LABORATORIOS DE INTELIGENCIA ARTIFICIAL
CURSO 2016–2017

PRÁCTICA 3: PROLOG

Fecha de publicación: 2017/03/23

Fechas de entrega: grupos jueves: miércoles 2017/04/05, 23:55
grupos viernes: jueves 2017/04/06, 23:55

Planificación: Ejercicios para la 1.ª semana: 1, 2, 3 y 4.
Ejercicios para la 2.ª semana: 5, 6, 7 y 8.

Entrega: Para la entrega en formato electrónico, se creará un archivo zip que contenga todo el material de la entrega, cuyo nombre, todo él en minúsculas y sin acentos, tildes, o caracteres especiales, tendrá la siguiente estructura:

[gggg]_p3_[mm]_[apellido1]_[apellido2].zip

donde

- [gggg] : Número de grupo de la pareja: 2301, 2311, 2312, 2313 2361, 2362 o 2363)
- [mm] : Número de orden de la pareja con dos dígitos (posibles valores: 01, 02, 03,...)
- [apellido1] : Primer apellido del miembro1 de la pareja
- [apellido2] : Primer apellido del miembro2 de la pareja

Los miembros de la pareja aparecen en orden alfabético. Ejemplos :

- 2311_p3_01_delval_sanchezmontanyes.zip (práct. 3 de la pareja 01 en el grupo de prácticas 2311)
- 2362_p3_18_salabert_suarez.zip (práctica 3 de la pareja 18 en el grupo de prácticas 2362)

Contenido del fichero comprimido:

- El archivo de código Prolog: [gggg]_p3_[mm].pl
- El archivo: [gggg]_p3_[mm]_readme.txt [OPCIONAL]
- Los archivos de la memoria (nombre siempre en minúsculas): [gggg]_p3_[mm]_memoria.txt o [gggg]_p3_[mm]_memoria.pdf

---

EJERCICIO 1 (1 punto). Implementa un predicado `pertenece(X, L)` que compruebe si un elemento está en una lista. El predicado debe satisfacerse tantas veces como elementos repetidos contenga.

```
?- pertenece(1, [2, 1, 3, 1]).
true ;
true ;
false.
```

El predicado debe funcionar tanto si `X` como `L` no están instanciadas y debe generar por backtracking y en orden todas las soluciones:

```
?- pertenece(X, [2,1,3,1]).
X = 2 ;
X = 1 ;
```

```
X = 3 ;
X = 1 ;
false.

?- pertenece(1, L).
L = [1|_G12270] ;
L = [_G12269, 1|_G12273] ;
L = [_G12269, _G12272, 1|_G12276] ;
L = [_G12269, _G12272, _G12275, 1|_G12279] ;
L = [_G12269, _G12272, _G12275, _G12278, 1|_G12282]
```

Utiliza los predicados trace y notrace para trazar la ejecución de los objetivos anteriores, consulta el tutorial de Prolog para ver cómo se usan.

EJERCICIO 2 (1 punto). Implementa el predicado `invierte(L, R)` que se satisface cuando `R` contiene los elementos de `L` en orden inverso. Utiliza el predicado `concatena/3`:

```
concatena([], L, L).
concatena([X|L1], L2, [X|L3]) :-
    concatena(L1, L2, L3).
```

que se satisface cuando su tercer argumento es el resultado de concatenar las dos listas que se dan como primer y segundo argumento.

Ejemplos:

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

?- concatena([1, 2, 3], [4, 5], L).
L = [1, 2, 3, 4, 5].

?- invierte([1, 2], [L).
L = [2, 1].

?- invierte([], L).
L = [].

?- invierte([1, 2], L).
L = [2, 1].
```

EJERCICIO 3 (1 punto). Implementa los predicados `preorder(Tree, List)`, `inorder(Tree, List)` y `postorder(Tree, List)` que recorren un árbol binario y devuelven el recorrido en una lista. Para representar árboles usaremos las funciones `tree(Info, Left, Right)` y `nil`. También usaremos el predicado `concatena/3`. Ejemplo:

```
?- inorder(tree(1, tree(2, tree(3,nil,nil), tree(4,nil,nil)), tree(5,nil,nil)), L).
L = [3, 2, 4, 1, 5].

?- preorder(tree(1, tree(2, tree(3,nil,nil), tree(4,nil,nil)), tree(5,nil,nil)), L).
L = [1, 2, 3, 4, 5].
```

```
?- postorder(tree(1, tree(2, tree(3,nil,nil), tree(4,nil,nil)), tree(5, nil,nil)), L).
L = [3, 4, 2, 5, 1].
```

EJERCICIO 4 (1 punto). Implementar el predicado `insertar(X, L, P, R)` que inserte un elemento (`X`) en una lista (`L`) en una posición (`P`), desplazando el resto de elementos. Se considera que la primera posición de una lista es la 1. Ejemplos:

```
?- insertar(a, [1, 2], 0, L).
false.

