Informe Proyecto de Programación Declarativa.

Annie Hernández Pérez
C−411

Como proyecto de Programación Declarativa se ha implementado un solucionador de nonogramas coloriados en Prolog, tomando como guía la aplicación $CroosMeColor$. El mismo recibe las restricciones de las filas y columnas mediante el predicado *board* y soluciona el nonograma que recibe con *solve*, la solución la deja guardada mediante el predicado *cell* y puede ser apreciado el resultado obtenido mediante el predicado *print_board*.

1. Inicialización

El predicado *board* es el encargado de la inicialización necesaria para el programa. Este recibe las restricciones para las filas y columnas mediante 4 listas de listas, la primera esta constituida por listas que representan las restricciones de colores para cada fila y la segunda con las de tamaño para estas; las otras dos listas son las equivalentes para las columnas. Un ejemplo del uso de este predicado para obtener el nonograma *house* de la aplicación $CrossMeColor$, usando la inicial de cada color en inglés para representarlo, sería:

$$?- board([[b],[b],[b],[b,y,b],[b]],$$
$$[[1],[3],[5],[2,1,2],[5]],$$
$$[[b],[b],[b,y,b],[b],[b]],$$
$$[[3],[4],[3,1,1],[4],[3]])$$

Estas listas son procesadas para obtener la información necesaria, se guarada el predicado *column_restrictions* con las restricciones de cada columna (tiene el número de la columna, su restricciones de colores y tamaños, y el tamaño de la misma) y el predicado *row_restrictions* que tiene estos mismos elementos para las filas, pero además la suma de las restricciones de tamaños (para saber cuantas celdas va a haber ocupadas en esa fila) y la cantidad de restricciones consecutivas que son del mismo color (son la cantidad de celdas en blanco que hay que dejar obligatoriamente). También se almacenan los predicados *block_column_restrictions* y *block_row_restrictions* para cada restricción de columna o de fila respectivamente, vamos a llamar bloque al conjunto de celdas del mismo color que tienen que ir consecutivas (las que nos establecen las restricciones); *block_column_restrictions* va a tener la columna en la que está, su índice en la misma (número de la restricción) y el color y el tamaño de ese bloque; mientras que *block_row_restrictions* va a tener esta misma información para las filas, pero además va a tener todas las columnas posibles donde puede

comenzar ese bloque.

Para saber las posibles columnas donde puede comenzar un bloque se calcula el espacio disponible en su fila, este espacio es el tamaño de la fila menos el que tiene que ser ocupado por los bloques (la suma de todos las restricciones de tamaño) y las celdas que hay que dejar obligatoriamente en blanco (cantidad de bloques del mismo color consecutivos que hay). Entonces cada bloque puede comenzar en la primera columna si es el primero, o en la columna consecuticutiva a donde terminó el bloque anterior, en esta columna mas 1 si son del mismo color, y tiene tantas columnas posibles a partir de esta como la cantidad de espacio disponible que haya en su fila. Estas posibles columnas son filtradas para eliminar algunos casos no válidos, se mira que desde la columna donde se empieza hasta en la que terminaría este bloque todas tengan el color del mismo en su lista de restricciones por colores.

Con toda la información anterior se crea el predicado *block*, sobre el que se va a trabajar, este tiene en que fila y columna se encuentra (inicialmente la columna es la primera de la lista de opciones calculadas), su índice en esta fila, su color, su tamaño y su lista columnas posibles donde puede empezar. Se tiene además el predicado *cell* que tiene su fila, su columna y su color, que va a representar las celdas del tablero e inicialmente van a ser todas blancas.

2. Algoritmo

Para solucionar el nonograma se utiliza el predicado *solve* que va a resolver el nonograma según las restricciones establecidas. Este recorre las filas colocando cada bloque en la primera columna de su lista de opciones y pintando la celdas para que correspondan con la distribución de los bloques. Como estas columnas fueron calculadas de forma tal que siempre fueran válidas para las restriciones por filas solo queda comprobar si satisfacen las resricciones por columnas, de esto se encarga el predicado *check_columns*. Este revisa las celdas comprobando que las restricciones de los bloques por columnas se cumplan y si triunfa termina el programa pero si falla se volverá a hacerse el que llena las filas que hará *backtrack* colocando los bloques en la siguiente columna posible (comprobando siempre que no sobreescriba ningún bloque y que no queden dos bloques del mismo color unidos) y volverán a chequearse las columnas.

3. Ejemplos

Se han puesto varios ejemplos de la aplicación *CroosMeColor* como pruebas del programa, los cuales tiene el mismo nombre que su correspondiente en la aplicación. Los mismos ya están diseñados para que primero agregen las restricciones, luego llamen a *solve* y a *print_board* y por último a un predicado que elimina las cláusulas guardadas para que pueda correrse otro ejemplo.

```
9 ?- flower.
r r   r r
r r r r r
  r y r
r r r r r
r r g r r
    g
    g   g
g   g g g
g g g g
  g g
true .
```

```
8 ?- bee.
    b     b
      b b
g g   b b   g g
g g y y y y g g
  g b b b b g
g g y y y y g g
g   b b b b   g
      y y
true .
```

```
7 ?- apple.
       b b
   r r b r r
 r r r r r r r
 r r r r r r r
 r r r r r r r
   r r r r r
     r r r
true .
```

```
4 ?- tree.
   g g g
 g g g g g
 g g g g g
 g g b g g
     b
true .
```

```
5 ?- house.
     b
   b b b
 b b b b b
 b b y b b
 b b b b b
true .
```

```
1 ?- tutorial4.
   b b b
 b b b b b
 g g g g g
true .
```

```
2 ?- tutorial5.
  b b b
b b b b b
g g g g g
g g   g g
true .
```

```
3 ?- tutorial6.
  g g g
g g g g g
g g g g g
g g b g g
true .
```

```
6 ?- traffic_lights.
b b b b
b r r b
b r r b
b b b b
b y y b
b y y b
b b b b
b g g b
b g g b
b b b b
true .
```