PLP - Recuperatorio del Primer Parcial - 1er cuatrimestre de 2019

Este examen se aprueba obteniendo al menos dos ejercicios bien menos (B-) y uno regular (R). Las notas para cada ejercicio son: -, I, R, B-, B. Entregar cada ejercicio en hojas separadas. Poner nombre, apellido, número de orden y cantidad de hojas en la primera hoja, y numerar las hojas. Se puede utilizar todo lo definido en las prácticas y todo lo que se dio en clase, colocando referencias claras.

Ejercicio 1 - Programación funcional

En este ejercicio **no** se permite el uso de recursión explícita, a menos que se indique lo contrario. Pueden usar los ejercicios de la práctica o los vistos en clase, colocando referencias claras. Dar **el tipo** de todas las funciones definidas.

Se desea representar grafos dirigidos en Haskell mediante su lista de nodos y su función de adyacencia:
`data Grafo a = G [a] (a -> [a])`
La expresión `G ns f` representa un grafo cuyos nodos son los elementos de `ns` y, dado un nodo `n` perteneciente a `ns`, `f n` es la lista de vecinos de `n`. Es decir, los nodos hacia los cuales llegan los arcos que salen de `n`.
Cada grafo `G ns f` satisface el siguiente invariante: tanto `ns` como las listas que devuelve `f` son listas finitas sin elementos repetidos, y todos los elementos de las listas en la imagen de `f` pertenecen a `ns`. No hay ninguna restricción sobre el comportamiento de `f` si se la aplica a un nodo no perteneciente a `ns` (puede devolver una lista, colgarse o arrojar un error).
Implementar las siguientes funciones.

a. `arcos :: Grafo a -> [(a,a)]`, que dado un grafo enumera todos sus arcos.

b. `válido :: Grafo a -> Bool` que indica si la estructura dada representa un grafo válido. Es decir, cumple el invariante de representación (se puede suponer que las listas son finitas).

c. `alcanzable :: Grafo a -> a -> a -> Bool`, que dados un grafo y dos nodos, indica si existe en el grafo un camino del primer nodo al segundo, pasando por al menos un arco. Tener en cuenta que puede haber ciclos. **En este punto se permite usar recursión explícita.**

d. `hayCiclo :: Grafo a -> Bool`, que devuelve `True` si y solo si el grafo dado tiene al menos un ciclo.

Ejercicio 2 - Cálculo Lambda Tipado

Se desea extender el Cálculo Lambda tipado para soportar intervalos de números naturales.
Se extenderán los tipos y términos de la siguiente manera:
$\sigma ::= \cdots \mid \mathsf{Intervalo} \qquad M ::= \cdots \mid [M, M] \mid \text{inicio}(M) \mid \text{fin}(M) \mid \text{case } M \text{ of } [\,] \rightsquigarrow M; x :: y \rightsquigarrow M$
donde $[M, N]$ es el intervalo que va desde el número M hasta N, $\text{inicio}(M)$ es el primer elemento del intervalo M, $\text{fin}(M)$ es el último, y $\text{case } M \text{ of } [\,] \rightsquigarrow N; x :: y \rightsquigarrow O$ es un observador que recorre el intervalo M como si fuese una lista.
Se considera que un intervalo $[M, N]$ es vacío si M es mayor que N, en cuyo caso el case ejecutará su primera rama. Si M es menor o igual que N, el case ejecutará su segunda rama, ligando la variable x a M y la variable y al resto del intervalo. (Si M es igual a N, estamos ante un intervalo con un elemento, por lo cual no es vacío).

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

b. Exhibir una derivación del siguiente juicio de tipado:
$\emptyset \triangleright \text{case } [0, \mathsf{Succ}(0)] \text{ of } [\,] \rightsquigarrow 0; x :: y \rightsquigarrow \text{inicio}(y) : \mathsf{Nat}$

c. Definir el conjunto de valores y las nuevas reglas de semántica. Pueden usar los conectivos booleanos de la guía. **No es necesario escribir las reglas de congruencia**, basta con indicar cuántas son.
Pista: no está definido el $>$ para el tipo Nat, pero sí puede usarse para comparar números en el metalenguaje (es decir, se puede comparar $n > n'$ pero no $M > N$ ni $\underline{n} > \underline{n'}$).

d. Mostrar paso por paso cómo reduce la expresión del punto b .

Ejercicio 3 - Inferencia de Tipos

En este ejercicio extenderemos el algoritmo de inferencia para incorporar el tipado de *colas*. El conjunto de tipos y términos es:

$\sigma ::= \cdots \mid \mathsf{Cola}_\sigma \qquad M ::= \cdots \mid \langle\rangle_\sigma \mid \mathsf{encolar}(M, M) \mid \mathsf{próximo}(M) \mid \mathsf{desencolar}(M) \mid \exists x \in M/M$ donde $\langle\rangle_\sigma$ es la cola vacía en la que se pueden encolar elementos de tipo σ; $\mathsf{encolar}(M_1, M_2)$ representa el agregado del elemento M_1 al **final** de la cola M_2; los observadores $\mathsf{próximo}(M_1)$ y $\mathsf{desencolar}(M_1)$ devuelven, respectivamente, el primer elemento de la cola (el primero que se encoló), y la cola sin el primer elemento (estos dos últimos solo tienen sentido si la cola no es vacía); y el observador $\exists x \in M_1/M_2$ indica si existe un elemento x en la cola M_1 que cumpla la condición M_2, donde M_2 es una expresión booleana que puede contener libre a la variable x.

La reglas de tipado son las siguientes:

$$\frac{}{\Gamma \triangleright \langle\rangle_\sigma : \mathsf{Cola}_\sigma} \qquad \frac{\Gamma \triangleright M : \mathsf{Cola}_\sigma \quad \Gamma \triangleright N : \sigma}{\Gamma \triangleright \mathsf{encolar}(N, M) : \mathsf{Cola}_\sigma} \qquad \frac{\Gamma \triangleright M : \mathsf{Cola}_\sigma}{\Gamma \triangleright \mathsf{próximo}(M) : \sigma} \qquad \frac{\Gamma \triangleright M : \mathsf{Cola}_\sigma}{\Gamma \triangleright \mathsf{desencolar}(M) : \mathsf{Cola}_\sigma}$$

$$\frac{\Gamma \triangleright M : \sigma \quad \Gamma, \{x : \sigma\} \triangleright N : \mathsf{Bool}}{\Gamma \triangleright \exists x \in M/N : \mathsf{Bool}}$$

a) Dar la extensión al algoritmo necesaria para soportar el tipado de la nueva estructura. (Se considera ya definida la extensión para expresiones de la forma $\mathsf{desencolar}(M)$).

b) Aplicar el algoritmo extendido para tipar la siguiente expresión:

$$\mathsf{if}\ \exists x \in \mathsf{encolar}(y, \langle\rangle)/\mathsf{isZero}(x)\ \mathsf{then}\ \mathsf{próximo}(x)\ \mathsf{else}\ 0$$