?- insertar(1, [], 1, L).
L = [1] ;
false.

?- insertar(1, [], 4, L).
false.

?- insertar(a, [1, 2, 3, 4], 4, L).
L = [1, 2, 3, a, 4] ;
false.

?- insertar(a, [1, 2, 3, 4], 5, L).
L = [1, 2, 3, 4, a] ;
false.
```

EJERCICIO 5 (1 punto). Implementar el predicado `extract(L1, X, L2)` que extrae un elemento (`X`) de una lista (`L1`) y deja en otra lista (`L2`) la lista original (`L1`) sin el elemento.

```
?- extract([1, 2, 3], X, L).
X = 1, L = [2, 3] ;
X = 2, L = [1, 3] ;
X = 3, L = [1, 2] ;
false.

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

EJERCICIO 6 (1 punto). Se conoce como el **problema de los matrimonios estables** (*stable marriage problem*[1]) al problema de emparejamiento de *n* hombres y *n* mujeres de manera sus emparejamientos sean estables. Tanto hombres como mujeres expresan sus preferencias mediante listas ordenadas por preferencia decreciente. Suponiendo un conjunto de hombres {**juan**, **pedro**, **mario**} y otro de mujeres {**maria**, **carmen**, **pilar**}, Las preferencias de hombres por mujeres y viceversa se expresaran en Prolog mediante los hechos `man_pref(X,L)` y `woman_pref(X,L)`, donde `X` es un hombre o una mujer y `L` es su lista de preferencias en orden decreciente. Representad las dos siguientes tablas de preferencias con predicados correspondientes:

[1] https://en.wikipedia.org/wiki/Stable_marriage_problem

M	maria	carmen	pilar
juan	1	2	3
pedro	2	1	3
mario	1	2	3

W	juan	pedro	mario
maria	1	2	3
carmen	2	1	3
pilar	1	2	3

El problema de los matrimonios estables ha tenido un papel fundamental en teoría económica para mercados con oferta y demanda en los que los precios no existen. El desarrollo teórico y su aplicación práctica le valieron el premio Nobel de economía a Lloyd Shapley y Alvin Roth[2]. Entre sus aplicaciones prácticas más relevantes se encuentran: la asignación de estudiantes y colegios en Nueva York, la asignación de médicos residentes y hospitales americanos, y la asignación de órganos para trasplantes a pacientes receptores.

EJERCICIO 7 (2 puntos). Representaremos con el operador infijo `-` a los matrimonios. Los operandos serán variables o átomos Prolog. Podemos unificar matrimonios de la siguiente manera:

```
?- X-Y = juan-maria.
X = juan,
Y = maria.

?- mario-Y = juan-maria.
false.
```

Implementa un predicado `unstable(M1-W1, Marriages)` que compruebe si un nuevo matrimonio (`M1-W1`) es **inestable** respecto a una lista de matrimonios estables (`Marriages`). Diremos que un matrimonio $M_1 - W_1$ es **inestable** si $\exists\ M_2 - W_2 \in Marriages \mid M_1$ prefiere W_2 a W_1 y W_2 prefiere M_1 a M_2. El predicado deberá acceder a los predicados `man_pref/2` y `woman_pref/2` y obtener las preferencias mediante el predicado `pos/3` visto en clase para compararlas. Usa también `pertenece/3`. Ejemplo:

```
?- unstable(juan-carmen, [pedro-maria]).
true ;
false.

?- unstable(juan-maria, [pedro-carmen]).
false.
```

EJERCICIO 8 (2 puntos). Implementa el predicado `smp(FreeMen, FreeWomen, MarriagesIn, MarriagesOut)` que calcule para unas listas de hombres (`FreeMen`) y mujeres (`FreeWomen`) sus

[2] http://www.nobelprize.org/nobel_prizes/economic-sciences/laureates/2012/popular-economicsciences2012.pdf

matrimonios estables. En la primera llamada, la lista de matrimonios iniciales (`MarriagesIn`) debe estar vacía. Implementa este predicado de manera que sean las mujeres quienes elijan a los hombres y utiliza el predicado predefinido `not/1` combinado con `unstable/2` para comprobar que un matrimonio es estable.

```
?- smp([juan, pedro, mario], [maria, carmen, pilar], [], Marriages).
Marriages = [mario-pilar, pedro-carmen, juan-maria] ;
false.
```

[]