PLP - Autoevaluación - 2^{do} cuatrimestre de 2020

Esta autoevaluación se entrega en el Campus Virtual. Es de caracter individual y obligatoria.

- Comienzo: Jueves 08/10 (17:00 hs)
- Límite de entrega: Lunes 12/10 (23:59 hs).
- Resolución en conjunto: Jueves 15/10 (17:00 - 19:30 hs)
- Para considerar aprobada la autoevaluación, deberán entregar los ejercicios 1 y 2 completos. El ejercicio 3 es **opcional**, sin embargo recomendamos fuertemente resolverlo.
- Si quedasen dudas sobre sus soluciones, podrán preguntarlas durante el horario de consulta o enviar un email con sus dudas puntuales.

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:

```
type Nodo = Integer
data Grafo = G [Nodo] (Nodo -> [Nodo])
```

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).

Ejercicios a implementar:

a. `ejGrafo :: Grafo`, el grafo que contiene los nodos numerados del 1 al 9 y cuya función de adyacencia conecta a cada número par a la totalidad de nodos del grafo y los números impares no tienen conexiones de salida.

b. `nodoReflex :: Grafo -> Nodo -> Bool`, que dados un grafo y un nodo, indica si este último pertenece a su propia lista de vecinos.

   Por ejemplo `nodoReflex ejGrafo 6` $\rightarrow True$ y en cambio `nodoReflex ejGrafo 1` $\rightarrow False$

c. `extenderCamino :: Grafo -> [Nodo] -> [[Nodo]]`, que dado un grafo y un camino c, válido dentro del grafo, de longitud mayor o igual a 1, devuelve los caminos resultantes de extender a c con un único salto en el grafo. **Sugerencia:** usar la función `last`.

d. `caminosDeLong :: Grafo -> Int -> [[Nodo]]`, que dado un grafo y un número natural n enumera los caminos del grafo que pasan exactamente por n arcos (cuentan todos los pasos, aun si se pasa dos o más veces por el mismo nodo o arco). Los caminos que pasan por 0 arcos son las listas de un solo nodo. **Sugerencia:** usar `foldNat` (si les sirve para pensarlo mejor, pueden hacer primero un borrador con recursión explícita).

e. `hamiltoniano :: Grafo -> Bool` que indica si existe un camino que pase por todos los nodos del grafo, sin repetir nodos.

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

g. `caminos :: Grafo -> [[Nodo]]`, que enumera todos los caminos del grafo. Tener en cuenta que la cantidad de caminos puede ser finita o infinita, según haya o no ciclos.

Ejercicio 2 - Cálculo Lambda Tipado

Se desea extender el Cálculo Lambda de booleanos, naturales y funciones para soportar **matrices infinitas** con **inserción por nombre** (es decir que no se reducen los elementos al insertarlos). Estas matrices pueden observarse accediendo a cualquier posición mediante dos números naturales (fila y columna). Pueden también "modificarse" de manera similar, agregando un elemento en una posición, y obteniendo una matriz igual a la original excepto por el elemento contenido en dicha posición. Las matrices se inicializan con un término definido por defecto, el cual será el elemento contenido en las posiciones donde no se haya agregado nada. Los elementos de la matriz (incluso el definido por defecto) **no deben reducirse** al ser insertados, sino únicamente al ser observados.

El conjunto de tipos y el de los términos se extienden de la siguiente manera:

$$\sigma \ ::= \ ... \mid \mathsf{MatInf}_\sigma \quad M, N, O, P \ ::= \ ... \mid \mathsf{MAT}(M) \mid M[N][O] \leftarrow P \mid M[N][O] \mid \mathsf{map}(M, N)$$

Donde MatInf_σ es el tipo de las matrices con elementos de tipo σ, $M[N][O] \leftarrow P$ significa agregar el elemento P en la fila N y columna O de la matriz M, $M[N][O]$ representa el elemento en la fila N y columna O de la matriz M, y $\mathsf{MAT}(N)$ representa una martiz infinita con el elemento N en todas sus posiciones. La expresión $\mathsf{map}(M, N)$ representa la aplicación de la función N a todos los elementos de la matriz M (incluyendo el valor por defecto).

