PLP - Segundo Recuperatorio - 1do cuatrimestre de 2024

#Orden	Libreta	Apellido y Nombre	Ej1	Ej2	Ej3	Nota Final
117.	64/22	Kerbs Octavio	B-	B-	B-	A

Este examen se aprueba obteniendo al menos dos ejercicios bien menos (B-) y uno regular (R). Las notas para cada ejercicio son: -, I, R, B-, B. 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) Representar en forma clausal las siguientes fórmulas de lógica de primer orden:

1. $\exists C.\forall X.(X \in a \Rightarrow X \leq C)$
   Existe una cota superior para el conjunto a.
2. $\forall X.\forall Y.(X \leq Y \Rightarrow n(Y) \leq n(X))$
   Si $X \leq Y$, entonces $-Y \leq -X$.
3. $\forall X.(X \in a \Leftrightarrow n(X) \in b)$
   X pertenece al conjunto a si y solo si -X pertenece al conjunto b.

b) Utilizando resolución, determinar si existe una cota inferior para el conjunto b:
$\exists C.\forall X.(n(X) \in b \Rightarrow C \leq n(X))$.

c) 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. No usar cut (!) ni predicados de alto orden como `setof`, con la única excepción de `not`.

a) Implementar el predicado `generarCapicúas(-L)`, que genera todas las listas capicúas de números naturales (sin incluir al cero). Por ejemplo:

```
?- generarCapicuas(L).
L = [1] ;
L = [2] ;
...
L = [1,3,1] ;
...
L = [2,3,3,2];
...
```

b) El predicado anterior ¿es reversible? Justificar.

c) Definir el predicado `tokenizar(+D,+F,-T)`, que es verdadero cuando `D` representa un diccionario de palabras conocidas (lista de listas de letras), `F` es una lista de letras sin espacios, y `T` es la lista de palabras en la que se puede partir la frase. El predicado debe generar todas las posibles maneras de dividir la frase en palabras conocidas. Por ejemplo:

```
?- tokenizar([[a],[a,b],[b,a],[c]], [a,b,a,c,a,b,a], T).
T = [[a],[b,a],[c],[a],[b,a]];
T = [[a],[b,a],[c],[a,b],[a]];
T = [[a,b],[a],[c],[a],[b,a]];
T = [[a,b],[a],[c],[a,b],[a]];
false.
```

d) Definir el predicado `mayorCantPalabras(+D,+F,-T)` que es verdadero cuando `T` es una tokenización de la frase `F` con el diccionario `D` que tiene la mayor cantidad de palabras. En este inciso no está permitido utilizar estructuras auxiliares.

Ejercicio 3 - Objetos y Deducción Natural

a) Considere las siguientes clases:

```
Object subclass: #A

    m1
    ^[self eval].

    eval
    ^self value.

    value
    ^0.
```

```
A subclass: #B

    m2
    ^[super eval].

    m3
    ^self.

    value
    ^1.

    eval
    ^2.
```

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) `B new m1 value`

II) `B new m3 m2 value`

b) Implementar un *closure* que tome como parámetro una colección y devuelva los elementos de la colección que saben responder al mensaje `#ptff`.
Sugerencia: pueden utilizar el mensaje `#respondsTo:`, que toma un mensaje y devuelve si el objeto sabe responderlo.

c) Demostrar en deducción natural que vale la siguiente fórmula, sin utilizar principios de razonamiento clásicos:

$$\exists X.P(X) \vee \forall Y.Q(Y) \implies \exists X.(Q(X) \vee P(X))$$