Memoria Práctica 2

Programación Declarativa Lógica y Restricciones

FECHA: 10/06/19

AUTORES

Alejandro Benavente Álvarez	y160319	alejandro.benavente.alvarez@alumnos.upm.es
David Cristóbal Pascual	y160336	d.cristobal@alumnos.upm.es
Alberto Doncel Aparicio	y160364	a.daparicio@alumnos.upm.es

CÓDIGO

```
%Nuestros datos
alumno_prode(Benavente,Alvarez,Alejandro,160319).
alumno_prode(Cristobal,Pascual,David,160336).
alumno_prode(Doncel,Aparicio,Alberto,160364). %PORTAVOZ
%--------------------------------

%1

% menor/4(num1, num2, Comp, Solucion) devuelve en M el menor entre A y B siguiendo
Comp / true si M es el menor de A y B
menor(A,B,Comp,M):-
number(A), % Comprueba si A es un numero
number(B), % Comprueba si B es un numero
X=..[Comp,A,B], % Crea una función con Comp entre A y B
call(X), % X es Goal
M is A. % Guarda A en M si A es el menor

% menor/4 devuelve en M el menor entre A y B siguiendo Comp / true si M es el menor de
A y B
menor(A,B,Comp,M):-
number(A),
number(B),
X=..[Comp,B,A],
call(X),
M is B. % Guarda B en M si B es el menor

%2.

% comp_rec/2(Lista1, Lista2)
% true si cada elemento de la Lista1 es <= que el mismo elemento de la Lista2
comp_rec([],[]).
comp_rec([X|Xs],[Y|Ys]):-
menor_o_igual(X,Y),
comp_rec(Xs,Ys).

% menor_o_igual(Elemento1, Elemento2)
% true si Elemento1 <= Elemento2 asi:
% variable libre es igual a otro termino
% un termino es menor que otro (ninguno variable libre)
% si su nombre es @< que el otro
% nombres identicos y aridad menor
% si nombre y aridad iguales se comparan de izquierda a derecha los terminos
menor_o_igual(A,_):-
var(A). % comprueba que E1 es una variable libre

menor_o_igual(_,A):-
```

```
var(A). % comprueba que E2 es una variable libre

menor_o_igual(A,B):-
functor(A,AF,_),
functor(B,BF,_),
AF @< BF. % comprueba si un el nombre de E1 es @< que el de E2

menor_o_igual(A,B):-
functor(A,_,AA),
functor(B,_,BA),
AA < BA. % comprueba si la aridad de E1 es menor que la de E2

menor_o_igual(A,B):-
functor(A,AF,AA),
functor(B,BF,BA),
AF == BF, % comprueba que los nombres de E1 y E2 sean iguales
AA == BA, % comprueba que las aridades de E1 y E2 sean iguales
A=..[_|X], % se quita la funcion para obtener el resto de elementos de E1
B=..[_|Y], % se quita la funcion para obtener el resto de elementos de E2
comp_rec(X,Y). % compara los elementos con comp_rec/2.

%3
ordenar(Lista,Comp,Orden).

% lista_hojas/2(Lista, Hojas)
% true si hojas es la lista Lista transformada en una lista de hojas
% devuelve en Hojas -> Lista transformada en hojas
lista_hojas([],[]). %una lista vacia devuelve una lista vacia
lista_hojas([A],Hojas):-
Hojas=[tree(A,void,void)]. %caso donde hay en la lista solo hay un elemento y lo anyade
en forma de hoja a nuestra lista de hojas
lista_hojas(Lista,Hojas):-
Lista=[X|Lista2], %separamos la cabeza de la lista del resto de la lista
Hojas=[tree(X,void,void)|Hojas2], %anadimos una hoja con la cabeza extraida de la lista
anterior en nuestra lista de hojas
lista_hojas(Lista2,Hojas2). %buscamos las demas hojas para anadir
hojas_arbol(Hojas, Comp, Arbol).

%ordenacion([A],Comp,Orden).

ordenacion(Arbol,Comp,Orden).
```

Justificaciones

Para la realización de la práctica como ayuda buscamos información contenida en la bibliografía (2) que nos ayudó a comprender mejor los operadores para las comparaciones.

CONSULTAS

menor/2

?- **menor(1,5,=<,M).**

M = 1 ? ;

no

?- **menor(6,1,=<,M).**

M = 1 ? ;

no

?- **menor(2,2,=<,3).**

no

?- **menor(50,2,=<,2).**

Yes

menor_o_igual/2

?- **menor_o_igual(A,_).**

yes

?- **menor_o_igual(_,A).**

yes

?- **menor_o_igual(q,q).**

yes

?- **menor_o_igual(p(x),q).**

yes

?- **menor_o_igual(p(x),p).**

no

?- **menor_o_igual(p(_,_,_),p(X)).**

no

?- **menor_o_igual(p(a,b,c),q).**

yes

?- **menor_o_igual(p(y),p(y)).**
yes

?- **menor_o_igual(p(p(x)),p(q)).**
yes

?- **menor_o_igual(p(p(x),f(A),p(q),p(q,X,Y,Z)),p(A,B,C,D)).**
Yes

?- **menor_o_igual(p(1,1,q(po(1))),p(1,1,q(pa(2)))).**

no

lista_hojas/2

?- **lista_hojas([],[]).**

yes

?- **lista_hojas([],X).**
X = [] ? **;**
no

?- **lista_hojas([A],X).**
X = [tree(A,void,void)] ? **;**
no

?- **lista_hojas([1,2,3,4,5],X).**
X = [tree(1,void,void),tree(2,void,void),tree(3,void,void),tree(4,void,void),tree(5,void,void)] ? **;**
no

?- **lista_hojas([A, p(q), p, p(a,b,c), p(a,b,c,d,e)],X).**
X =
[tree(A,void,void),tree(p(q),void,void),tree(p,void,void),tree(p(a,b,c),void,void),tree(p(a,b,c,d,e),void,
void)] ? ;
no

COMENTARIOS ADICIONALES

Dificultades

Las principales dificultades que hemos tenido a la hora de realizar este trabajo ha sido trabajar con árboles dado que no nos ha terminado de quedar claro su funcionamiento y cómo trabajar con ellos.

Esto nos ha impedido terminar la práctica y nos obliga a entregarla sin terminar.

Material empleado

Para la realización de esta práctica nos hemos valido de dos editores de texto Emacs y Visual Studio Code en los que hemos escrito y probado el código empleando siempre Ciao para ejecutarlo.
El control de versiones lo hemos llevado a cabo con un repositorio privado de GitHub. Nos ha ayudado a trabajar con orden y a poder volver hacia atrás cuando hemos tenido cualquier clase de problema.
Finalmente, este documento, está realizado con LibreOffice Writer.
A parte, además de los apuntes oficiales, nos hemos ayudado de diversas fuentes expuestas en la sección BIBLIOGRAFÍA.

BIBLIOGRAFÍA

(1) Programación declarativa, http://cliplab.org/prode/, [Comentario: página web oficial donde hemos consultado los apuntes oficiales.].

(2) Comparison operators, https://www.cse.unsw.edu.au/~billw/dictionaries/prolog/comparison.html, [Comentario: página web donde nos hemos informado de los distintos operadores.].