2018-2019

PRÁCTICA 1: PROGRAMACIÓN LÓGICA PURA

Participantes:

Ciprian Ilut 160348

Carlos Mateos Martín 160185

Javier del Río García 160451

Código

```
%%%%%%%%%%%%%%%%%ALUMNOS DEL GRUPO%%%%%%%%%%%%%%%%%

alumno_prode(ilut, xxxxxx,ciprian,160348).

alumno_prode(mateos,martin,carlos,160185).

alumno_prode(del_rio,garcia,javier,160451).

%%%%%%%%%%%%%%%%%ALUMNOS DEL GRUPO%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%FUNCIONES AUXILIARES%%%%%%%%%%%%%%%%%

igual(X,X).

% Predicado que se verifica cuando los dos argumentos son iguales.


nat(0).

nat(s(X)) :- nat(X).

% Predicado que reconoce números naturales. Utilizado para comprobar las dimensiones de
las construcciones.


esPar(0).

esPar(s(s(X))):- esPar(X).

% Predicado que se verifica si el argumento es par. Utilizado principalmente para esEdificioPar


menorIgualNr(0, X) :- nat(X).

menorIgualNr(s(X), s(Y)) :- menorIgualNr(X,Y).

% Predicado que se verifica si el primer argumento es menor o igual que el segundo.


menorNr(0,s(X)) :- nat(X).

menorNr(s(X),s(Y)) :- menorNr(X,Y).

%Predicado que se verifica si el primer argumento es menor que el segundo.
```

```
suma(0,X,X).

suma(s(X),Y,s(Z)):- suma(X,Y,Z).

% Predicado que unifica en Z la suma de los dos primeros argumentos.


unAtributo(pieza(Ancho, Alto, Prof, Color),Ancho, Alto,Prof, Color).

% Predicado que nos permite conocer uno o varios de los atributos con los que cuenta una pieza.


dosCabeza([Cab,Cab1|Lista],Cab, Cab1).

% Predicado que nos permite conocer los dos primeros elementos de una lista


unaCabeza([Cab|Lista],Cab).

% Predicado que nos permite conocer el primer elemento de una lista.


meterCabeza(Cab,Lista, [Cab|Lista]).

% Predicado que nos permite añadir un elemento a una lista.


eliminarCabeza([Cabeza|Construccion],Construccion).

% Predicado que nos permite eliminar el primer elemento de una lista.


menorIgualPiezaDC(pieza(An0,Al0,Pr0, C0),pieza(An1,Al1,Pr1, C1)) :-

        menorIgualNr(An0,An1),

        menorIgualNr(Pr0,Pr1).

% Predicado que se verifica cuando la primera pieza pasada como argumento tiene un ancho y profundidad menor.


%%%%%%%%%%%%%%%%FUNCIONES AUXILIARES%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%ESTRUCTURAS%%%%%%%%%%%%%%%%
```

```
color(r).

color(a).

color(v).

color(am).

colorB(b).
```

% Hechos que indican los colores válidos.

```
pieza(Ancho, Alto, Prof, Color) :-
        menorNr(0,Ancho),
        menorNr(0,Alto),
        menorNr(0,Prof),
        nat(Ancho),
        nat(Alto),
        nat(Prof),
        color(Color).
```

% Predicado que se verifica si los parámetros cumplen las condiciones para ser una pieza:

- El ancho es mayor que 0
- El alto es mayor que 0
- La profundidad es mayor que 0
- Todos estos últimos valores son naturales
- El color es uno de los válidos especificados anteriormente

```
construccion([pieza(An, Al, Prof, C)]):-pieza(An,Al,Prof,C).

construccion([pieza(An, Al, Prof, C)|L]) :- pieza(An,Al,Prof,C),construccion(L).
```

