Soluções CuMadness em PROLOG

Autores: Paulo Sérgio Silva Babo(up201404022) e Nuno Fílipe Sousa e Silva(up201404380)

Afiliações: FEUP-PLOG, Turma 3MIEIC02 ,Grupo CuMadness_3

Resumo

Neste trabalho teriamos então que resolver o problema CuMadness utilizando a linguagem PROLOG e as diversas tecnicas de restrições da mesma. Este problema consiste em atribuir cores a uma grelha nas faces de um cubo seguindo uma regra feita para o puzzle (explicada mais abaixo) utilizando as celulas adjacentes a cada celula da grelha. O principal problema foi então descubrir essas celulas adjacentes a cada uma das celulas, num cubo com faces de NxN celulas e a sua colorização seguindo a regra do puzzle. Para resolver estes problemas ultizamos uma serie de formulas matematicas em que dada a celula nos retornava as celulas adjacentes e uma posterior restriçao nas cores dessas mesmas peças , utilizando a biblioteca clpfd do SICStus .

1 Introdução:

O objetivo deste projeto era implementar a resolução de um problema de otimização na linguagem Prolog com restrições.

O problema em questão é a resolução do puzzle cuMadness onde o objetivo é preencher o cubo, de forma a que as peças cumpram as regras do puzzle.

Segue no decorrer deste documento a descrição do puzzle assim como a abordagem usada na resolução e a análise da mesma. A solução implementada permite obter as soluções do puzzle para qualquer cubo, assim como a visualização das mesmas.

2 Descrição do Problema:

O problema principal deste trabalho é então resolver o puzzle CuMadness para cubos de faces com grelhas de NxN. O puzzle consiste em que cada célula da grelha seja pintada de uma das seguintes cores: azul, vermelho, amarelo e verde e que a regra do puzzle seja cumprida. A regra principal é a seguinte cada célula vermelha pode apenas ter uma célula adjacente amarela, cada amarela exactamente uma adjacente verde, cada verde exatamente uma azul, e cada azul exactamente uma vermelha. A cor do resto das adjacentes não importa.

3 Abordagem:

3.1 Variaveis de decisão:

As variáveis de decisão são as células da grelha de cada face do cubo e as suas células adjacentes, dadas por fórmulas matemáticas feitas de forma a que percorrendo uma lista de uma dimensão sejam dadas as adjacentes correspondentes.

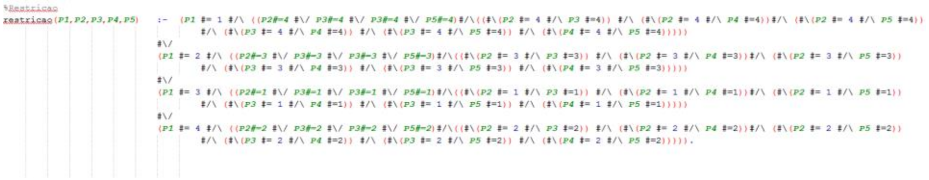
O domínio de cada célula é então um número de 1 a 4 que representam as cores ditas anteriormente, cada número corresponde a uma cor da seguinte maneira:

1 -> Vermelho(R)
2 -> Verde(G)
3 -> Azul(B)
4 -> Amarelo(Y)

3.2 Restrições:

As restrições como explicado no ponto 2 , servem para resolver o puzzle sendo entao que cada célula vermelha pode apenas ter uma célula adjacente amarela, cada amarela exactamente uma adjacente verde, cada verde exatamente uma azul, e cada azul exactamente uma vermelha.

