PLP - Primer Parcial - 2^{do} cuatrimestre de 2015

Este examen se aprueba obteniendo al menos **65 puntos** en total, y al menos **5 puntos** por cada tema. Poner nombre, apellido y número de orden en cada hoja, y numerarlas. Se puede utilizar todo lo definido en las prácticas y todo lo que se dio en clase, colocando referencias claras.

Programación funcional

Ejercicio 1 - (45 puntos)

Durante este ejercicio **no** se puede usar recursión explícita, a menos que se indique lo contrario. Para resolver un ítem pueden utilizarse las funciones definidas en los items anteriores, más allá de si fueron resueltos correctamente o no. Dar el tipo de todas las funciones pedidas. Puede utilizar listas por comprensión.

Considerar la siguiente definición alternativa para listas:

```
data ConcList a = Nil | Singleton a | Append (ConcList a) (ConcList a)
```

En vez de obligar a trabajar siempre sobre la cabeza y la cola de una lista, esta estructura permite deconstruir la lista de manera más flexible. Por ejemplo, puede ser conveniente para implementar algoritmos de tipo *divide and conquer* en donde se busca operar recursivamente sobre partes de la lista, o para paralelizar el procesamiento de los elementos de la misma.

a. Definir `foldCL`, el esquema de recursión estructural para el tipo `ConcList`. **Puede utilizar recursión explícita para este punto**.

b. Definir la función `longitut` que permita calcular el tamaño de una `ConcList`.

c. Definir la función `duplicarApariciones` que convierta una `ConcList` en otra donde cada elemento es reemplazado por dos apariciones consecutivas del mismo. Por ejemplo:

   duplicarApariciones$([1, 2, 3]) \rightarrow [1, 1, 2, 2, 3, 3]$

d. Considere la siguiente función, que construye una `ConcList` balanceada a partir de una lista tradicional:

```
toConcList :: [a] -> ConcList a
toConcList []     = Nil
toConcList (x:xs) = if null xs
                    then
                      Singleton x
                    else
                      let
                        (l1, l2) = split (x:xs)
                      in
                        Append (toConcList l1) (toConcList l2)
```

   Argumentar qué complicación hubiese traído utilizar `foldr` para definir la función anterior.

e. Definir la función `consecutivos :: [ConcList Int]`, que genere una lista (infinita) de todas las `ConcList` de números consecutivos, sin importar el orden. Por ejemplo:

   take$(10,$ consecutivos$) \rightarrow [[0],\ [0, 1],\ [1],\ [0, 1, 2],\ [1, 2],\ [2],\ [0, 1, 2, 3],\ [1, 2, 3],\ [2, 3],\ [3]]$

Cálculo lambda

En esta sección trabajaremos sobre el cálculo lambda con listas. Partimos de los siguiente conjuntos de términos, tipos y valores:

$$M ::= \ \ldots \mid [\ ]_\sigma \mid M :: N \qquad \sigma ::= \ \ldots \mid [\sigma] \qquad V ::= \ \ldots \mid [\ ]_\sigma \mid V_1 :: V_2$$

Las reglas de tipado definidas para listas son las siguientes:

$$\frac{}{\Gamma \rhd [\ ]_\sigma : [\sigma]}\ (\text{T-LIST-NIL}) \qquad \frac{\Gamma \rhd M : \sigma \quad \Gamma \rhd N : [\sigma]}{\Gamma \rhd M :: N : [\sigma]}\ (\text{T-LIST-CONS})$$

Ejercicio 2 - Extensiones (25 puntos)
Se desea extender el cálculo anterior para poder expresar listas de naturales a través de intervalos. Se quiere que, por ejemplo, la expresión $[\underline{1} \mathinner{..} \underline{3}]$ evalúe a la lista $\underline{1} :: \underline{2} :: \underline{3} :: [\ ]_{\mathsf{Nat}}$. Los intervalos cuyo límite izquierdo sea mayor al derecho denotarán la lista vacía.

a. Indicar formalmente cómo se modifica el conjunto de términos y tipos (en caso de ser necesario) e introducir las reglas de tipado para la extensión propuesta.

b. Exhibir una derivación para el siguiente juicio de tipado. De no ser posible explicar el problema.
$\{n : \mathsf{Nat}\} \rhd (\lambda x : \mathsf{Nat}.\ [\mathsf{succ}(x) \mathinner{..} n])\ \mathsf{succ}(0) : [\mathsf{Nat}]$

c. Indicar formalmente cómo se modifica el conjunto de valores, y dar la semántica operacional de a un paso para la extensión propuesta.

d. Mostrar cómo reducen paso por paso los siguientes términos:

- $[\underline{1} \mathinner{..} \underline{3}]$
- $(\lambda x : \mathsf{Nat}.\ [\mathsf{succ}(x) \mathinner{..} x])\ 0$

Ejercicio 3 - Inferencia de tipos (30 puntos)
Queremos además modificar el cálculo para incorporar la posibilidad de utilizar listas por comprensión. Formalmente, se modificará el conjunto de términos de la siguiente manera:

$$M ::= \ \ldots \mid [\!|\ M_1 \ |\ x \ \leftarrow \ M_2 \ |\!]$$

Donde M_2 será una lista de elementos a recorrer, x es una variable que representa un valor de la lista, empezando desde el primer elemento, y M_1 representa a los valores de la lista resultante. Notar que la variable x puede aparecer libre en M_1. Por ejemplo:
$[\!|\ \mathsf{succ}(x) \ |\ x \ \leftarrow \ (\underline{1} :: \underline{2} :: [\ ]_{\mathsf{Nat}}) \ |\!]$ reduce en **varios** pasos a $\underline{2} :: \underline{3} :: [\ ]_{\mathsf{Nat}}$.

Para lograr esta funcionalidad, se agrega la siguiente regla de tipado:

$$\frac{\Gamma \cup \{x : \tau\} \rhd M_1 : \sigma \qquad \Gamma \rhd M_2 : [\tau]}{\Gamma \rhd [\!|\ M_1 \ |\ x \ \leftarrow \ M_2 \ |\!] : [\sigma]}\ (\text{T-LIST-COMP})$$

a. Extender el algoritmo de inferencia para que soporte listas y la nueva construcción propuesta. No es necesario que soporte la extensión del ejercicio 2.

b. Aplicar el algoritmo extendido para tipar la siguiente expresión, o demostrar que no tipa, explicitando las sustituciones realizadas en cada paso :

$$[\!|\ (\lambda y.\ y\ x\ )\ |\ x \ \leftarrow \ 0 :: z :: [\ ]\ |\!]$$