Guía de Trabajos Prácticos XII – Programación Lógica IV

1. Crear un programa en prolog que dada un conjunto de palabras (definidas como hechos mediante el predicado "p/1") y un patrón de búsqueda, retorne una lista con todas las palabras que coincidan con el patrón de búsqueda.

   Se puede tomar como ejemplo las siquientes declaraciones:
   p(árbol).
   p(animal).
   p(base).
   p(caramelo).
   p(caracol).
   p(curso).
   p(dedo).
   p(dedal).
   p(diferencia).
   p(diferencial).
   p(diferente).
   p(elefante).
   p(electricidad).
   p(esfera).
   p(fenómeno).
   p(fiera).
   p(guante).
   p(hormiga).
   p(hormigón).
   p(irlanda).

   El patrón de búsqueda será cualquier átomo que contenga letras en minúsculas y los símbolos '*' (representa una secuencia de cero o más letras no definidas), y '?' (representa una letra no definida).

   La búsqueda no debe diferenciar entre vocales con y sin acento y solo se manejarán letras en minúscula tanto en la definición como en el patrón de búsqueda.

   Ejemplos:
   ?- **buscar(*, L).**
   L = [árbol, animal, base, caramelo, caracol, curso, dedo, dedal, diferencia, diferencial, diferente, elefante, electricidad, esfera, fenómeno, fiera, guante, hormiga, hormigón, irlanda].

   ?- **buscar(arbol, L).**
   L = [árbol].

   ?- **buscar(arbol*, L).**
   L = [árbol].

?- **buscar(arbol?, L).**
L = [].

?- **buscar(ele*, L).**
L = [elefante, electricidad].

?- **buscar(ele*ad, L).**
L = [electricidad].

?- **buscar(*l, L).**
L = [árbol, animal, caracol, dedal, diferencial].

ATENCIÓN: Se puede usar cualquier predicado predefinido. En particular para pasar un átomo a lista de átomos de los caracteres que lo componen se puede usar el predicado "atom_chars/2", por ejemplo:
?- atom_chars(hola, L).
L = [h, o, l, a].

ATENCIÓN: En caso de ser necesario, utilizar el apóstrofe para deliminar y explicitar los átomos.

2. Genere un programa en prolog que resuelva el siguiente problema:

   teniendo una matriz de 3x3, se deben colocar todos y cada uno de los números de 1 al 9 de tal forma que la suma de cada una de las filas, cada una de las columnas y cada una de las diagonales principales dé el mismo número.

   Importante: NO USAR PREDICADOS PREDEFINIDOS.

3. Dado una lista de números enteros, crear un programa en prolog que compruebe si agregando operaciones aritméticas (+, -, *, /) y uno y solo un signo de igualdad (=) entre los números de la lista, se puede obtener una igualdad en la misma.

   Ejemplo:
   ?- igualdad([2, 3, 4, 2], X).

   X = [2, *, 3, =, 4, +, 2]
   X = [2, *, 3, -, 4, =, 2]

   NOTA 1: Validar que la lista contenga al menos dos elementos y solo contenga números enteros.

   NOTA 2: Para simplificar el desarrollo, no se debe tener en cuenta la precedencia de los operadores, es decir lo que se
   opera es le resultado parcial obtenido hasta ahora con el próximo número de la

lista.

NOTA 3: Al definir la operación a agregar, tener en cuenta que si el próximo número es cero, no se puede agregar una división.

Predicados aconsejados:

lista_valida/1: valida que el parámetro recibido sea una lista válida

lista_aritmetica/2: recibe como primer argumento una lista de enteros y retorna en el segundo la lista con los operadores aritméticos intercalados

operar/3: recibe en el primer argumento el resultado parcial, en el segunto una lista con el resto de la operación, y retorna en el tercer argumento el resultado final.

igualdad/2: recibe una lista de números enteros y retorna una lista con los números, los operadores y la igualdad.

4. Implemente en prolog los predicados que se describen a continuación teniendo en cuenta las siguientes consideraciones:

   - No se deben usar predicados predefinidos
   - Se estará trabajando con conjuntos
   - Los mismos se representarán como listas de elementos
   - Los conjuntos no pueden tener elementos repetidos, condición que hay que validar en los conjuntos de entradas y asegurar en el de salida

   1) union(C1, C1, U)
   - el predicado union/3 ejecuta la unión entre los conjuntos C1 y C2 y devuelve el resultado en el conjunto U.

   union([1, 2, 3], [3, 4, 5], U).
   U = [1, 2, 3, 4, 5].

