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

Este examen se aprueba obteniendo al menos dos ejercicios bien menos (B-) y uno regular (R), y se promociona con tres ejercicios bien menos (B-) y las respuestas adecuadas a las preguntas teóricas. Las notas para cada ejercicio son: -, I, R, B-, B. 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 - Subtipado

Considerar el Cálculo Lambda extendido con listas. Se agrega además la siguiente extensión con intervalos de números naturales e iteradores.

$\sigma ::= \cdots \mid \mathsf{Intervalo} \mid \mathsf{Iterador}_{\sigma}$

$M ::= \cdots \mid [M, M] \mid \text{inicio}(M) \mid \text{fin}(M) \mid \text{case } M \text{ of } [\,] \rightsquigarrow M; x :: y \rightsquigarrow M \mid \text{iterar}(M) \mid \text{próximo}(M) \mid \text{avanzar}(M)$

Se incorporan las siguientes reglas de tipado:

$$\frac{\Gamma \triangleright M : \mathsf{Nat} \quad \Gamma \triangleright N : \mathsf{Nat}}{\Gamma \triangleright [M, N] : \mathsf{Intervalo}} \qquad \frac{\Gamma \triangleright M : \mathsf{Intervalo}}{\Gamma \triangleright \text{inicio}(M) : \mathsf{Nat}} \qquad \frac{\Gamma \triangleright M : \mathsf{Intervalo}}{\Gamma \triangleright \text{fin}(M) : \mathsf{Nat}} \qquad \frac{\Gamma \triangleright M : \mathsf{Intervalo}}{\Gamma \triangleright \text{iterar}(M) : \mathsf{Iterador}_{\mathsf{Intervalo}}}$$

$$\frac{\Gamma \triangleright M : [\sigma]}{\Gamma \triangleright \text{iterar}(M) : \mathsf{Iterador}_{[\sigma]}} \qquad \frac{\Gamma \triangleright M : \mathsf{Iterador}_{\tau}}{\Gamma \triangleright \text{fin}(M) : \mathsf{Bool}} \qquad \frac{\Gamma \triangleright M : \mathsf{Iterador}_{\tau}}{\Gamma \triangleright \text{avanzar}(M) : \mathsf{Iterador}_{\tau}}$$

$$\frac{\Gamma \triangleright M : \mathsf{Iterador}_{\mathsf{Intervalo}}}{\Gamma \triangleright \text{próximo}(M) : \mathsf{Nat}} \qquad \frac{\Gamma \triangleright M : \mathsf{Iterador}_{[\sigma]}}{\Gamma \triangleright \text{próximo}(M) : \sigma} \qquad \frac{\Gamma \triangleright M : \mathsf{Intervalo} \quad \Gamma \triangleright N : \tau \quad \Gamma, h : \mathsf{Nat}, t : \mathsf{Intervalo} \triangleright O : \tau}{\Gamma \triangleright \text{case } M \text{ of } [\,] \rightsquigarrow N; h :: t \rightsquigarrow O : \tau}$$

a) ¿Qué relación de subtipado es razonable establecer entre los intervalos y las listas, y entre iteradores de distintos tipos? Escribir las reglas correspondientes y justificar en términos del principio de sustitutividad.

b) Determinar cuál de las siguientes expresiones debería ser tipable, y demostrar que lo es utilizando las nuevas reglas[1].

   I) $(\lambda i.\mathsf{Iterador}_{[\mathsf{Float}]}.\text{próximo}(i))\ \text{iterar}(\text{-1} :: [0, \mathsf{Succ}(0)])$

   II) $(\lambda i.\mathsf{Iterador}_{\mathsf{Intervalo}}.\mathsf{Succ}(\text{próximo}(i)))\ \text{iterar}(2{,}5 :: [\,]_{\mathsf{Float}})$

Ejercicio 2 - Programación Orientada a Objetos

a) Definir en JavaScript la función constructora `InfiniteSequence`, que genere objetos con un atributo `val` y un método `next`, con el siguiente comportamiento:

   `new InfiniteSequence().val` $\rightsquigarrow 1$

   `new InfiniteSequence().next().val` $\rightsquigarrow 2$

   `new InfiniteSequence().next().next().val` $\rightsquigarrow 3$

   ...Y así sucesivamente. Es decir, el resultado de enviar el mensaje `next()` es un nuevo objeto cuyo atributo `val` es el sucesor del `val` del objeto receptor, y el resto se mantiene igual. **El objeto receptor no debe modificarse** al recibir el mensaje `next()`.

b) Definir en el cálculo ς un objeto que se comporte de la misma manera que los objetos creados mediante la función constructora `InfiniteSequence` del punto anterior. Tener en cuenta que `next` es una función.

[1] Suponer que $\Gamma \triangleright$ -1 : Int y $\Gamma \triangleright$ 2,5 : Float, y que las listas son covariantes.

Ejercicio 3 - Programación Lógica

Implementar los predicados respetando en cada caso la instanciación pedida. Los generadores deben cubrir todas las instancias válidas de aquello que generan sin repetir dos veces la misma. No usar cut (`!`) ni predicados de alto orden como `setof`, con la única excepción de `not`.

En este ejercicio trabajaremos con procesos interactivos, abstrayéndonos de su representación. Un proceso puede ser ejecutado paso por paso hasta que finaliza, o en algunos casos puede seguir ejecutándose indefinidamente.

En cada paso, un proceso realiza una acción cuyo efecto puede ser generar una salida o solicitar un dato de entrada. Estos dos efectos se representarán respectivamente como `s(X)` y `e(X)`, siendo `X` el dato producido o ingresado.

Disponemos del predicado `accion(+Proceso, ?Efecto, -SiguienteProceso)`, que es verdadero cuando el proceso `Proceso` puede realizar una acción con efecto `Efecto`, siendo `ProcesoSiguiente` el proceso al que reduce luego de realizar esa acción. El proceso puede no ser determinístico, lo que significa que puede haber más de una acción disponible para un mismo proceso, produciendo ya sea el mismo efecto o efectos diferentes.

a) Definir el predicado `traza(+Proceso, -Traza)`, que es verdadero cuando `Traza` es una secuencia de efectos que se producen desde el inicio de `Proceso` hasta su finalización.

   Por ejemplo, si el proceso modela el comportamiento de una máquina de café, una traza posible sería: `[e(opciónCafé), e(dinero), s(café)]`.

b) El predicado definido para el funtor `traza/2` ¿es reversible en alguno de sus parámetros? Justificar.

c) Definir el predicado `simula(+Proceso1,+Proceso2)` que es verdadero cuando, para cada acción de entrada o salida que realice `Proceso1`, `Proceso2` puede realizar la misma acción resultando en un proceso que a su vez simula al proceso al que redujo `Proceso1`. Es decir, siempre que `accion(Proceso1, X, Sig1)` sea verdadero, debe existir `Sig2` tal que `accion(Proceso2, X, Sig2)` y `Sig2` simula a `Sig1`.