PLP - Recuperatorio del Primer Parcial - 1er cuatrimestre de 2017

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.

Ejercicio 1 - Programación Funcional (40 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 ítems anteriores, más allá de si fueron resueltos correctamente o no. Dar el tipo de todas las funciones implementadas.

El esquema de recursión primitiva permite definir funciones recursivas a partir de, no solo el resultado recursivo (como en el caso de `fold`), sino también de los elementos originales que componen la estructura sobre la que se aplican.
Por ejemplo, para las listas con los constructores usuales, una función definida por recursión primitiva es de la forma

$$f([\,]) = z \qquad f(x : xs) = g(x, xs, f(xs))$$

Para capturar este esquema en Haskell, definimos la función `recr` de la siguiente manera:

```
recr::(a->[a]->b->b)->b->[a]->b
recr \_ z [] = z
recr g z (x:xs) = g x xs rec
    where rec = recr g z xs
```

En este ejercicio trabajaremos sobre árboles binarios, según la definición presentada durante la cursada [1]:
`data AB a = Nil | Bin (AB a) a (AB a)`
Dado que veremos varios tipos de esquemas recursivos, además de la corrección de la solución, se evaluará la pertinencia de los esquemas de recursión utilizados (por este motivo, si se utilizan funciones definidas durante la cursada, se deberán incluir estas definiciones).

a) Esquemas de Recursión

   I. Dar el tipo y definir `recrAB`, el esquema de recursión primitiva para el tipo `AB a`. En este ítem, **y solo en este**, está permitido el uso de recursión explícita.

   II. Dar el tipo y definir `foldAB`, el esquema de recursión estructural para el tipo `AB a`, usando `recrAB`.

b) Funciones sobre árboles binarios. En los siguientes ítems se pide implementar funciones recursivas sobre árboles binarios.

   I. Definir `maxAB :: AB Int -> Int`, que devuelva el máximo elemento de un árbol binario de enteros positivos. Si este máximo no existe, debe devolver -1.

   II. Definir `maxAB1 :: Ord a => AB a => a`, que devuelva el máximo de un árbol binario, suponiendo que este no es vacío.

   III. Definir `esABB :: Ord a => AB a -> Bool`, que decide si un árbol binario tiene invariante de árbol binario de búsqueda. Recordemos que en un árbol binario de búsqueda, el subárbol izquierdo de cualquier nodo (si no está vacío) contiene valores menores que el que contiene dicho nodo, y el subárbol derecho (si no está vacío) contiene valores mayores.

   IV. Definir `insertarABB :: Ord a => AB a -> a -> AB a`, que inserta un elemento en un árbol con invariante de ABB, manteniendo el invariante. Recordemos que un elemento debe insertarse en el último lugar que revisaría una búsqueda infructuosa.

[1] En este ejercicio podrá utilizar, en caso de creer conveniente, la función `error: [Char] -> a`.

Ejercicio 2 - Cálculo Lambda Tipado (35 puntos)

Se desea extender el cálculo lambda tipado para soportar *diccionarios* con historia. Modificaremos el conjunto de tipos y términos de la siguiente manera:

$$\sigma ::= \cdots \mid \text{Dicc}_{\sigma,\tau}$$
$$M ::= \cdots \mid \text{Vacío}_{\sigma,\tau}(M) \mid M[N] \leftarrow O \mid \text{fold}_D\ M\ \text{base}(def) = N\,;\ \text{recursión}(k,\ val,\ rec) = O$$

Un diccionario es una estructura de datos que almacena valores, todos del mismo tipo, en función de una clave (no necesariamente del mismo tipo que los valores). Para esto contaremos con el tipo $\text{Dicc}_{\sigma,\tau}$ en donde σ es el tipo de la clave y τ es el tipo de los valores. El término $\text{Vacío}_{\sigma,\tau}(M)$ construye un diccionario vacío en el cual M es el valor por defecto que tendrá el diccionario. Con la expresión $M[N] \leftarrow O$ agregamos un nuevo valor O en el diccionario M para la clave N. Adicionalmente, introducimos el esquema de recursión estructural fold_D que nos permite obtener la historia del diccionario recorriendo su estructura.
En la expresión $\text{fold}_D\ M\ \text{base}(def) = N\,;\ \text{recursión}(k,\ val,\ rec) = O$, M es el diccionario sobre la cual se realiza la recursión, N es lo que se debe retornar como caso base y O es el paso recursivo. La variable def aparece libre en la expresión N y deberá ligarse con el valor por defecto del diccionario. De forma similar, las variables k, val y rec aparecen libres en O y representarán respectivamente la clave, el valor definido para esta clave y el resultado recursivo.

