PLP - Primer Parcial - 2^{do} cuatrimestre de 2017

Este examen se aprueba obteniendo al menos dos ejercicios bien menos (`B-`) y uno regular (`R`). Las notas para cada ejercicio son: `M`, `R`, `B-`, `B`. 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

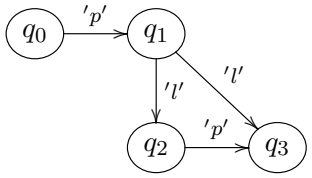
Las máquinas de estados no determinísticas (MEN) se pueden ver como una descripción de un sistema que, al recibir como entrada una constante de un alfabeto (que en general llamamos Σ), y encontrándose en un estado q, altera su estado según lo indique una función de transición (que en general llamamos δ). Observemos que al ser una máquina no determinística, el resultado de esta función de transición no es un único estado sino un conjunto de ellos.

Modelaremos estos autómatas mediante el tipo `MEN`[1]

```
data MEN a b = AM {sigma :: [a], delta :: (a -> b -> [b])}
```

Luego, el sistema representado por `m :: MEN a b` que se encuentra en un estado `q :: b` , después de recibir una entrada `s :: a` tal que `s` $\in$ `sigma m`, se encontrará en alguno de los estados `delta m s q` (si esta lista es vacía significa que `s` es una transición inválida, mientras que si contiene muchos estados, significa que puede alcanzar cuaquiera de ellos, sin que podamos suponer nada sobre cuál será).

M_{PLP} =




```
MPLP :: MEN Char Char
MPLP = AM ['l','p'] tran
  where tran s e | e == q0 && s == 'p'= [q1]
                 | e == q1 && s == 'l'= [q2, q3]
                 | e == q2 && s == 'p'= [q3]
                 | otherwise = []
```

Se pide definir las siguientes funciones, **sin utilizar recursión explícita**: [2].

a. I) `agregarTransicion :: a -> b -> b -> MEN a b -> MEN a b` que, dada una constante s y dos estados q_0 y q_f, agrega al autómata la transición por s desde q_0 a q_f. Si lo necesita, puede suponer que la transición no está previamente definida en el autómata, que s ya pertenece al alfabeto y que está definida la igualdad para los tipos `a` y `b`, indicando qué suposiciones realiza y por qué.

   II) `interseccion :: Eq a => MEN a b -> MEN a c -> MEN a (b,c)` que dados dos autómatas m y n, devuelve el autómata intersección, cuyo alfabeto es la unión de los dos alfabetos, cuyos estados son el producto cartesiano del conjunto de estados de cada uno y que puede moverse de (q_m, q_n) por el símbolo s al estado (q'_m, q'_n) si y solo si m puede moverse de q_m a q'_m por s y n puede moverse de q_n a q'_n por el mismo s.

b. `alcanzables :: MEN a b -> b -> [a] -> [b]` que, dados un autómata m, un estado q y una cadena de símbolos ss, devuelve todos los estados en los que se puede encontrar m después de haber leido los símbolos de ss (en ese orden), habiendo partido del estado q. Recordar que el autómata es no determinístico y, por ende, para cada símbolo y estado, la función de transición `delta` devuelve una lista de estados posibles a los que se puede mover. Si lo necesita, puede suponer definida la igualdad para los tipos `a` y `b`.

   En el autómata de ejemplo, `alcanzables` M_{PLP} q_0 "*pl*" $\rightsquigarrow$ [q_2,q_3]

[1] Observar que con esta definición, automáticamente se definen las funciones `sigma :: MEN a b -> [a]` que devuelve el alfabeto de la máquina y `delta :: MEN a b -> (a -> b -> [b])` que devuelve su función de transición

[2] Asumir definida la función `union :: Eq a => [a] -> [a] -> [a]` que se comporta igual que `(++)` pero sin agregar repetidos al resultado

c. `trazas :: MEN a b -> b -> [[a]]` que, dado un autómata m y un estado q, devuelve la lista con todas las trazas posibles en m a partir de q, es decir, todas las cadenas de símbolos que pueden llevar a m desde q a algún estado mediante transiciones válidas.
Asumir que existe al menos un ciclo en el autómata, por lo que la lista resultante es infinita. Si lo necesita, puede suponer que tanto el alfabeto como el resultado de las transiciones son finitos, y que está definida la igualdad para los tipos `a` y `b`. Deberá indicar qué suposiciones realiza y por qué.

En el autómata de ejemplo, `trazas` M_{PLP} $q_0 \rightsquigarrow$ `[['p'], ['p', 'l'], ['p', 'l', 'p']]`

Ejercicio 2 - Cálculo Lambda Tipado

Se desea extender el lenguaje de cálculo lambda tipado para soportar los RoseTrees alternados. Estos poseen dos tipos distintos, los cuales se van intercalando en cada nivel del árbol. Formalmente, el nuevo tipo $RT_{\sigma,\tau}$ representará a los RoseTrees que tengan elementos del tipo σ en sus niveles impares y elementos de tipo τ en los pares, suponiendo que los niveles se enumeran tomando como 1 a la raíz del árbol. Además, se desea definir un `map` para este tipo de árboles, que llamaremos $mapRT$. Éste toma dos funciones F y G (una para cada tipo de los elementos del RoseTree) y las aplica sucesivamente a los elementos de los tipos correspondientes, devolviendo otro RoseTree alternado con los resultados de las aplicaciones. Si lo necesita, puede suponer definida la extensión de listas vista durante la cursada, junto con su correspondiente `map`. En definitiva, la extensión de este ejercicio modifica los conjuntos de tipos y términos de la siguiente manera:

$$\sigma \ ::= \ ... \mid RT_{\sigma,\tau} \qquad M, N, F, G \ ::= \ ... \mid Rose(M, N) \mid mapRT(F, G, M)$$

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

b. Definir el conjunto de valores e introducir las reglas de semántica para la extensión propuesta.

c. Exhibir una reducción del siguiente término. Deberán mostrarse todos los pasos que involucren las reglas de semántica definidas en el ejercicio anterior, pero pueden condensarse los que no.

$$mapRT(not,\ iZ,\ Rose(\mathsf{False}, Rose(0, []_{RT_{\mathsf{Bool},\mathsf{Nat}}}) :: \ []_{RT_{\mathsf{Nat},\mathsf{Bool}}}))$$

Donde $iZ =^{def} \lambda x : \mathsf{Nat.isZero}(x)$ y $not =^{def} \lambda y : \mathsf{Bool.if}\ y\ \mathsf{then\ False\ else\ True}$

Ejercicio 3 - Inferencia de tipos

En este ejercicio extenderemos el algoritmo de inferencia para soportar un nuevo constructor de abstracciones sobre listas. Suponiendo que se cuenta con la extensión de listas vista durante la cursada, la extensión de los términos para este ejercicio es:

$$M ::= \ ... \mid \lambda x{:}\sigma :: xs.M$$

Y la regla de tipado es la siguiente:

$$\frac{\Gamma \cup \{x{:}\sigma,\ xs{:}[\sigma]\} \triangleright M : \tau}{\Gamma \triangleright \lambda x{:}\sigma :: xs.M : [\sigma] \to \tau}$$

De esta manera, podemos construir abstracciones cuyo cuerpo use tanto la cabeza como la cola de la lista.

a) Extender el algoritmo de inferencia para soportar el tipado de la nueva extensión.

b) Aplicar el algoritmo extendido con el método del árbol a la siguiente expresión, indicando las sustituciones aplicadas.

$$((\lambda x :: y.\ x)\ x)((\lambda x :: y.\ y)\ x)$$