PLP - Segundo Parcial - *Verano* 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. **Entregar cada ejercicio en hojas separadas.**

Ejercicio 1 - Subtipado

Considerar la extensión del cálculo lambda para definir la conjunción ($\sigma ::= ... \mid \sigma_1 \wedge \sigma_2$) con las siguientes reglas de **subtipado**:

$$\frac{}{\sigma \wedge \tau <: \sigma}\ (\text{S-Comp}) \qquad \frac{}{\sigma \wedge \tau <: \tau \wedge \sigma}\ (\text{S-Conm}) \qquad \frac{\sigma <: \sigma' \qquad \tau <: \tau'}{\sigma \wedge \tau <: \sigma' \wedge \tau'}\ (\text{S-Inter})$$

a) Indicar si cambia la relación de subtipado agregando la siguiente regla. Justifique.

$$\frac{\tau <: \tau'}{\sigma \wedge \tau <: \tau'}\ (\text{S-Comp2})$$

b) En este inciso extenderemos además el sistema de tipos para trabajar con expresiones sobre idiomas.

$\sigma ::= ... \mid \sigma_1 \wedge \sigma_2 \mid \texttt{idioma} \mid \texttt{idiomaIngles} \mid \texttt{idiomaEspañol} \mid \texttt{personaQueHabla}(\sigma) \quad M ::= ... \mid \texttt{hablar}(M, M)$

Donde $\texttt{personaQueHabla}(\sigma)$ es una persona que puede hablar **todos** los idiomas de tipo σ, teniendo en cuenta que una persona puede hablar más de un idioma. Por ejemplo, una persona nacida en Inglaterra que nunca aprendió otro idioma, diremos que es de tipo $\texttt{personaQueHabla}(\texttt{idiomaIngles})$, mientras que una persona nacida en Argentina que estudió además inglés es de tipo $\texttt{personaQueHabla}(\texttt{idiomaEspañol} \wedge \texttt{idiomaIngles})$. Notar que obviamos en la sintaxis las reglas que permiten construir términos que tienen tipo idioma, idiomaIngles, idiomaEspañol, $\texttt{personaQueHabla}(\sigma)$. Consideremos entonces la siguiente regla de **tipado** y subtipado:

$$\frac{\Gamma \triangleright M : \texttt{personaQueHabla}(\sigma) \quad \Gamma \triangleright N : \sigma}{\Gamma \triangleright \texttt{hablar}(M, N) : \texttt{Unit}}\ (\text{T-Pers})$$

$$\frac{}{\texttt{idiomaIngles} <: \texttt{idioma}}\ (\text{S-II}) \qquad \frac{}{\texttt{idiomaEspañol} <: \texttt{idioma}}\ (\text{S-EI})$$

Se pide: definir la/s regla/s de subtipado adecuada/s para completar esta extensión, justificando en términos del principio de sustitutividad.

Ejercicio 2 - Javascript

En este ejercicio trabajaremos con pilas acotadas, es decir, pilas que pueden apilar elementos mientras su tamaño sea menor a una cota.

a. Definir el objeto `vacia`, que representa la pila vacía acotada a dos elementos. En particular deberá responder los siguientes mensajes:

   I) `esVacia`: retorna siempre `true`.

   II) `cota`: retorna 2.

   III) `apilar(x)`: retorna una pila no vacía, cuyo tope es el elemento `x`, y además provee de los mensajes `tope` y `desapilar`. El mensaje `tope` debe retornar el elemento `x` y `desapilar` retorna el objeto receptor del mensaje `apilar(x)`. Tener en cuenta que solo se apilará cuando la cantidad de elementos sea menor a la cota del objeto receptor. Notar que la nueva pila responda adecuadamente a `esVacia`, `cota` y `apilar`.

b. Considere el objeto `vacia.apilar('a').apilar('b')`. Indicar cuántos y qué objetos deben visitarse cuando se ejecuta el método *dispatch* del mensaje `apilar('b')`.

Ejercicio 3 - Lógica

Para la resolución de este ejercicio no está permitido usar cut (`!`) ni predicados de alto orden como `setof`, con la única excepción de `not`.

a) Definir el predicado `pesoMaximo(+Lista, -Peso)` que dada una lista `Lista` heterogénea que puede contener números enteros o listas, debe ser verdadero cuando `Peso` sea el peso del elemento más pesado de la lista. Para ello se tendrá en cuenta que el peso está representado:

- en los `números`, por su valor absoluto.
- en las `listas`, por la suma de los pesos de sus elementos.

?- pesoMaximo([5, [2,3], 8, [4,8,9], [5,4,[1,11]], -2], P).

P = 21 ;

false

La lista vacía no tiene peso, y cada lista tiene al menos un elemento.

b) Definir el predicado `elementoMasPesado(+Lista, -Elemento)` que dada una lista como la del punto anterior y un elemento de la misma, tenga éxito cuando `Elemento` sea el elemento más pesado de la lista.

?- elementoMasPesado([5, [2,3], 8, [4,8,9], [5,4,[1,11]], -2], E).

E = [4, 8, 9] ;

E = [5, 4, [1, 11]] ;

false

Preguntas Teóricas

a. Indicar si las siguientes afirmaciones son verdaderas o falsas. Justificar:

I) $\{\{P(x, g(x))\}, \{P(f(g(x)), g(x))\}$ es una forma clausal de $\forall x.\exists y.(\exists z.P(z, y) \wedge P(x, y))$.

II) Considere la siguiente fórmula en forma clausal:

$$\{\{P(X, g(a), Y)\}, \{\neg P(Z, g(X), Z)\}, \{P(W, g(W), g(W))\}\}$$

1) Existen al menos dos refutaciones SLD distintas.
2) Prolog computa la solución $\{X \leftarrow a, Z \leftarrow a\}$.

b. Definir, si es posible, un predicado `p(X)` tal que el árbol SLD para el goal $\neg$`p(W)` es infinito y

I) Prolog responde `false` a la consulta `not(p(W))`.

II) Prolog responde `true` a la consulta `not(p(W))`.

c. Considere un conjunto de traits $\mathtt{t}_1, \ldots, \mathtt{t}_n$ definidos de manera tal que para todo i

$$\mathtt{t}_i = [x = \lambda(z)b_i^x, y = \lambda(z)b_i^y]$$

I) Definir una clase `C` que provee un constructor `new` que recibe como parámetro a uno de estos traits y crea instancias que responden a los mensajes x e y de acuerdo con el trait usado al momento de la creación. Por ejemplo, si se definen $\mathtt{t}_1 = [x = \lambda(z)0, y = \lambda(z)\mathtt{true}]$ y $\mathtt{t}_2 = [x = \lambda(z)1, y = \lambda(z)[]]$, entonces `C.new(`$\mathtt{t}_1$`).x` evalúa `0` y `C.new(`$\mathtt{t}_2$`).x` evalúa `1`.

II) Cómo se modifica la definición de `C` si las instancias generadas deben permitir cambiar dinámicamente al trait que utilizan. Por ejemplo, `C.new(`$\mathtt{t}_1$`).cambiar(`$\mathtt{t}_2$`).x` debe evaluar `1`.