Matheus Henrique Schaly – 18200436

08-05-2020

Straights – Prolog

1. Programação de restrições

As duas principais diferenças entre Prolog puro e constraint logic programming (CLP) é:

1) Restrições podem ser adiadas até que suas verdades sejam decididas com segurança. Consequentemente, aumentando a generalidade do programa.

2) Restrições podem levar em consideração independentemente da ordem em que são declaradas. Portanto, melhorando significativamente o desempenho do programa.

A programação de restrições foi utilizada em diversas partes do código, como pode ser visto na imagem abaixo.

1) Todos os valores devem estar entre o menor valor da matriz e o maior valor da matriz.
2) Todos os valores das linhas devem ser distintos.
3) Todos os valores das colunas devem ser distintos.
4) Todos os valores da matriz devem ser positivos.
5) O valor máximo menos o valor mínimo de uma tira de valores em branco deve ser igual ao tamanho da tira menos 1.
6) Todos os valores de uma tira de valores em branco devem estar entre (ou serem iguais) o valor máximo e o valor mínimo da tira.

```
append(Filled_Rows, Vs),                    % Appends every list in Filled_Rows into a single list
Vs ins Lower_N..Upper_N,1                   % Constraints main matrix to range from lowest matrix value to its lenght
maplist(all_distinct, Filled_Rows),2        % Constraints each row to contain only distinct values
transpose(Filled_Rows, Filled_Columns),     % Transposes the main matrix
maplist(all_distinct, Filled_Columns),3     % Constraints each column to contain only distinct values
transpose(Colors, ColorsT),                 % Transposes the color matrix
testWhites(Filled_Rows, Colors),            % Checks, in each row, if every sequence of white cells is valid
testWhites(Filled_Columns, ColorsT),        % Checks, in each column, if every sequence of white cells is valid
```

```
% Recursive case, main row has a value and color matrix has a white cell
% Applies a constraint which states that each value has to be greater than 0
testSubWhites([H1|T1], [0|T2], Acc) :-
    H1 #> 0,4
    testSubWhites(T1, T2, [H1|Acc]).

% Checks if a sequence of white cells is valid
% Applies a contraint which states that the maximum and minimum value of the
% row has to be equal its length minus 1
checkSequence(Acc) :-
    constraint_list_min(Acc, Min),
    constraint_list_max(Acc, Max),
    length(Acc, N1),
    N2 = N1 - 1,
    Max - Min #= N2,5
    constraint_min_max(Acc, Max, Min).
```

```
% Recursive case, checks if each value is between (or equal) Max and Min value
% Applies two contraints which state that each element in a white strip has to be
% between (or equal) the maximum and minimum values of the string
constraint_min_max([H|T], Max, Min) :-
    H #=< Max, 6
    H #>= Min, 6
    constraint_min_max(T, Max, Min).

% Applies contraint_min to each element contained in the white strip
constraint_list_min([L|Ls], Min) :-
    foldl(constraint_min, Ls, L, Min).

% Applies a constraint and finds the minumum value between two values.
constraint_min(X, Y, Min) :-
    Min #= min(X, Y). 6

% Applies contraint_min to each element contained in the white strip
constraint_list_max([L|Ls], Max) :-
    foldl(constraint_max, Ls, L, Max).

% Applies a constraint and finds the maximum value between two values.
constraint_max(X, Y, Max) :-
    Max #= max(X, Y). 6
```

2. Vantagens e desvantagens da programação de restrições

As vantagens são a generalidade do programa e o ganho significativo do desempenho.

Além disso, acabei realizando a implementação do puzzle primeiramente não usando programação de restrições. Com isso notei que, ao utilizar programação de restrições:

Vantagens:

1) O código é menor.
2) A velocidade do programa é significativamente maior. Uma matriz 9x9 roda instantaneamente. Por outro lado, sem utilizar programação por restrições, o programa começava a demorar já alguns poucos segundos com uma matriz 6x6.

Desvantagem:

1) Mais tempo foi necessário para programar utilizando programação de restrições.

