PLP - Segundo Parcial - edición especial
2º cuatrimestre de 2007

Este examen se aprueba obteniendo al menos **70 puntos**. 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. El puntaje de los distintos items no está necesariamente en relación a su dificultad.

Ejercicio 1 - Programación Lógica (30 puntos)

Implementar los siguientes predicados respetando la instanciación pedida. No utilizar cut (!) ni predicados de alto orden (como **setof**). La única excepción es el **not**, que está permitido.

a) **(8 puntos)** Escribir el predicado `perm(+Lista,-Perm)` que dice si `Perm` tiene exactamente los elementos de `Lista`, en algún orden. No se deben devolver soluciones repetidas si la lista no tiene elementos repetidos.

Ejemplo:

```
perm([1,2,3],R).
R = [1,2,3] ;
R = [1,3,2] ;
R = [2,1,3] ;
R = [2,3,1] ;
R = [3,1,2] ;
R = [3,2,1] ;
No
```

b) **(2 puntos)** Si se permitiera usar predicados de alto orden, ¿cómo se podría evitar generar soluciones repetidas en el ítem anterior aun cuando la lista tuviera elementos repetidos?

c) **(14 puntos)** Escribir el predicado `mismasRamas(+Arbol1,-Arbol2)` que indica si el multiconjunto de ramas de ambos árboles binarios es igual. Los árboles se representarán como:

```
esArbol(nil).
esArbol(bin(I,_,D)):-esArbol(I),esArbol(D).
esRama([],nil).
esRama([X | Xs],bin(I,X,_)):-esRama(Xs,I).
esRama([X | Xs],bin(_,X,D)):-esRama(Xs,D).
```

La definición de rama está dada por el predicado `esRama`. No se deben devolver soluciones repetidas. Considerar usar el ítem anterior.

Ejemplo (por simplicidad $\{\alpha\}$ representará `bin(nil,`α`,nil)`):

```
mismasRamas(bin({5},1,bin({2},5,{3})),R).
R = bin({5},1,bin({2},5,{3})) ;
R = bin({5},1,bin({3},5,{2})) ;
R = bin(bin({2},5,{3}),1,{5}) ;
R = bin(bin({3},5,{2}),1,{5}) ;
R = bin(bin({2},5,nil),1,bin({3},5,nil)) ;
R = bin(bin(nil,5,{2}),1,bin({3},5,nil)) ;
R = bin(bin({2},5,nil),1,bin(nil,5,{3})) ;
R = bin(bin(nil,5,{2}),1,bin(nil,5,{3})) ;
R = bin(bin({3},5,nil),1,bin({2},5,nil)) ;
R = bin(bin(nil,5,{3}),1,bin({2},5,nil)) ;
R = bin(bin({3},5,nil),1,bin(nil,5,{2})) ;
R = bin(bin(nil,5,{3}),1,bin(nil,5,{2})) ;
No
```

Asumir dado el predicado `agregarATodas(-Elemento,-Listas,-Resultado)` que tiene éxito si `Listas` es una lista de listas y `Resultado` es el resultado de agregar `Elemento` al comienzo de cada una de las listas. Notar que este predicado es reversible.

d) **(6 puntos)** Escribir el predicado `inorder(-Arbol,-Lista)` que dice si la lista representa el recorrido *inorder* de los nodos del árbol. el cual consiste en recorrer primero el subárbol izquierdo, luego la raíz y luego el subárbol derecho. Notar que `inorder` debe ser totalmente reversible, y recordar que `append(-Ll,-Lr,-L)` lo es.

Ejemplo:

```
inorder(bin({5},1,bin({2},5,{3})),R).
R = [5,1,2,5,3] ;
No
inorder(R,[1,2,3]).
R = bin(nil,1,bin(nil,2,{3})) ;
R = bin(nil,1,bin({2},3,nil)) ;
R = bin({1},2,{3}) ;
R = bin(bin(nil,1,{2}),3,nil) ;
R = bin(bin({1},2,nil),3,nil) ;
No
```

Ejercicio 2 - Resolución Lógica (20 puntos)

Sean s una función unaria que representa el sucesor, N un predicado unario que representa ser natural y M un predicado binario que representa la relación mayor o igual.

Dados los siguientes axiomas usuales:

1. El sucesor de todo número natural es un natural.
2. El sucesor de x es mayor o igual que el sucesor de y si y sólo si x es mayor o igual que y.
3. Ningún natural es mayor o igual que su sucesor.

a) **(10 puntos)** Escribir los axiomas como fórmulas de primer orden y pasarlas a forma clausal.

b) **(10 puntos)** Demostrar usando el método de resolución general que no existe un máximo número natural (o sea, un número que sea mayor o igual que todo el resto).