% Predicados que permiten conocer si una lista de elementos es una construcción. Es decir, si todos los elementos cumplen con las condiciones para ser una pieza.

```
construccionFila([C]):-color(C).

construccionFila([C]):-colorB(C).
```

```
construccionFila([C|L]) :-color(C),construccionFila(L).

construccionFila([C|L]) :-colorB(C),construccionFila(L).

% Predicados utilizados en edificio. Permiten conocer si las filas están compuestas por colores válidos.

%%%%%%%%%%%%%%%ESTRUCTURAS%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%esTorre%%%%%%%%%%%%%%%

esTorre([pieza(An, Al, Prof, C)]):-pieza(An,Al,Prof,C).

esTorre(Construccion) :-
        construccion(Construccion),
        dosCabeza(Construccion, P1, P2),
        menorIgualPiezaDC(P1, P2),
        eliminarCabeza(Construccion, ConstruccionCopia),
        esTorre(ConstruccionCopia).

% Este predicado nos permite conocer si una construcción es una torre. Para ello, se realiza de manera recursiva una comprobación de los dos primeros elementos de la lista. Si el segundo es mayor que el primero, se elimina este último de la lista y se vuelve a llamar al predicado.

%%%%%%%%%%%%%%%esTorre%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%alturaTorre%%%%%%%%%%%%%%%

alturaTorre(Construccion, A) :-
        esTorre(Construccion),
        eliminarCabeza(Construccion, ConstruccionCopia),
        unaCabeza(Construccion, P),
        unAtributo(P, _,V,_,_),
        contarAltura(ConstruccionCopia, V, A).
```

```
contarAltura([],A,A).

contarAltura(Construccion, B,A):-

        eliminarCabeza(Construccion, ConstruccionCopia),

        unaCabeza(Construccion, C),

        unAtributo(C, _,V,_,_),

        suma(V,B,Aux),

        contarAltura(ConstruccionCopia, Aux, A).
```

% Los predicados anteriores se utilizan para averiguar la altura de una torre. El primero de ellos, alturaTorre, llama al predicado contarAltura que sumará de manera recursiva la altura de cada una de las piezas que se encuentren en la construcción.

```
%%%%%%%%%%%%%%%alturaTorre%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%coloresTorre%%%%%%%%%%%%%%%

coloresTorre(Construccion, Colores) :-

        esTorre(Construccion),

        unaCabeza(Construccion, P),

        unAtributo(P, _,_,_,C),

        eliminarCabeza(Construccion, ConstruccionCopia),

        meterCabeza(C, [],CAux),

        guardarColores(ConstruccionCopia, CAux, Colores).


invertirColores([], Colores, Colores).

invertirColores(CAux, CAux1, Colores):-

        unaCabeza(CAux, P),

        eliminarCabeza(CAux, CAux3),

        meterCabeza(P, CAux1, CAux2),

        invertirColores(CAux3, CAux2, Colores).
```

```
guardarColores([], CAux,Colores):-
        invertirColores(CAux, [], Colores).
guardarColores(Construccion, CAux, Colores):-
        unaCabeza(Construccion, P),
        unAtributo(P, _,_,_,C),
        eliminarCabeza(Construccion, ConstruccionCopia),
        meterCabeza(C, CAux, CAux1),
        guardarColores(ConstruccionCopia, CAux1, Colores).
```

% Este conjunto de predicados se utiliza para conocer los colores de las piezas que conforman una torre. Se comienza llamando al predicado principal, coloresTorre, que pasará a llamar al predicado guardarColores, igual que el primero pero que nos permitirá unificar la lista de colores de las piezas que vamos recorriendo. Por último, ya que se han ido incluyendo los colores en la cabeza de una lista auxiliar, se llama al predicado invertir colores para que queden en el mismo orden que en la torre original.

```
%%%%%%%%%%%%%%%coloresTorre%%%%%%%%%%%%%%%
```

```
%%%%%%%%%%%%%%%coloresIncluidos%%%%%%%%%%%%%%%

coloresIncluidos(Construccion1,Construccion2) :-
        esTorre(Construccion1),
        esTorre(Construccion2),
        coloresTorre(Construccion1, Colores1),
        coloresTorre(Construccion2, Colores2),
        contieneColores(Colores1,Colores2).

comparaColor(C1,C2,A,B) :-
        igual(C1,C2),
          A=s(0).
```

```
comparaColor(C1,C2,A,B) :-
        B=s(0).

comparaConResto(B,[],A):-
    igual(A,s(0)).
comparaConResto(C1, Colores2,A):-
    unaCabeza(Colores2,C2),
    igual(C1,C2),
    comparaConResto(C1,[],s(0)).
comparaConResto(C1, Colores2,A):-
    unaCabeza(Colores2,C2),
     eliminarCabeza(Colores2,Colores2Aux),
     comparaConResto(C1,Colores2Aux,s(s(0))).

contieneColores([],B).
contieneColores(Colores1, Colores2):-
    unaCabeza(Colores1, C1),
    eliminarCabeza(Colores1, Colores1Aux),
    comparaConResto(C1, Colores2,M),
    contieneColores(Colores1Aux,Colores2).
```

% Primeramente se consigue la lista de colores de cada torre. A continuación, se extrae un elemento de la lista de colores y se compara con la otra lista. Si está, se elimina ese elemento y se vuelve a realizar otra comparación.

