PARTE I (6 puntos): Autómata celular

Un autómata celular esta formado por una cinta infinita (por ambos extremos) que contiene células, las cuales tienen siempre un color dado, y un conjunto de reglas que determinan cómo evoluciona el estado de la cinta: cómo cambian de color sus células. La evolución del autómata consiste en aplicar las reglas a un estado de la cinta para llegar a un nuevo estado, modificando los colores de las células. Por simplicidad, utilizaremos solo el blanco y el negro, representados por o y x, respectivamente.

El nuevo color de una célula depende solo de los colores (originales) de sus dos células vecinas, a izquierda y derecha. Gracias a esto las reglas se pueden definir a partir de solo tres células, tal y como se muestra en la siguiente tabla.

ooo	xoo	oxo	oox	xox	xxo	oxx	xxx
o	x	o	x	x	x	x	o

La tabla representa ocho reglas (en vertical). La primera establece que una célula blanca con ambas vecinas blancas permanece blanca, la segunda que una célula blanca con la vecina izquierda negra y la derecha blanca se vuelve negra, la tercera que una célula de color negro con ambas vecinas de color blanco se vuelve blanca, etc.

La primera regla se conoce como la regla nula y se da por supuesto que pertenece a todo autómata celular. Además, el número de células negras en cualquier estado de una cinta es siempre finito. Gracias a estas dos propiedades, el estado de una cinta se puede representar de forma finita, sin más que mostrar la mínima secuencia de células que empieza y acaba en una célula blanca y contiene a todas las que son negras, como en:

oxxxoooxo donde al aplicar las reglas anteriores se obtiene el nuevo estado: oxxoxxoxoxo

Las (infinitas) células que no se muestran, a izquierda y derecha de la cinta, son todas blancas. Nótese además que (la representación de) el nuevo estado contiene dos células más y que las reglas se han aplicado a todas las células del estado inicial, incluidas las dos blancas de los extremos (y a todas las demás que no aparecen, que eran blancas y lo siguen siendo).

Se debe programar un predicado **cells/2** cuyo segundo argumento sea el estado resultante de aplicar las reglas de un autómata al estado inicial del primer argumento (ambos representados por listas como en [o,x,o,x,o] para el estado oxoxo). Nótese que un autómata no puede evolucionar si su cinta contiene tres células consecutivas a las que no se puede aplicar ninguna regla: en tal caso el predicado debe fallar (de forma finita).

El predicado también debe fallar si no se cumple la especificación dada, es decir, si cualquiera de los estados no comienza o no acaba en una célula blanca o si el estado resultante no contiene dos células más que el estado inicial.

Las reglas de evolución se proporcionan mediante hechos de un predicado regla/4 como en el hecho regla(o,x,o,o) para la tercera regla de la tabla de más arriba. Estos hechos residen en un fichero de nombre 'automaton.pl', por lo que el programa principal no debe contener ninguna definición de regla/4; en su lugar, se debe incluir la línea ":- ensure_loaded(automaton)".

Considérense llamadas a cells/2 en las que se proporciona el primer argumento pero no el segundo y, al revés, llamadas en que se proporciona el segundo argumento pero no el primero. Discútase el comportamiento del programa en cada caso y si es posible mejorarlo.

PARTE II (4 puntos): Codificación Huffman Plog

Uno de los algoritmos más utilizados para compresión de datos es el de Huffman Plog, cuyo objetivo es encontrar un código binario prefijo que codifique un determinado mensaje de texto. Este algoritmo toma como entrada el conjunto de símbolos que aparecen en el mensaje a codificar y la frecuencia de aparición (peso) de cada símbolo en dicho mensaje. El algoritmo consta básicamente de dos etapas: (1) la creación del árbol binario en el que se etiquetan los nodos con los símbolos y los pesos asociados (frecuencia de aparición) y (2) la obtención del código binario prefijo a partir del árbol binario, aplicando una serie de operaciones a los pesos.

Por tanto, se consideran árboles binarios (siguiendo la representación vista en clase) cuyos nodos almacenan una estructura de datos representada por el functor par/2, donde el primer argumento se refiere al símbolo que aparece en el mensaje a codificar y el segundo argumento a su peso (utilizando números naturales en notación de Peano). No se consideran nodos de peso cero.

En la primera etapa del algoritmo, se necesita comprobar si el árbol binario creado está balanceado en las frecuencias pares. Para ello, se pide que el alumno escriba el predicado **arbolBalanceadoPar/1** que se verifique si dado un árbol binario la suma de los pesos pares de su árbol izquierdo es igual a la suma de los pesos pares de su árbol derecho.

Por ejemplo:

```
?- arbolBalanceadoPar(tree(par(h,s(0)),tree(par(o,s(s(0))),void,void),tree(par(y,s(0)),void,void))).
no
?- arbolBalanceadoPar(tree(par(c,s(0)),tree(par(e,s(s(0))),void,void),tree(par(s,s(s(0))),void,void))).
yes
```

En la segunda etapa del algoritmo, se debe realizar una operación de amplificación de los pesos de los nodos hoja. En concreto, dicho aumento se ha de aplicar en cada uno de los nodos hoja en el número de veces que indique la raíz del árbol origen. Para ello, se pide que el alumno escriba el predicado **arbolAmplificado/2** que se verifique si dado un árbol binario origen (primer argumento) el árbol binario amplificado (segundo argumento) es igual que el origen salvo en los nodos hoja en los que el peso se ha incrementado, con respecto al peso original, en el número de veces que indica la raíz del árbol origen.

Por ejemplo:

```
?-
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(0))),void,void),tree(par(l,s(s(s(s(0))))),void,void))
?- arbolAmplificado(tree(par(n,s(0)),tree(par(o,s(0)),void,void),void),AAmp).
AAmp = tree(par(n,s(0)),tree(par(o,s(0)),void,void),void) ?
```