PLP - Segundo Recuperatorio - 2do cuatrimestre de 2024

#Orden	Nro. Libreta	[redacted] y Nombre	E#1	E#2	E#3	Nota Final
[redacted]						

Este examen se aprueba obteniendo al menos dos ejercicios bien (B) y uno regular (R), y se promociona con al menos dos ejercicios muy bien (MB) y uno bien (B). Es posible obtener una aprobación condicional con un ejercicio muy bien (MB), uno bien (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, MB, E. Entregar cada ejercicio en hojas separadas. Poner nombre, apellido y número de orden en todas las hojas, y numerarlas. Se puede utilizar todo lo definido en las prácticas y todo lo que se dio en clase, colocando referencias claras. El orden de los ejercicios es arbitrario. Recomendamos leer el parcial completo antes de empezar a resolverlo.

Ejercicio 1 - Resolución

a) Dada la siguiente base de conocimientos en Prolog:

```
member(X,[X|XS]).
member(X,[Y|XS]) :- member(X,XS).

natural(cero).
natural(succ(X)) :- natural(X).

listaDeNats([]).
listaDeNats([X|XS]) :- natural(X), listaDeNats(XS).
```

Explicar **brevemente** con palabras qué sucede al realizar la siguiente consulta:

```
listaDeNats(XS), member(succ(cero),XS).
```

b) Expresar la base de conocimientos y la consulta anterior como fórmulas lógicas.

c) Pasar las fórmulas a forma clausal y encontrar una solución a la consulta utilizando resolución.

d) La resolución utilizada en el punto anterior ¿fue SLD? Justificar.

Ejercicio 2 - 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. Se deben indicar los patrones de instanciación de todos los predicados auxiliares. No usar cut (!) ni predicados de alto orden como `setof`, con la única excepción de `not`.

a) Implementar el predicado `caminoDesde(+P, -C)` que es verdadero cuando `C` es un camino en un tablero infinito a partir de una posición inicial `P`. Representaremos a los caminos como una lista de posiciones `(X,Y)`, donde `X` e `Y` son números enteros. En cada paso hay cuatro direcciones posibles: arriba, abajo, izquierda y derecha. Por ejemplo

```
?- caminoDesde((0,0),C).
C = [(0, 0)] ;
C = [(0, 0),(0, 1)] ;
C = [(0, 0),(0, -1)] ;
C = [(0, 0),(1, 0)] ;
C = [(0, 0),(-1, 0)] ;
C = [(0, 0),(0, 1),(0, 2)] ;
C = [(0, 0),(0, 1),(0, 0)] ;
C = [(0, 0),(0, 1),(1, 1)] ;
C = [(0, 0),(0, 1),(-1, 1)] ;
C = [(0, 0),(0, -1),(0, 0)] ;
C = [(0, 0),(0, -1),(0, -2)] ;
...
```

b) Trabajaremos con el conocido problema de optimización de la mochila. Consideremos definidos los hechos `objeto(?Id,?P,?V)` que instancian diversos objetos con identificador `Id`, peso `P` y valor `V`. Implementar el predicado `mochila(+C,-L)` que es verdadero cuando `L` es una lista de identificadores de objetos que al guardarlos en la mochila no supera su capacidad `C`. Por ejemplo, supongamos definidos los siguientes objetos:

```
objeto(1,50,10).
objeto(2,75,15).
objeto(3,60,5).
objeto(4,10,1).
```

Luego, podemos realizar la siguiente consulta:

```
?- mochila(70,L).
L = [] ;
L = [1] ;
L = [1,4] ;
L = [3] ;
L = [3,4] ;
L = [4].
```

Notar que consideramos a las listas como conjuntos, por lo tanto no se deben devolver soluciones repetidas, es decir, listas que contengan los mismos elementos en distinto orden. El orden de los elementos dentro de la lista es arbitrario.

c) Definir el predicado `mejorMochila(+C,-L)` que es verdadero cuando `L` es una lista de identificadores de objetos que se pueden guardar en la mochila tal que no superen la capacidad `C` y maximicen el valor. Puede haber más de una solución. Por ejemplo:

```
?- mejorMochila(70,L).
 L = [1,4] ;
 false.
```