Se cuentan con dos predicados comparaConResto que tienen la misma cabecera para evitar el backtracking y el uso de ';' en las consultas.

```
%%%%%%%%%%%%%%%coloresIncluidos%%%%%%%%%%%%%%%%
```

```
%%%%%%%%%%%%%%%esEdificioPar%%%%%%%%%%%%%%%%
esEdificioPar([Fila]):-
```

```
        construccionFila(Fila),
        NrClavosAux = 0,
        nrClavos(Fila, NrClavosAux,NrClavos),
        esPar(NrClavos),
        eliminarCabeza(Construccion, ConstruccionAux).
esEdificioPar(Construccion) :-
        unaCabeza(Construccion,Fila),
        construccionFila(Fila),
        NrClavosAux = 0,
        nrClavos(Fila, NrClavosAux,NrClavos),
        esPar(NrClavos),
        eliminarCabeza(Construccion, ConstruccionAux),
        esEdificioPar(ConstruccionAux).

nrClavos([], NrClavosAux,NrClavos):-
        menorIgualNr(s(0),NrClavosAux),
        igual(NrClavosAux,NrClavos).
nrClavos([],NrClavosAux,NrClavos):-
        igual(NrClavosAux, 0),
        igual(0,NrClavos).

nrClavos(Fila,NrClavosAux,NrClavos):-
        unaCabeza(Fila,C),
        eliminarCabeza(Fila,FilaAux),
        igual(C,b), nrClavos(FilaAux,NrClavosAux, NrClavos).

nrClavos(Fila,NrClavosAux,NrClavos):-
```

```
        unaCabeza(Fila,C),
        suma(s(0),NrClavosAux, NrClavosAux1),
        eliminarCabeza(Fila,FilaAux),
        nrClavos(FilaAux,NrClavosAux1,NrClavos).
```

% En este predicado se toma una construcción, se cuenta el número de clavos de una fila y, si es par, se elimina dicha fila de la lista y se pasa a comprobar la siguiente fila hasta que ya no queden más elementos. Si llegado a este punto todas las filas tenían un número par de clavos, el edificio es par. Se cuentan con dos predicados que diferencian si el argumento es una fila (lista) o es una lista de filas (matriz).

```
%%%%%%%%%%%%%%%esEdificioPar%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%esEdificioPiramide%%%%%%%%%%%%%%%
esEdificioPiramide([Fila]):-
        construccionFila(Fila),
        NrClavosAux=0,
        nrClavos(Fila,NrClavosAux,NrClavos),
    menorNr(s(0),NrClavos).

esEdificioPiramide(Construccion):-
        dosCabeza(Construccion,C1,C2),
        construccionFila(C1),
        construccionFila(C2),
        NrClavosAux=0,
        nrClavos(C1,NrClavosAux,NrClavos1),
        nrClavos(C2,NrClavosAux,NrClavos2),
        menorNr(NrClavos1,NrClavos2),
        eliminarCabeza(Construccion,ConstruccionAux),
        esEdificioPiramide(ConstruccionAux).
```

% Se comprueba cada nivel y se cuenta el número de clavos. Después, se elimina esa fila ya visitada y se pasa a la siguiente, dónde se volverán a contar los clavos. Si el número de la primera fila es menor que la segunda, se sigue comprobando.

Al igual que en esEdificioPar, tenemos dos predicados iguales que diferencian entre construcción y fila.

Consultas al programa

esTorre/1

?- esTorre([pieza(s(0),s(0),s(0),r)]).

Yes

?- esTorre([pieza(s(0),s(0),s(0),r),pieza(s(0),s(0),s(0),a)]).

Yes

?- esTorre([pieza(s(0),s(0),s(0),r),pieza(s(0),s(s(0)),s(0),r),pieza(s(s(0)),s(s(0)),s(s(0)),v),pieza(s(s(s(0))),s(s(s(0))),s(s(s(0))),am)]).

