TC2006 Lenguajes de Programación

Tarea 7: Programación Lógica en PROLOG

Semestre: Enero-Mayo de 2017

Forma de trabajo: Equipos de 2 integrantes
Forma de entrega: como asignación de Blackboard
Fecha de entrega: Miércoles 3 de Mayo de 2017

Esta tarea tiene como objetivo el que pongan en práctica los conocimientos y habilidades de programación básicas en el lenguaje Prolog. **NO** se aceptarán trabajos que no tengan la cantidad de integrantes especificada. Cualquier situación al respecto debe negociarse a la brevedad con el profesor.
Deberán subir como asignación de Blackboard, el archivo compreso con nombre **M_tarea7.zip** (donde sustituyan M por sus matrículas separadas con guiones bajos) que contenga el código Prolog que implementa la solución a los problemas. Se **penalizará** si no agregan comentarios con sus datos y la funcionalidad de cada procedimiento entregado.

1. ***(20 puntos)*** Tenemos el siguiente conocimiento directo:
   - Pedro padece gripe.
   - Pedro padece hepatitis.
   - Juan padece hepatitis.
   - María padece gripe.
   - Carlos padece intoxicación.
   - La fiebre es síntoma de gripe.
   - El cansancio es síntoma de hepatitis.
   - La diarrea es síntoma de intoxicación.
   - El cansancio es síntoma de gripe.
   - La aspirina suprime la fiebre.
   - El Lomotil suprime la diarrea.

   Además podemos aportar el siguiente conocimiento inferido:
   - Un fármaco alivia una enfermedad si la enfermedad tiene un síntoma que sea suprimido por el fármaco.
   - Una persona debería tomar un fármaco si padece una enfermedad que sea aliviada por el fármaco.

   Construye un programa en Prolog que refleje dicho conocimiento y permita resolver las siguientes consultas, además, escribe en comentarios, el código requerido para hacer cada query en Prolog:
   a) ¿Podemos conocer qué dolencia tiene Pedro? ¿Y María?
   b) ¿Quién padece gripe?
   c) ¿Qué síntomas tiene Pedro?
   d) ¿Quién padece diarrea?
   e) ¿Y quién está cansado?
   f) ¿Hay algún fármaco que alivie a Pedro?
   g) ¿Hay algún síntoma que compartan Juan y María?

2. ***(10 puntos)*** Implementar el predicado **intersectan** en Prolog que verifique si dos listas planas pasadas como sus argumentos tienen elementos en común.
   Probar con:
   ```
   ?- intersectan([a,b,c],[d,f,a])     => yes
   ?- intersectan([1,2,b],[a,3,c,d,4]) => no
   ```

3. ***(10 puntos)*** Implementar el predicado **rango** en Prolog que obtenga una lista incremental de números enteros entre dos valores pasados como argumentos. Asumir que el segundo argumento es un número mayor o igual al primero.
   Probar con:
   ```
   ?- rango(3,3,R) => R = [3]
   ```

```
?- rango(2,7,R) => R = [2,3,4,5,6,7]
```

4. ***(10 puntos)*** Implementar el predicado **cartesiano** en Prolog que obtenga una lista de pares de elementos construida como el producto cartesiano de dos conjuntos representados como listas.
Probar con:

```
?- cartesiano([][1,3,8],R)        => R = []
?- cartesiano([1,3,8],[a,b],R)
=> R = [[1,a],[1,b],[3,a],[3,b],[8,a],[8,b]]
```

5. ***(10 puntos)*** Implementar el predicado **cuenta_profundo** en Prolog que cuente las veces que aparece un elemento particular dentro de una lista imbricada.
Probar con:

```
?- cuenta_profundo(b,[a,b,c],R)                      => R = 1
?- cuenta_profundo(g,[a,[g,[c,d],[e,b,[g]],h],g],R) => R = 3
?- cuenta_profundo(z,[a,[b,[c,d],[e,f,[g]],h],i],R) => R = 0
```

6. ***(10 puntos)*** Implementar el predicado **tabla** en Prolog que obtenga la tabla de multiplicar de un número, como una lista de pares, donde el primer elemento del par contiene los multiplicandos y el segundo el producto
Probar con:

```
?- tabla(1,R)  => R = [[[1,1],1],[[1,2],2] ... [[1,10],10]]
?- tabla(4,R)  => R = [[[4,1],4],[[4,2],8] ... [[4,10],40]]
```

7. ***(10 puntos)*** Implementar el predicado **lista_unicos** en Prolog que obtenga una lista con los elementos que no aparecen repetidos dentro de una lista imbricada.
Probar con:

```
?- lista_unicos([a,b,a,b,a,c],R)       => R = [c]
?- lista_unicos([a,[b,[c,[a],c],b]],R) => R = []
?- lista_unicos([a,[b,[c,[d]]]],R)     => R = [a b c d]
```

8. ***(10 puntos)*** Implementar el predicado **mayores** en Prolog que regrese una lista con los elementos mayores que un valor dado en un árbol binario descrito con la función:
árbol(Raíz, SubárbolIzquierdo, SubárbolDerecho).
Probar con:

```
?- mayores(10,nil,H)     => H = []
?- mayores(10, arbol(8,arbol(5,arbol(2,nil,nil),arbol(7,nil,nil)),
                       arbol(9, nil, arbol(15, arbol(11, nil, nil), nil))),H).
   => H = [15,11]
 ?- mayores(0, arbol(8,arbol(5,arbol(2,nil,nil),arbol(7,nil,nil)),
                       arbol(9, nil, arbol(15, arbol(11, nil, nil), nil))),H).
   => H = [8,5,2,7,9,15,11]
```

9. ***(10 puntos)*** Implementar el predicado **siembra** en Prolog que a partir de una lista de números cree un árbol binario de búsqueda descrito con la función:

arbol(Raíz, SubárbolIzquierdo, SubárbolDerecho).

En un árbol binario de búsqueda, la raíz siempre es mayor o igual que los valores en su subárbol izquierdo mientras que es menor que los valores en su subárbol derecho.
Probar con:

```
?- siembra([],A)  => A = nil
?- siembra([2,3,1],A) => A = arbol(2,arbol(1,nil,nil),arbol(3,nil,nil))
?- siembra([8,5,2,7,9,15,11],A)
   => A = arbol(8,arbol(5,arbol(2,nil,nil),arbol(7,nil,nil)),
                  arbol(9, nil, arbol(15, arbol(11, nil, nil), nil)))
```