PLP - Segundo Parcial - 1er cuatrimestre de 2018

Este examen se aprueba obteniendo al menos dos ejercicios bien (B) y dos regular (R), o con tres ejercicios bien menos (B-) y uno insuficiente (I) pero habiendo entregado algo que contribuya a la solución del ejercicio. 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 - 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`.

Tenemos la siguiente base de conocimiento sobre los equipos participantes del mundial de fútbol, que capturan, por un lado, el conjunto de los equipos participantes y por otro, una estimación de la probabilidad de que un equipo le gane a otro.

`participantes([argentina, equipo1, equipo2, ....]).`

`probabilidadDeGanar(?E1, ?E2, ?P)`, que es verdadero cuando la probabilidad de que E1 le gane a E2 es P. Podemos suponer que está definido exactamente para todas las combinaciones válidas de equipos participantes.

a) Definir los predicados `gana(?E1, ?E2)`, `empatan(?E1, ?E2)` y `pierde(?E1, ?E2)`. Consideramos que un equipo le gana a otro si la probabilidad de ganarle estimada supera 0.5, empatan si es exactamente igual y pierde si es inferior.

b) Definir el predicado `puntaje(+E, +G, -P)` que es verdadero cuando el puntaje del equipo E, después de jugar contra todos los equipos de la lista G, es P (notar que E no pertenece a G). Recordemos que se otorgan tres puntos al ganador de un partido, un punto a cada equipo si el resultado es un empate y ningún punto al perdedor.

c) Definir el predicado `pasaAOctavos(+E, ?G)` que es verdadero cuando el equipo E supera la fase de grupos si forma parte del grupo $\{E\} \cup G$. Recordemos que en la fase de grupos cada equipo juega contra todos los otros equipos de su grupo (una vez) y clasifica el primero en puntaje del grupo. Supongamos que si existiera más de un equipo con el mejor puntaje, clasifican todos estos.

d) Definir el predicado `vamosArgentina(?G)` que es verdadero cuando G es un grupo de 4 equipos participantes, del cual argentina forma parte y en el cual supera la fase de grupos.

Ejercicio 2 - Resolución

Sean las siguientes leyes sobre la amistad:

- "Soy amigo de mis amigos": $\forall x \forall y \; A(x,y) \supset A(y,x)$
- "Los amigos de mis amigos son mis amigos": $\forall x \forall y \forall z \; (A(x,y) \wedge A(y,z)) \supset A(x,z)$

a) Demostrar, utilizando resolución, que si alguien tiene un amigo, entonces también es amigo de sí mismo: $\forall x (\exists p \; A(x,p)) \supset A(x,x)$

b) ¿La demostración realizada es SLD? ¿Por qué?

Ejercicio 3 - Programación Orientada a Objetos

En este ejercicio implementaremos funcionalidad que permita realizar *monkey patching*, esto es la posibilidad de alterar la respuesta a algunos mensajes en tiempo de ejecución.

Para esto, cualquier clase deberá poder responder al mensaje `patch:responseWith` que dado un selector y un valor, producirá una nueva clase con el mismo comportamiento que la original salvo al momento de recibir un mensaje con el selector provisto, donde responderá con el valor asignado (ver `escenario1`). El comportamiento para el resto de los mensajes debe permanece inalterado.

```
escenario1
  | patchedArray anArray aPatchedArray |
 anArray := Array new: 20.
 patchedArray := Array patch: #capacity
                      responseWith: 1.

 aPatchedArray := patchedArray new: 20.

 assert: anArray capacity equals: 20.
 assert: aPatchableArray capacity equals: 1.
```

```
escenario2
  | patchedArray aPatchedArray otherPatchedArray |
 patchedArray := Array patch: #capacity responseWith 247.

 aPatchedArray := patchedArray new: 20.
 otherPatchedArray := patchedArray new: 40.

 assert: aPatchableArray capacity equals: 247.
 assert: otherPatchableArray capacity equals: 247.
```

a. Implementar el mensaje `patch:responseWith:` dejando muy en claro en qué clase(s) se haría y si es un mensaje de clase o de instancia. Pista: prestar atención al `escenario2` y recordar el mensaje `compile:`.

b. Implementar el mensaje `patchMessagesWith:` que dado un diccionario de selectores a valores, cree una clase cuyos mensajes definidos con los selectores provistos retornen el valor apuntado por el diccionario.

Ejercicio 4 - Subtipado

Los servicios web suelen exponerse a través de lo que se llaman APIs. Una API(Application Programming Interface) es una capa de abstracción que permite integrar diversas piezas de software mediante el uso de una interfaz independiente de la implementación subyacente. De esta manera, decimos que una API expone ciertos servicios (con su nombre y su tipo) y las aplicaciones los consumen utilizando su nombre.

Las APIs suelen ir modificándose con el tiempo y para que los programas que las consumen continuen siendo válidos, los cambios deben ser retrocompatibles. Es decir, una nueva versión de la API debe poder ser utilizada en lugar de la vieja sin producir errores.

Modelaremos una API mediante la siguiente extensión del Cálculo Lambda.

$$\sigma ::= \cdots \mid \text{API}\,[(s_1, \sigma_1 \to \tau_1), \ldots, (s_n, \sigma_n \to \tau_n)]$$
$$M ::= \cdots \mid \mathit{consumir}\langle A, s\rangle \mid \mathit{exponer}\langle s_1 = M, \ldots, s_n = M\rangle$$

$$\frac{\Gamma \rhd M_i : \sigma_i \to \tau_i \qquad i \in \{1 \ldots n\}}{\Gamma \rhd \mathit{exponer}\langle s_1 = M_1, \ldots, s_n = M_n\rangle : \text{API}\,[(s_1, \sigma_1 \to \tau_1), \ldots, (s_n, \sigma_n \to \tau_n)]}\ (\text{T-Exponer})$$

$$\frac{\Gamma \rhd A : \text{API}\,[(s_1, \sigma_1 \to \tau_1), \ldots, (s_n, \sigma_n \to \tau_n)] \quad i \in \{1 \ldots n\}}{\Gamma \rhd \mathit{consumir}\langle A, s_i\rangle : \sigma_i \to \tau_i}\ (\text{T-Consumir})$$

a. Extender la definición de subtipado para el tipo de APIs, justificando las decisiones en función del principio de sustitutividad, de manera tal que la siguiente expresión sea bien tipada:

$$\emptyset \rhd (\lambda a : \text{API}[(\mathit{inc}, \mathit{Nat} \to \mathit{Int})].\ \mathit{consumir}\langle a, \mathit{inc}\rangle\ 0)\ \mathit{exponer}\langle \mathit{inc} = \lambda x : \mathit{Nat}.\ \mathit{succ}(x), \mathit{natId} = \lambda x : \mathit{Nat}.\ x\rangle : \sigma$$

b. Elegir un σ y exhibir una derivación para el juicio de tipado del inciso anterior.