Ejercicio 3 - Inferencia y Objetos

a) Consideremos extendido el Cálculo Lambda (con listas) para representar *RoseTrees*. Para eso se extiende el conjunto de tipos y el de términos de la siguiente manera:

$$\sigma ::= \ldots \mid \mathrm{RT}_\sigma \qquad M ::= \ldots \mid \mathsf{Rose}(M, M) \mid \mathsf{Map}\ x : \sigma \mapsto M\ \mathsf{in}\ M$$

Los *RoseTrees* son los usuales, mientras que $\mathsf{Map}\ x : \sigma \mapsto M\ \mathsf{in}\ N$ representa un *RoseTree* con la misma estructura que N, pero que en cada nodo contiene el resultado de reemplazar las apariciones libres de x en M por el valor del nodo correspondiente en N, como se ve en el siguiente ejemplo:

$$\mathsf{Map}\ t : Nat \mapsto \mathsf{succ}(t)\ \mathsf{in}\ \mathsf{Rose}(\underline{0}, \mathsf{Rose}(\underline{1}, []) :: []_{\mathrm{RT}_{Nat}}) \rightsquigarrow \mathsf{Rose}(\underline{1}, \mathsf{Rose}(\underline{2}, []) :: []_{\mathrm{RT}_{Nat}})$$

El algoritmo de inferencia se extiende de la siguiente manera:

$\mathbb{W}(\mathsf{Rose}(U, V)) \stackrel{def}{=} S\Gamma_1 \cup S\Gamma_2 \vdash S(\mathsf{Rose}(M, N)) : S\ \mathrm{RT}_\sigma$
donde:

- $\mathbb{W}(U) = \Gamma_1 \vdash M : \sigma$
- $\mathbb{W}(V) = \Gamma_2 \vdash N : \tau$
- $S = \mathrm{mgu}\ (\{\tau \stackrel{?}{=} [\mathrm{RT}_\sigma]\} \cup \{\rho_1 \stackrel{?}{=} \rho_2 \mid x : \rho_1 \in \Gamma_1 \ \wedge\ x : \rho_2 \in \Gamma_2\})$

$\mathbb{W}(\mathsf{Map}\ x \mapsto U\ \mathsf{in}\ V) \stackrel{def}{=} S\Gamma_{1'} \cup S\Gamma_2 \vdash S(\mathsf{Map}\ x : \tau_x \mapsto M\ \mathsf{in}\ N) : S\ \mathrm{RT}_{\sigma_1}$
donde:

- $\mathbb{W}(U) = \Gamma_1 \vdash M : \sigma_1$
- $\mathbb{W}(V) = \Gamma_2 \vdash N : \sigma_2$
- $\tau_x = \begin{cases} \alpha \text{ si } x : \alpha \in \Gamma_1, \\ \text{variable fresca si no} \end{cases}$
- $\Gamma_{1'} = \Gamma_1 \ominus \{x\}$
- $S = \mathrm{mgu}\ (\{\sigma_2 \stackrel{?}{=} \mathrm{RT}_{\tau_x}\} \cup \{\rho_1 \stackrel{?}{=} \rho_2 \mid y : \rho_1 \in \Gamma_{1'} \ \wedge\ y : \rho_2 \in \Gamma_2\})$

Inferir el tipo de la siguiente expresión, o demostrar que no es tipable:

$$\mathsf{Map}\ z \mapsto (\lambda x.iszero(x))\ z\ \mathsf{in}\ \mathsf{Rose}(\underline{1}, z)$$

b) Considere las siguientes clases:

```
Object subclass: #X

  action1
    ^[self compute] value.

  compute
    ^10.

  baseValue
    ^self value + 5.

  value
    ^1.
```

```
X subclass: #Y

  action2
    ^super baseValue.

  baseValue
    ^20.

  value
    ^3.
```

Para cada una de las siguientes expresiones, hacer una tabla donde se indique, en orden, cada mensaje que se envía, qué objeto lo recibe, en qué clase está el método respectivo, y cuál es el resultado final de cada colaboración:

i) `Y new action1`

ii) `Y new action2`