Yes

?- esTorre([pieza(0,s(0),0,r)]). % Pieza con Anchura y Profundidad 0

No

?- esTorre([pieza(s(0),s(s(s(0))),s(s(0)),r),pieza(s(0),s(s(0)),s(0),r)]). % Pieza superior con dimensiones mayores

no

alturaTorre/2

?- alturaTorre([pieza(s(0),s(0),s(0),r),pieza(s(0),s(0),s(0),a)],s(s(0))).

Yes

?- alturaTorre([pieza(s(0),s(s(s(0))),s(0),r),pieza(s(0),s(s(s(0))),s(0),r)],s(s(s(s(s(s(0)))))).

yes

?-
alturaTorre([pieza(s(0),s(0),s(0),r),pieza(s(0),s(s(0)),s(0),r),pieza(s(0),s(s(s(0))),s(0),r)],s(s(s(s(s(s(0)))))).

Yes

?- alturaTorre([pieza(s(0),s(0),s(0),r),pieza(s(0),s(s(0)),s(0),r),pieza(s(0),s(s(s(0))),s(0),r)],X). % Misma prueba pero esta vez para conocer la altura en vez de darla como argumento.

X = s(s(s(s(s(s(0)))))) ?

coloresTorre/2

?- coloresTorre([pieza(s(0),s(0),s(0),a)],[a]).

Yes

?- coloresTorre([pieza(s(0),s(0),s(0),r),pieza(s(0),s(0),s(0),r),pieza(s(0),s(0),s(0),a)],[r,r,a]).

Yes

?-
coloresTorre([pieza(s(0),s(0),s(0),v),pieza(s(0),s(0),s(0),r),pieza(s(0),s(0),s(0),r),pieza(s(0),s(0),s(0),r),pieza(s(0),s(0),s(0),a)],X). % Prueba para obtener la lista de colores

X = [v,r,r,r,a] ?

?- coloresTorre([pieza(s(0),s(0),s(0),r),pieza(s(0),s(0),s(0),r),pieza(s(0),s(0),s(0),a)],[r,a]).
%Prueba en el que se le pasa una lista errónea como argumento

no

coloresIncluidos/2

?-
coloresIncluidos([pieza(s(0),s(0),s(0),r)],[pieza(s(0),s(0),s(0),r),pieza(s(0),s(0),s(0),v),pieza(s(0),s(0),s(0),a)]).

Yes

?-
coloresIncluidos([pieza(s(s(0)),s(0),s(0),r)],[pieza(s(0),s(0),s(0),r),pieza(s(0),s(0),s(0),v),pieza(s(0),s(0),s(0),a)]).

Yes

?-
coloresIncluidos([pieza(s(0),s(0),s(0),r),pieza(s(0),s(0),s(0),v),pieza(s(0),s(0),s(0),a)],[pieza(s(s(0)),s(0),s(0),r)]). % Prueba con una lista construcción 1 con más colores que construcción 2

no

esEdificioPar/1

?- esEdificioPar([[b,b,r,r,b,b],[b,v,v,v,v,b],[b,b,v,v,b,b]]).

Yes

?- esEdificioPar([[b,b,r,r,b,b],[b,v,v,v,v,b],[b,b,v,v,b,b],[a,a,am,a,a,a],[b,b,am,am,b,b]]).

Yes

?- (esEdificioPar([[b,r,r,r,r,b],[b,v,v,v,v,b],[b,b,v,v,b,b],[a,a,a,am,a,a,a],[b,b,am,b,b,b]])).
%Prueba con el último nivel impar

no

esEdificioPiramide/1

?- esEdificioPiramide([[b,b,r,b,b],[b,r,a,am,b],[r,r,r,r,r]]).

yes

?-
esEdificioPiramide([[b,b,b,b,v,b,b,b,b],[b,b,b,r,v,r,b,b,b],[b,b,r,a,am,v,a,b,b],[b,r,r,r,r,r,r,r,b],[a
m,r,r,r,r,r,r,r,am]]).

Yes

?- esEdificioPiramide([[r,r,p,p,r,r],[a,b,b,b,am,v],[r,b,b,b,b,b]]). % Prueba con un edificio que no cumple la condición de pirámide

no