Ejercicio 3 - Smalltalk - Herencia (10 puntos)

Sea `A` una clase de smalltalk y `B` y `C` dos clases, ambas hijas de `A`. Sean a, b y c la cantidad de mensajes que entienden `A`, `B` y `C` respectivamente. Indicar para cada ítem si es necesariamente verdadero, necesariamente falso o puede ser verdadero o falso según el caso. De ser éste último caso, mostrar informalmente un ejemplo donde se haga verdadero y uno donde se haga falso.

a) **(2 puntos)** $a < b$
b) **(2 puntos)** $b \leq c$
c) **(2 puntos)** $a \leq c$
d) **(2 puntos)** `super do;` ejecuta el mismo código si se llama desde `B` o `C`.
e) **(2 puntos)** `(super yourself) do;` ejecuta el mismo código si se llama desde `B` o `C`.

Ejercicio 4 - Smalltalk - Colecciones (25 puntos)

a) **(14 puntos)** Extender la clase `Collection` para que soporte un mensaje `select: selBlock where: whBlock groupBy: gbBlock having: hvBlock` cuyos 4 parámetros son bloques con la siguiente semántica:

- `whBlock` lleva un parámetro y al ser evaluado en cada elemento contenido en la colección receptora devuelve un booleano. Aquellos elementos para los cuales el bloque devuelva falso deben ser ignorados por completo y el resto considerados en lo que sigue.
- `gbBlock` lleva un parámetro que al ser evaluado en los elementos de la colección receptora devuelve un identificador de grupo. Todos los elementos que pasaron el where deben ser agrupados con los que tengan su mismo identificador. Cada grupo consistirá en una colección del mismo tipo de la receptora del mensaje con los elementos originales agregados en el orden relativo en el que estaban.
- `hvBlock` lleva dos parámetros y al ser evaluado en cada grupo (el primer parámetro es el identificador del grupo y el segundo la colección con los elementos del grupo) devuelve un boolean que indica si el grupo se debe dar como resultado o no.
- `selBlock` lleva dos parámetros y al ser evaluado en cada grupo no descartado devuelve un elemento que se debe agregar a la colección resultado. La colección resultado también es del mismo tipo que la receptora. El orden del resultado no tiene importancia.

Ejemplo: sea `oC` la `OrderedCollection` con los elementos: `(2 1 10 3 5 0 6 9 4 1)`

```
oC select: [:c :x|x] where: [:x | x∼=3] groupBy: [:x | x rem: 5] having: [:c :x | true]
```

devuelve:

```
OrderedCollection (OrderedCollection (10 5 0) OrderedCollection (1 6 1) OrderedCollection (2)
OrderedCollection (9 4))
```

Nota: se puede asumir que los identificadores de grupo pueden ser comparados entre ellos con `=`. Asumir que la colección acepta el mensaje `add:`.

Ayuda: usar los métodos `includesKey:` y `keysAndValuesDo:` de la clase `Dictionary`. Este último funciona como `do:` pero toma dos parámetros, siendo el primero la clave y el segundo el valor.

b) **(5 puntos) Usando el ítem (a)** extender con el mensaje `sinRepetidos` la clase `Collection`. Este mensaje no toma ningún parámetro y devuelve una colección igual a la receptora pero sin elementos repetidos. El orden de los elementos del resultado no tiene importancia.

c) **(6 puntos) Usando el ítem (a)** extender con el mensaje `cantApariciones` la clase `Collection`. Este mensaje toma una colección como parámetro y devuelve una colección del mismo tipo que la receptora con 2 elementos: la cantidad de elementos en común con el parámetro y la cantidad de elementos de la receptora que no aparecen en el parámetro.

Ejercicio 5 - Subtipado (15 puntos)

En general los elementos de cierto conjunto pueden ser vistos también como una función constante que va hacia dicho conjunto. Por ejemplo, el número 3 puede ser visto como el número 3 o como la función constante que siempre devuelve 3. Para reflejar esto, agregaremos a los axiomas de subtipado el siguiente:

$$\frac{}{T <: S \to T}$$

Encontrar en este nuevo contexto, la cantidad de tipos X que cumplen las siguientes expresiones, considerando la desigualdad estricta (o sea, X no es igual a ninguna de sus cotas) y las permutaciones de registros como el mismo tipo.

a) **(9 puntos)** $Int \to Bool <: X <: Float \to Int \to Nat$

b) **(6 puntos)** $\{x : Nat \to Bool, y : Bool\} <: X <: \{y : Float\}$