3. Como usar o programa

O tabuleiro utiliza duas matrizes. A primeira chama-se Rows e contém os valores do tabuleiro, onde "_" significa ausência de valor. Tal matriz é dada por uma lista de listas onde cada lista representa uma linha. A segunda matriz chama-se Colors e contém as cores da matriz, onde 1 simboliza um campo preto e 0 um campo branco.

Há seis exemplos de tabuleiros já criados no código fonte. Para fornecer um novo input ao programa o usuário precisa criar ou alterar um predicado diretamente no código fonte.

Para executar, por exemplo, a matriz 9x9 fornecida, execute o seguinte comando: ?-get_9x9_matrices_1(Rows, Colors), straights(Rows, Colors).

O output é fornecido por um *write* no código. Sendo assim, o output pode ser visualizado a partir do software que chamou o programa após a string "Possible *solution:*" como pode ser visto abaixo. Observe que os números 0 simbolizam ausência de valor pois o campo é de cor preta e não possuía valor prévio.

```
?- get_9x9_matrices_1(Rows, Colors), straights(Rows, Colors).

Main matrix input:
[0,0,_63706,5,0,1,_63730,3,0]
[_63754,9,_63766,3,_63778,2,_63790,_63796,_63802]
[7,_63820,1,_63832,_63838,_63844,0,_63856,5]
[5,0,0,4,_63898,0,_63910,8,6]
[0,_63940,_63946,_63952,_63958,_63964,_63970,_63976,0]
[8,_64000,4,0,_64018,_64024,0,0,_64042]
[_64054,1,0,_64072,_64078,_64084,5,4,_64102]
[_64114,_64120,_64126,_64132,_64138,_64144,_64150,_64156,_64162]
[0,7,_64186,9,0,5,6,0,0]


Color matrix input:
[1,1,0,0,1,0,0,1,1]
[0,0,0,0,0,0,0,0,0]
[0,0,1,0,0,0,1,0,0]
[0,1,1,0,0,1,0,0,1]
[1,0,0,0,0,0,0,0,1]
[1,0,0,1,0,0,1,1,0]
[0,0,1,0,0,0,1,0,0]
[0,0,0,0,0,0,0,0,0]
[1,1,0,0,1,0,0,1,1]


Possible solution:
[0,0,6,5,0,1,2,3,0]
[6,9,5,3,8,2,1,7,4]
[7,8,1,2,4,3,0,6,5]
[5,0,0,4,3,0,9,8,6]
[0,4,3,6,5,7,8,9,0]
[8,3,4,0,7,6,0,0,2]
[2,1,0,7,9,8,5,4,3]
[3,2,9,8,6,4,7,5,1]
[0,7,8,9,0,5,6,0,0]

Rows = [[0, 0, 6, 5, 0, 1, 2, 3|...], [6, 9, 5, 3, 8, 2, 1|...],
Colors = [[1, 1, 0, 0, 1, 0, 0, 1|...], [0, 0, 0, 0, 0, 0, 0|...

?-
```

4. Dificuldades

As dificuldades e soluções foram:

1. Como representar o tabuleiro: A solução adotada foi separar o tabuleiro em duas matrizes, uma contendo os valores e a outra contendo as cores.
2. Como lidar com os campos pretos: A solução adotada foi atribuir valores negativos decrescentes a tais campos. Negativos pois eles precisam ser diferenciados dos valores pré-existentes da matriz, e decrescentes para eles não gerarem repetição.
3. Como lidar com as tiras de campos brancos: A solução adotada foi criar o algoritmo que identifica as tiras de campos brancos, atribui uma restrição aos valores mínimos e máximos e atribui restrições aos valores intermediários.

5. Comparação entre o paradigma lógico e o paradigma funcional

Uma diferença fundamental é que Prolog utiliza predicados enquanto Haskell e Scheme utilizam primariamente funções. Já em relação à resolução do puzzle, há uma grande diferença ao programar utilizando recursividade e utilizando programação de restrições. Recursividade é normalmente utilizada até em linguagens como Python, Java ou C++. Entretanto, recordo-me de ver programação de restrições apenas em Prolog. Achei bem interessante como Prolog

consegue resolver o puzzle ao utilizar programação de restrições. Apesar de recursividade ser muito mais natural para mim, achei mais fácil programar utilizando programação de restrições.