UNIVERSIDAD POLITÉCNICA DE MADRID

Programación Declarativa: Lógica Pura

Grado Matemáticas e informática

GARCÍA MARTÍN, IGNACIO **u12m071**
MOYA LÓPEZ, AITOR **v13m022**
RANILLA CORTINA, SANDRA **v13m046**
ROSADO GONZÁLEZ, RODRIGO **v13M018**

15/4/2016

Índice

Autómata Celular

- Código y Explicación

Para el desarrollo de este ejercicio de la práctica hemos diseñado el siguiente código de Programación Lógica Pura:

Tras varias lecturas del enunciado y una breve explicación en el aula del ejercicio hemos decidido construir el esqueleto del código con 7 predicados, los cuales se observan a continuación.

El primer predicado construido, **regla/4**, describe todas las reglas de cambio de color de las células, teniendo como argumentos la célula en sí, dos vecinas de esta misma y por último, su nuevo color:

```
% " Reglas "
regla(o,o,o,o).
regla(x,o,o,x).
regla(o,x,o,o).
regla(o,o,x,x).
regla(x,o,x,x).
regla(x,x,o,x).
regla(o,x,x,x).
regla(x,x,x,o).
```

Para seguir, hemos desarrollado 3 predicados, **devuelve1/2**, **devuelve2/2** y **devuelve3/2**, que sirven para devolver respectivamente el primer, el segundo y el tercer elemento de una lista, y en conjunto los podríamos contar como un único predicado, puesto que el fin es que los 3 juntos nos permitan manejarnos con **regla/4**:

```
% " Metodo auxiliar devuelve 3 elementos contiguos de una lista "
devuelve1([X|Xs],X).
devuelve2([X|Xs],Y):-devuelve1(Xs,Y).
devuelve3([X|Xs],Z):-devuelve2(Xs,Z).
```

Para poder concatenar listas hemos definido el predicado **append1/3** el cual dadas dos listas las concatene en el tercer argumento. Este predicado lo usamos en **cells1/2**, para concatenar los colores obtenidos mediante **regla/4**, es decir, los elementos de lista final:

```
% " Metodo auxiliar append "
append1([],L,L).
append1([X|Y],L,[X|L1]):-append1(Y,L,L1).
```

Para continuar, el penúltimo predicado desarrollado es **cells1/2**, predicado auxiliar del principal, **cells/2**. En este, lo que hacemos es de manera recursiva llamar a **append1/3**, **regla/4**, **devuelve/2** y **devuelve/3**, para a partir de una lista-secuencia dada hacer el cambio de color de cada terna de la secuencia, teniendo como caso base el final de la lista, para el cual la lista-secuencia inicial y la final siempre deben terminar en **xo** ó **oo**:

```
% " Predicado auxiliar Cells "
cells1([x,o],[x,o]).
cells1([o,o],[o]).
cells1([X|Xs],[W|Yt]):-
    append1([W],Ys,[Y|Yt]),
    cells1(Xs,Ys),
    regla(X,C,S,W),
    devuelve2([X|Xs],C),
    devuelve3([X|Xs],S).
```

Por último, construimos el predicado **cells/2** en el cual se tratan los casos de comienzo de secuencia que deben ser en **ox** ó **oo** tanto la inicial como la final y se llama a **cells/1**, predicado que realiza el resto del cambio de color y finalmente se obtiene la pedida:

```
% " Predicado Cells "
cells([],[]).
cells([X|L1],L2):-
    devuelve1([X|L1],o),
    devuelve2([X|L1],x),
    append1([o,x],L3,L2),
    cells1([X|L1],L3).

cells([X|L1],L2):-
    devuelve1([X|L1],o),
    devuelve2([X|L1],o),
    append1([o],L3,L2),
    cells1([X|L1],L3).
```

- Consultas

Se proporciona primer y segundo argumento

```
1 ?- cells([o,x,x,o,x,x,o],[o,x,x,x,x,x,x,x,o]).
true ;
```

En esta consulta comprobamos que el predicado **cells/2** construido comprueba de manera correcta si se corresponden el cambio de color de la lista dada como primer argumento con el resultado que viene dado como segundo argumento.

Se proporciona primer argumento y no segundo argumento

```
1 ?- cells([o,x,x,x,o,x,o],L).
L = [o, x, x, o, x, x, o, x, o] ;
```

En esta consulta vemos que el predicado devuelve de manera correcta dada una lista-secuencia el resultado de aplicar el cambio de color. Esta consulta es similar al que viene en el enunciado de la práctica

No se proporciona primer argumento y si segundo argumento

```
1 ?- cells(L,[o,x,o,x,o,x,o,x,o]).
L = [o, x, o, o, o, x, o] ;
L = [o, x, o, x, o, x, o] ;
L = [o, x, o, o, o, x, o, o] ;
L = [o, x, o, x, o, x, o, o] ;
L = [o, o, x, o, o, o, x, o] ;
L = [o, o, x, o, x, o, x, o] ;
L = [o, o, x, o, o, o, x, o, o] ;
L = [o, o, x, o, x, o, x, o, o] ;
```

En este caso, realizamos el proceso inverso, dada una lista con el cambio ya realizado se nos devuelven todas las posibles secuencias, las cuales aplicándoles el cambio de color darían como resultado la lista indicada como segundo argumento.

