \texttt{a}_1, \texttt{f}\ \texttt{a}_2, \cdots, \texttt{f}\ \texttt{a}_\texttt{n}]$$

   Con esta función programar **en una sola linea** las funciones:

   a) `sumarList :: Num a => a -> [a] -> [a]` que toma un número y una lista de números y le suma a cada elemento de la lista el primer parámetro. Por ejemplo:

```
sumarList 3 [4,6,7] = [7,9,10]
```

Ayuda. En Haskell se puede definir la función que toma un número y le suma un numero fijo x como `(x+)` (esto se puede hacer con cualquier operador binario). Ver el tipo de esta construcción en Hugs con el comando.

```
> :t (2+)
```

b) `encab :: a -> [[a]] -> [[a]]` que toma un elemento y lo pone en la cabeza de cada elemento de las listas del segundo parámetro. Por ejemplo:

```
encab 3 [[2,1],[],[4,7]] = [[3,2,1],[3],[3,4,7]]
```

Ayuda. La misma ayuda sirve para este ejercicio: en Haskell se puede definir `(x:)` como la función que toma una lista y le agrega `x` al principio ya que el constructor de listas '`:`' es un operador binario. Probar en Hugs

```
> :t (4:)
```

3. Programar la función `intercal` del ejercicio 1, utilizando la función `map` y sin utilizar funciones auxiliares.

   Ayuda. Pensar inducción sobre la lista igual que en el ejercicio 1. Para el caso base (lista vacía) es lo mismo. Para el caso inductivo ver si se puede usar `map`.

4. Don Cambiazo es el quiosquero del barrio. Es un hombre muy dedicado a la actividad comercial que realiza ya que la misma se ha convertido no solo en su medio de subsistencia si no en una obsesión. Todos los días (incluidos sábados, domingos y feriados) se levanta dos horas antes de abrir el quisco, se hace unos mates, prende LV3 y se dispone a meditar distintas maneras de mejorar la forma en que desarrolla su actividad laboral. La problemática que suele analizar en estos momentos de autocontemplación es diversa y a menudo se centra en como hacer para tener siempre monedas para dar vuelto.

   Una mañana, Don Cambiazo se despertó algo incomodo. El pobre hombre había dormido mal ya que tuvo una pesadilla recurrente toda la noche. En su pesadilla él estaba muy cómodo atendiendo el quiosco, cuando de pronto un nuevo cliente entró y le pidió un caramelo Media Hora. Con la confianza de saber que su quiosco tiene de todo, incluido esos horribles caramelos, saco sin mirar uno de ellos de una cajita del mostrador y se lo dio al cliente, al mismo tiempo que decía "son 34 centavos". El cliente le acercó una moneda de un peso, y él, de forma rutinaria, abrió la caja de las monedas para darle el vuelto. La caja tenía tres compartimentos con monedas de 5, 2 y 1 centavos cada uno. No solo eso, si no que cada compartimento contenía infinitas monedadas de cada valor. Esto le produjo mucho placer, ya que la disposición infinita de cambio era una de sus fantasías más codiciadas. Pero resultó en el sueño, que al momento de devolver el vuelto, él se quedaba inmóvil. Por más que intentaba calcular cuantas monedas de cada valor tenía que dar al cliente no lo podía hacer. Esta era una tarea rutinaria que Don Cambiazo la hacía casi sin pensar. Pero en su pesadilla no lo podía hacer. En ese momento el cliente se empezaba a reír, y el despertaba de forma violenta y exaltada.

Esa mañana, traumado por su pesadilla, Don Cambiazo no prendió la radio y se puso a pensar como solucionar el problema del cambio. El sabía que su quiosco siempre tenía un montón de monedas de cada valor (infinitas), pero le preocupaba profundamente no poder algún día calcular el cambio. A partir de esto, decidió entonces comprarse una computadora y hacer él mismo un programa que resuelva el problema. Mucha idea de programación Don Cambiazo no tenía así que, asesorado por un cliente alumno del FaMAF, se puso a leer el tutorial de Haskell y unos apuntes sobre inducción. Después de una semana de intenso estudio y algunas pruebas llegó a un programa que hacía algo parecido a lo que Don Cambiazo hacía manualmente. Al programa había que darle como parámetro una lista con los distintos valores disponibles de monedas en orden decreciente y la cantidad que se desea convertir en cambio. La función devuelve una lista de las monedas disponibles que sumadas dan el valor a cambiar. La primera versión del programa era la siguiente:

```
cambio :: Integral a => [a] -> a -> [a]
cambio [] 0 = []
cambio [] (v+1) = error "No hay cambio"
cambio (m:ms) v | m <= v = m: cambio  (m:ms) (v-m)
                | m > v = cambio ms v
```

Según Don Cambiazzo, el programa funciona así: hago inducción sobre la listas de valores de monedas para cambio. Si no tengo ninguna y el valor a convertir es 0, devuelvo la lista vacía (cambio de \$0 es ninguna moneda). Si el valor a convertir es distinto de cero, devuelvo error (ya que no tengo cambio). Para el caso inductivo, si la moneda más grande es menor o igual que el valor, puedo dar esta moneda como cambio y volver a hacer todo el proceso con el valor a cambiar menos la moneda que dí. Si el valor de la moneda es mayor que el valor a cambiar, esa moneda no se puede usar para dar el cambio, entonces repito todo el proceso sin este valor de moneda (recordar que la lista de valores de monedas esta en orden decreciente).

Muy contento con su programa (no solo calcula el cambio si no que siempre lo devuelve con las monedas más grandes!!) Don Cambiazzo se fue a dormir, confiado que no iba a volver a tener esa pesadilla otra vez. Sin embargo no fue así. Esa noche tuvo el mismo sueño pero en la caja había monedas de 5 y 2 centavos únicamente. Al igual que en el otro sueño no pudo calcular el cambio. Pero esta vez el cliente en vez de reírse se fue enojado.

Inmediatamente después de despertarse prendió la computadora y probó reproducir su pesadilla en el Hugs:

```
Main> cambio [5,2] (100-34)
[5,5,5,5,5,5,5,5,5,5,5,5,5
  Program error: No hay cambio
```

Totalmente deprimido, Don Cambiazzo llegó tarde a atender el quiosco por primera vez en su vida. Al llegar estaba esperándole el cliente alumno del FaMAF para comprarle un caramelo Media Hora. Inmediatamente y antes de abrir la persiana del negocio le comentó su desventura con el Haskell. El cliente, después de ver el programa de Don Cambiazzo,

le sugirió que pruebe hacer un programa que devuelva todas las soluciones posibles y que después tome la primera solución. Esto es:

```
cambio :: Integral a => [a] -> a -> [a]
cambio ms v = head  (posCambio ms v)

posCambio :: Integral a => [a] -> a -> [[a]]
posCambio [] 0     = [[]]
posCambio [] (v+1) = []
...
```

Ejercicio: Podría usted usted ayudar a Don Cambiazzo y completar el programa?

Don Cambiazzo, sorprendido por el consejo, atinó a preguntar porqué el programa solo toma la primer solución. Teniendo todas las soluciones en la pantalla podría elegir a mano la que más le guste y el costo del cálculo sería el mismo, ya que el programa aconsejado termina calculando todas las soluciones. Su cliente le respondió que si no lo hacía seguramente tendría que comprarse una computadora más grande.

Preguntas: ¿Es correcto el consejo del cliente? ¿Por qué?