Tendo então isto em mente e utilizando a biblioteca clpfd do SICStus chegamos ás seguintes restrições:



```
%Restricao
restricao(P1,P2,P3,P4,P5)   :-  (P1 #= 1 #/\ ((P2#=4 #\/ P3#=4 #\/ P3#=4 #\/ P5#=4)#/\((#\(P2 #= 4 #/\ P3 #=4)) #/\ (#\(P2 #= 4 #/\ P4 #=4))#/\ (#\(P2 #= 4 #/\ P5 #=4))
        #/\ (#\(P3 #= 4 #/\ P4 #=4)) #/\ (#\(P3 #= 4 #/\ P5 #=4)) #/\ (#\(P4 #= 4 #/\ P5 #=4)))))
#\/
(P1 #= 2 #/\ ((P2#=3 #\/ P3#=3 #\/ P3#=3 #\/ P5#=3)#/\((#\(P2 #= 3 #/\ P3 #=3)) #/\ (#\(P2 #= 3 #/\ P4 #=3))#/\ (#\(P2 #= 3 #/\ P5 #=3))
        #/\ (#\(P3 #= 3 #/\ P4 #=3)) #/\ (#\(P3 #= 3 #/\ P5 #=3)) #/\ (#\(P4 #= 3 #/\ P5 #=3)))))
#\/
(P1 #= 3 #/\ ((P2#=1 #\/ P3#=1 #\/ P3#=1 #\/ P5#=1)#/\((#\(P2 #= 1 #/\ P3 #=1)) #/\ (#\(P2 #= 1 #/\ P4 #=1))#/\ (#\(P2 #= 1 #/\ P5 #=1))
        #/\ (#\(P3 #= 1 #/\ P4 #=1)) #/\ (#\(P3 #= 1 #/\ P5 #=1)) #/\ (#\(P4 #= 1 #/\ P5 #=1)))))
#\/
(P1 #= 4 #/\ ((P2#=2 #\/ P3#=2 #\/ P3#=2 #\/ P5#=2)#/\((#\(P2 #= 2 #/\ P3 #=2)) #/\ (#\(P2 #= 2 #/\ P4 #=2))#/\ (#\(P2 #= 2 #/\ P5 #=2))
        #/\ (#\(P3 #= 2 #/\ P4 #=2)) #/\ (#\(P3 #= 2 #/\ P5 #=2)) #/\ (#\(P4 #= 2 #/\ P5 #=2))))).
```

Fig. 1. Restrições.

Utilizando o predicado restricao(P1,P2,P3,P4,P5) e as seguintes condições garantimos que, por exemplo, cada peça vermelha (1) tem apenas uma adjacente amarela (4), a condição faz então com que pelo menos uma das peças seja uma amarela, e depois restringe a apenas uma utilizando combinações em que essas duas peças não podem ser as duas 4.

O programa garante então assim desta forma as condições para o puzzle, fazendo isto para todas as células do cubo e atribuindo posteriormente os valores a essas células.O programa garante então assim desta forma as condições para o puzzle, fazendo isto para todas as células do cubo e atribuindo posteriormente os valores a essas células.

3.3 Função de Avaliação:

Não achamos necessária a implementação de uma função de avaliação visto que as restrições , não deixam com que a condição de vitoria do puzzle falhe, se existente.

3.4 Estratégia de Pesquisa:

A estratégia de pesquisa utilizada é percorrer uma lista representativa de um cubo de tamanho 6xNxN e para cada posição da lista ver as adjacentes e aplicar as restrições, só após aplicar o labeling de forma a que as restrições sejam aplicadas .

4 Visualização da Solução:

Para a visualização da solução e do cubo com cada célula preenchida é utilizado o método display_cube(X,N), que dá display á lista X (cubo) de tamanho 6*N*N e escreve-o no ecrã em modo de texto na forma de representação de um cubo em 2D, como se pode ver na seguinte figura, sendo deste modo possível ter uma melhor visualização das células adjacentes e do valor que cada uma destas toma.

```
             #########
            | B  Y  G |

            | G  R  B |

            | Y  B  B |

 ########################################
| R  R  R | R  G  R | G  Y  R | R  B  Y |

| G  Y  R | R  Y  Y | G  B  R | Y  G  G |

| B  Y  G | B  B  B | Y  B  R | Y  G  B |

 ########################################
            | G  R  R |

            | Y  Y  G |

            | R  R  R |

             #########
```

Fig. 2. Representação do cubo 3x3 em modo de texto do SICStus

5 Resultados:

Reproduzimos o problema para cubos de 2x2, 3x3, 4x4, 5x5, em que nos 3 primeiros obtemos resultados e uma grande quantidade de soluções diferentes, porém no 5x5 provavelmente devido á enorme quantidade de soluções o programa correu durante 1 hora e 30 minutos tendo sendo posteriormente desligado então não termos conseguido apurar o tempo exato.

Table 1. de tempos para obtenção da solução

Tamanho	Tempo
2x2	0.28s
3x3	0.56s
4x4	490s
5x5	-

6 Conclusões:

Para concluir este projeto, podemos afirmar que a utilização de Prolog para certos problemas complexos, facilita o trabalho do programador, apesar de em outras tarefas mais simples, como mostrar a representação do cubo, ser desnecessariamente mais complicado, comparado com outras linguagens

7 Bibliografia:

http://www.jaapsch.net/puzzles/culica.htm

http://www.swi-prolog.org/

http://stackoverflow.com/questions/28457984/how-do-you-change-write-options-in-prolog-to-print-a-long-list

http://stackoverflow.com/questions/14888174/how-do-i-determine-if-exactly-one-boolean-is-true-without-type-conversion