Secuencia no válida

```
1 ?- cells([x,o,x,o,x,x,x,o],L).
false.

2 ?- cells([o,x,o,x,x,x,o,o,x],L).
false.
```

Por último, visualizamos dos ejemplos por los cuales el predicado indica fallo. El primero devuelve falso, puesto que la secuencia empieza por xo y hemos establecido, por el enunciado, que una secuencia siempre debe comenzar por oo ó ox.

Para el segundo caso, falla porque la secuencia termina por ox y al igual que en el caso anterior, hemos establecido que una secuencia siempre debe terminar por oo ó xo.

- Comentarios

En el desarrollo de este ejercicio hemos encontrado diferentes dificultades, entre las que prima el entendimiento del predicado cells/2, una vez esto fue comprendido el resto de dificultades tienden a ser de familiarización con Ciao Prolog.

En lo que se refiere al enunciado y desarrollo creemos haber entendido correctamente el funcionamiento del predicado cells/2. Visualizando la secuencia como un autómata que tiene que empezar y finalizar por ox ó oo y xo ó oo respectivamente.

Tomando ternas de tres elementos, de principio a fin de la secuencia, se aplican las reglas de cambio de color, todo el procedimiento de reglas obviamente se realiza de manera recursiva mediante el predicado auxiliar cells1/2.

Codificación Huffman Plog

- Código y Explicación

Para el desarrollo de este ejercicio de la práctica hemos diseñado el siguiente código de Programación Lógica Pura:

Tras varias lecturas del enunciado y una breve explicación en el aula del ejercicio hemos decidido separar este ejercicio en dos apartados diferenciados, uno para el desarrollo de la primera parte del algoritmo y uno para la segunda parte, en total se observan a continuación 15 predicados.

Primero, mostramos los predicados auxiliares utilizados para el desarrollo tanto del árbol balanceado como el árbol amplificado. En total tendremos 11 predicados auxiliares:

El primer predicado construido, **suma/3**, que realiza la suma de dos números y lo guarda en el tercer argumento, como predicado auxiliar utiliza **nat/1** que describe los números naturales de Peano, como argumento tiene el número en sí:

```
% " Suma de numeros naturales " %
nat(0).
nat(s(X)):-nat(X).

suma(0,X,X):-nat(X).
suma(s(X),Y,s(Z)):-suma(X,Y,Z).
```

A continuación mostramos el predicado, mod/3, que describe el módulo, como argumentos tiene, el número en sí del que se va a hacer el módulo, la base sobre la que se aplica y el resultado. Como predicado auxiliar para este usamos también el predicado menor/2 que nos dice si un número es menor que otro número, y también suma/3:

```
% " Funcion modulo " %
menor(0,X):-nat(X).
menor(s(X),s(Y)):-menor(X,Y).


mod(X, Y, X) :-menor(X, Y).
mod(X, Y, Z) :-suma(X1, Y, X),
               mod(X1, Y, Z).
```

A continuación mostramos el predicado, pre_order/2 que realiza la ordenación de un árbol en formato lista. Como predicado auxiliar para este usamos también el predicado concatenar/3 que concatena dos listas en su tercer argumento, y también segundo/2:

```
% " Funcion preorder " %
pre_order(void,[]).
pre_order(tree(X,Left,Right),Order):-pre_order(Left,OrderLeft),
                                     pre_order(Right,OrderRight),
                                     segundo(X,N),
                                     concatenar([N|OrderLeft],OrderRight,Order).

concatenar([],L,L).
concatenar([X|Xs],Y,[X|Zs]):-concatenar(Xs,Y,Zs).
```

Para seguir mostramos el predicado **sumaLista/2** que dada una lista guarda en su segundo argumento la suma de aquellos elementos de la lista que son pares. Como predicado auxiliar para este usamos también el predicado **compruebamod/2** que dado un número comprueba si tiene módulo 2, y también **segundo/2** que es utilizado en el predicado anterior, que devuelve el segundo elemento del **par(c,n)**:

```
% " Funcion suma lista " %
compruebamod(N,0):- mod(N,s(s(0)),s(0)).
compruebamod(N,N):- mod(N,s(s(0)),0).

segundo(par(C,N),N).

sumaLista([],0).
sumaLista([X|L],S):- sumaLista(L,S1),
                     compruebamod(X,P),
                     suma(S1,P,S).
```

Para terminar con los predicados auxiliares, mostramos el predicado **comparaLongitud/2** que comprueba si la longitud de dos listas es la misma. Como predicado auxiliar para este usamos también el predicado **longitudLista/2** que devuelve la longitud lista, y este a su vez usa también **suma/3**:

```
% " Funcion Compara Longitud Lista " %
comparaLongitud(X,Y):-pre_order(X,LX),
                      pre_order(Y,LY),
                      longitudLista(LX,NX),
                      longitudLista(LY,NY),
                      NX = NY.


longitudLista([],0).
longitudLista([_X|Xs],Y):-longitudLista(Xs,T),
                          suma(s(0),T,Y).
```