a) Introducir las reglas de tipado para la extensión propuesta.

b) Definir el conjunto de valores e introducir las reglas de semántica para esta extensión.

c) Escribir la función $valor_{\sigma,\tau}$ como **macro**, la cual debe tomar una $\text{Dicc}_{\sigma,\tau}$, un término de tipo σ (que servirá como clave) y debe devolver el valor más reciente sobre la clave pasada, o el valor por defecto en caso de que no haya asignación. Asumir que está definida la igualdad para el tipo σ ($=_\sigma$).

Ejercicio 3 - Inferencia de tipos (25 puntos)

En este ejercicio extenderemos el algoritmo de inferencia para soportar el tipado de naves espaciales como las vistas en el TP de Programación Funcional. Para expresar esta extensión en Cálculo Lambda, los tipos y términos se extienden de la siguiente manera:

$$\begin{aligned} \sigma ::=\ & \ldots \mid \texttt{Componente} \mid \texttt{NaveEspacial} \\ M, N, O ::=\ & \ldots \mid \texttt{Contenedor} \mid \texttt{Motor} \mid \texttt{Escudo} \mid \texttt{Cañón} \\ & \mid \texttt{Base}(M) \mid \texttt{Módulo}(M, N, O) \mid \texttt{Case}\ M\ \texttt{Base}\langle w\rangle \rightsquigarrow N;\ \texttt{Módulo}\langle x, y, z\rangle \rightsquigarrow O \end{aligned}$$

Las reglas de tipado son las siguientes:

$$\frac{}{\Gamma \triangleright \texttt{Contenedor} : \texttt{Componente}} \quad \frac{}{\Gamma \triangleright \texttt{Motor} : \texttt{Componente}} \quad \frac{}{\Gamma \triangleright \texttt{Escudo} : \texttt{Componente}} \quad \frac{}{\Gamma \triangleright \texttt{Cañón} : \texttt{Componente}}$$

$$\frac{\Gamma \triangleright M : \texttt{Componente}}{\Gamma \triangleright \texttt{Base}(M) : \texttt{NaveEspacial}} \qquad \frac{\Gamma \triangleright M : \texttt{Componente} \quad \Gamma \triangleright N : \texttt{NaveEspacial} \quad \Gamma \triangleright O : \texttt{NaveEspacial}}{\Gamma \triangleright \texttt{Módulo}(M, N, O) : \texttt{NaveEspacial}}$$

$$\frac{\Gamma \triangleright M : \texttt{NaveEspacial} \quad \Gamma \cup \{w : \texttt{Componente}\} \triangleright N : \sigma \quad \Gamma \cup \{x : \texttt{Componente}, y : \texttt{NaveEspacial}, z : \texttt{NaveEspacial}\} \triangleright O : \sigma}{\Gamma \triangleright \texttt{Case}\ M\ \texttt{Base}\langle w\rangle \rightsquigarrow N;\ \texttt{Módulo}\langle x, y, z\rangle \rightsquigarrow O : \sigma}$$

a) Dar la extensión al algoritmo necesaria para soportar el tipado de las naves espaciales y el `Case` correspondiente. Asumir que el algoritmo ya fue extendido para tipar correctamente los componentes.

b) Aplicar el algoritmo extendido a la siguiente expresión, explicitando las sustituciones involucradas en cada paso. Dar el resultado obtenido o indicar dónde se produce un error.

$$\texttt{Case}\ \texttt{Base}(x)\ \texttt{Base}\langle y\rangle \rightsquigarrow \mathsf{True};\ \texttt{Módulo}\langle x, y, z\rangle \rightsquigarrow \mathsf{False}$$