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

b. Demostrar el siguiente juicio de tipado o explicar por qué no es derivable:

   i) $\{i : Nat, m : \mathsf{MatInf}_{Nat}\} \rhd \mathsf{map}((m[i][i] \leftarrow 0), \lambda x\colon \mathsf{Nat}.isZero(x)) : \mathsf{MatInf}_{Bool}$

c. Indicar formalmente cómo se modifica el conjunto de valores o explicar por qué no se modifica.

d. Dar la semántica operacional de a un paso para la extensión propuesta. Recordar que se puede usar lógica para predicar sobre propiedades verificables en tiempo de compilación. Por ejemplo, pueden utilizar como premisa para una regla si dos términos son o no equivalentes sintácticamente: $M \equiv N$ (especificando que $\equiv$ refiere a la igualdad sintáctica); preguntarse por el tipo de una sub-expresión que se esté evaluando; entre otras.

e. Mostrar cómo reducen paso a paso las siguientes expresiones. Identificar las reglas utilizadas en cada paso:

   i) $\big(\mathsf{MAT}(\lambda y\colon \mathsf{Bool}.y)[\mathsf{succ}(0)][0] \leftarrow (\lambda x\colon \mathsf{Bool.true}))[0][\mathsf{succ}(0)]$

   ii) $((\lambda x : \mathsf{Bool}.\mathsf{MAT}(x)[0][0] \leftarrow False)\ \mathsf{true})[0][pred(succ(0))]$

   iii) $\mathsf{map}(\mathsf{MAT}(0)[0][0] \leftarrow succ(0), \lambda x\colon \mathsf{Nat}.isZero(x))[0][0]$

Ejercicio 3 - Subtipado (opcional)

Se extiende el Cálculo Lambda para soportar arreglos estáticos. Los arreglos estáticos son creados con un tamaño fijo y no pueden ser redimensionados. El conjunto de términos y tipos se modifica de la siguiente manera:

$$M ::= \cdots \mid \mathsf{new}_{\underline{n}}[M] \mid M[\underline{n}] \leftarrow N \mid M[\underline{n}] \qquad \sigma ::= \cdots \mid \sigma_n^*$$

Notar que σ_n^* indica que habrá un tipo distinto para cada arreglo de tipo σ y tamaño n.

En el momento de la creación ($\mathsf{new}_{\underline{n}}[M]$), además de indicarse el tamaño del arreglo se debe indicar el elemento *por defecto* M, que será el contenido de las posiciones aún no asignadas. Con las expresiones $M[\underline{n}] \leftarrow N$ y $M[\underline{n}]$ se asigna y observa una posición $\underline{n}$ respectivamente. Cuando se asigna una misma posición más de una vez la última es la que prevalece, y si se observa una posición no asignada se obtiene el valor *por defecto*. Las posiciones del arreglo se cuentan desde 0.

Las reglas de tipado son las siguientes:

$$\frac{\Gamma \triangleright M : \sigma}{\Gamma \triangleright \mathsf{new}_{\underline{n}}[M] : \sigma_n^*}\ (\text{T-New}) \qquad \frac{\Gamma \triangleright M : \sigma_n^* \quad \Gamma \triangleright N : \sigma \quad k < n}{\Gamma \triangleright M[\underline{k}] \leftarrow N : \sigma_n^*}\ (\text{T-Assign}) \qquad \frac{\Gamma \triangleright M : \sigma_n^* \quad k < n}{\Gamma \triangleright M[\underline{k}] : \sigma}\ (\text{T-Read})$$

a. Definir la(s) regla(s) de subtipado para arreglos estáticos. Justificar en términos del principio de sustitutividad (es decir, en función de las operaciones que se pueden realizar sobre ellos).

b. ¿Cuál o cuáles de las siguientes expresiones debería(n) tipar? Explicarlo y demostrar que tipa(n) utilizando las nuevas reglas.

   i) $(\lambda a : \mathsf{Nat}_{\underline{2}}^*.\mathsf{succ}(a[\underline{1}]))\ \mathsf{new}_{\underline{1}}[0]$

   ii) $(\lambda a : \mathsf{Int}_{\underline{2}}^*.\mathsf{succ}(a[\underline{1}]))\ \mathsf{new}_{\underline{2}}[2{,}5]$

   iii) $(\lambda a : \mathsf{Int}_{\underline{2}}^*.\mathsf{succ}(a[\underline{1}]))\ \mathsf{new}_{\underline{3}}[0]$

   Considerar definidas las siguientes reglas (además de las vistas en clase y las definidas arriba).

$$\frac{\Gamma \triangleright M : \mathsf{Int}}{\Gamma \triangleright \mathsf{succ}(M) : \mathsf{Int}}\ (\text{T-SuccInt}) \qquad \frac{}{\Gamma \triangleright 2{,}5 : \mathsf{Float}}\ (\text{T-2,5})$$