A continuación mostramos el código que desarrolla la primera parte del algoritmo, **arbolBalanceadoPar/1**, que a partir de un árbol comprueba si la suma de los pesos pares de su árbol izquierdo es igual a la suma de los pesos pares de su árbol derecho, utilizando como predicado auxiliar **compara/2**, que dadas dos listas, y utilizando a su vez el predicado **sumaLista/2** que nos calcula esa suma de elementos pares de una lista, comprueba si tienen la misma suma pedida:

```
% " Funcion Arbol Balanceado " %
arbolBalanceadoPar(void).
arbolBalanceadoPar(tree(par(C,N),Left,Right)):- pre_order(Left,LL),
                                                pre_order(Right,LR),
                                                compara(LL,LR).


% " Funcion compara dos listas " %
compara(void,L2):-false.
compara(L1,void):-false.
compara(L1,L2):-sumaLista(L1,S1),
                sumaLista(L2,S2),
                S1 = S2.
```

Para finalizar mostramos el código que desarrolla la primera parte del algoritmo, **arbolAmplificado/2**, que a partir de un árbol devuelve como segundo argumento otro árbol idéntico al dado, salvo que en sus nodos hoja se le suma al número que ya tengan en el **par(c,n)** el número del nodo raíz del árbol. Como predicado auxiliar utiliza el predicado **suma/3** y el predicado **comparaAmpliacion/3**, que lo que realiza es la comprobación de que efectivamente en sus nodos hoja se cumpla esta suma.

```
% " Funcion Arbol Amplificado " %
arbolAmplificado(void,void).
arbolAmplificado(tree(par(C,X1),Left1,Right1),tree(par(C,X2),Left2,Right2)):-suma(X1,0,X2),
                                                                              comparaAmpliacion(X1,tree(par(C,X1),Left1,Right1),tree(par(C,X2),Left2,Right2)),
                                                                              comparaLongitud(tree(par(C,X1),Left1,Right1),tree(par(C,X2),Left2,Right2)).


comparaAmpliacion(0,X,X).
comparaAmpliacion(N,tree(par(C,X1),void,void),tree(par(C,X2),void,void)):-suma(X1,N,X2).
comparaAmpliacion(N,tree(par(C,X1),Left1,void),tree(par(C,X2),Left2,void)):-suma(X1,0,X2),comparaAmpliacion(N,Left1,Left2).
comparaAmpliacion(N,tree(par(C,X1),void,Right1),tree(par(C,X2),void,Right2)):-suma(X1,0,X2),comparaAmpliacion(N,Right1,Right2).
comparaAmpliacion(N,tree(par(C,X1),Left1,Right1),tree(par(C,X2),Left2,Right2)):- suma(X1,0,X2),comparaAmpliacion(N,Left1,Left2),
                                                                                 comparaAmpliacion(N,Right1,Right2).
```

- Consultas

Para comenzar realizamos consultas sobre el predicado **arbolBalanceadoPar/1**.

Árbol que cumple que la suma de sus nodos pares del árbol izquierdo es igual a la de sus nodos pares del árbol derecho

```
1 ?- arbolBalanceadoPar(tree(par(c,s(0)),tree(par(e,s(s(0))),void,void),tree(par(s,s(s(0))),void,void))).
true
```

Árbol que no cumple que la suma de sus nodos pares del árbol izquierdo es igual a la de sus nodos pares del árbol derecho

```
1 ?- arbolBalanceadoPar(tree(par(h,s(0)),tree(par(o,s(s(0))),void,void),tree(par(y,s(0)),void,void))).
false.
```

Para finalizar realizamos consultas sobre el predicado **arbolBalanceadoPar/1**.

El predicado es llamado sin el segundo argumento, este lo devuelve el propio predicado

```
1 ?- arbolAmplificado(tree(par(a,s(s(0))),tree(par(m,s(0)),void,void),tree(par(l,s(s(0))),void,void)),AAmp).
AAmp = tree(par(a, s(s(0))), tree(par(m, s(s(s(0)))), void, void), tree(par(l, s(s(s(s(0))))), void, void)) ;
```

El predicado comprueba que es correcto el segundo argumento

```
1 ?- arbolAmplificado(tree(par(n,s(0)),tree(par(o,s(0)),void,void),void),tree(par(n, s(0)), tree(par(o, s(s(0))), void, void), void)).
true ;
```

- Comentarios

En el desarrollo de este ejercicio hemos encontrado diferentes dificultades, entre las que prima el entendimiento del predicado arbolBalanceadoPar/1 y arbolAmplificado/2, de comprensión más sencilla que el predicado del ejercicio 1 cells/2, pero mayor uso y número de funciones auxiliares.