   2) interseccion(C1, C1, U)
   - el predicado interseccion/3 ejecuta la intersección entre los conjuntos C1 y C2, y devuelve el resultado en el conjunto U.

   interseccion([1, 2, 3], [3, 4, 5], U).
   U = [3].

   3) diferencia(C1, C1, U)
   - el predicado diferencia/3 ejecuta la operación diferencia de conjuntos entre los conjuntos C1 y C2, y devuelve el resultado en el conjunto U.

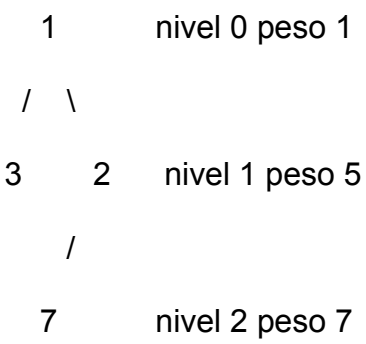
   diferencia([1, 2, 3], [3, 4, 5], U).

U = [1, 2].

IMPORTANTE: No utilizar predicados predefinidos y verificar que los conjuntos de entrada sean válidos, es decir que no contengan elementos repetidos, en caso de ser inválidos alguno de los dos, el predicado debe fallar.

5. Un árbol binario es vacío o consta de tres partes: la raíz (que debe ser un átomo), la rama izquierda (que debe ser un árbol binario) y la rama derecha (que debe ser un árbol binario). La raíz se denota por un átomo que debe ser único por árbol. Al árbol vació lo representamos por medio del átomo "nil".

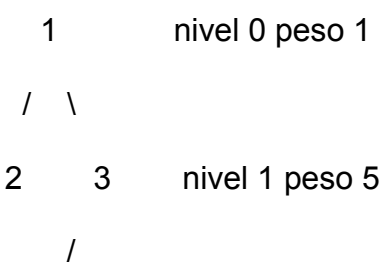
   * nil representa el árbol vacío.
   * t(I, R, D) representa el árbol de raíz R, rama izquierda I y rama derecha D.



   así el árbol del ejemplo sería: t(t(nil, 3, nil), 1, t(t(nil, 7, nil), 2, nil)).

   Se debe construir un programa en prolog que valide que la representación del árbol sea correcta:
   - que la forma del árbol sea correcta, es decir un nodo y dos ramas por nodo, las ramas pueden ser otro nodo o nil en caso de no existir
   - que cada nodo esté representado por un número "Entero"

   En caso de ser válido el árbol, se debe determinar el peso de un nivel en particular del mismo, siendo el peso del nivel la suma de los nodos del mismo. En caso de requerir un nivel mayor al de los contenidos por el árbol, devolver cero.



5 nivel 2 peso 5

?- peso_nivel(t(t(nil, 2, nil), 1, t(t(nil, 5, nil), 3, nil)), 0, Peso).
Peso = 1

?- peso_nivel(t(t(nil, 2, nil), 1, t(t(nil, 5, nil), 3, nil)), 2, Peso).
Peso = 5

?- peso_nivel(t(t(nil, 2, nil), 1, t(t(nil, 5, nil), 3, nil)), 3, Peso).
Peso = 0

1 nivel 0 peso 1

/ \

2 3 nivel 1 peso 5

/ \ / \

3 3 8 4 nivel 2 peso 18

?- peso_nivel(t(t(t(nil, 3, nil), 2, t(nil, 3, nil)), 1, t(t(nil, 8, nil), 3, t(nil, 4, nil))), 1, Peso).
Peso = 5

?- peso_nivel(t(t(t(nil, 3, nil), 2, t(nil, 3, nil)), 1, t(t(nil, 8, nil), 3, t(nil, 4, nil))), 2, Peso).
Peso = 18

?- peso_nivel(t(t(t(nil, 3, nil), 2, t(nil, 3, nil)), 1, t(t(nil, 8, nil), 3, t(nil, 4, nil))), 10, Peso).
Peso = 0