Yin Yang – Resolução do tabuleiro usando Programação em Lógica com Restrições

José Miguel Costa e Luís Miguel Gonçalves

Faculdade de Engenharia da Universidade do Porto
Rua Roberto Frias, sn, 4200-465 Porto, Portugal

Resumo: O artigo presente serve como complemento ao segundo projeto da Unidade Curricular de Programação em Lógica do Mestrado Integrado em Engenharia Informática e Computação. O projeto consiste no desenvolvimento de um programa em ProLog utilizando Programação em Lógica com Restrições. O problema escolhido foi o jogo de tabuleiro Yin Yang. Ao longo do artigo irá ser demonstrado como, través da utilização dos predicados disponibilizados pelo SICstus ProLog, foi possível resolver o problema em específico e de forma eficiente.

Introdução

O objetivo deste projeto era implementar a resolução de um problema de otimização em Prolog com Restrições, e a partir dos temas disponíveis, tanto puzzles como problemas menos concretos, decidimos implementar o puzzle Yin Yang.

Este puzzle consiste em colocar peças num tabuleiro dividido em duas regiões, preta e branca.

Este artigo descreve detalhadamente o puzzle Yin Yang, bem como a abordagem do grupo para a implementação de uma solução capaz de resolver o puzzle com diferentes tamanhos, a explicação da forma usada para visualizar o tabuleiro resolvido em modo de texto, diferentes estatísticas da resolução do puzzle e a conclusão do projeto.

Descrição do Problema

O puzzle Yin Yang consiste num tabuleiro quadrado onde estão já algumas peças pretas e brancas posicionadas e, tal como no sudoku, o objetivo do jogador é preencher totalmente o tabuleiro colocando mais peças, neste caso, pretas e brancas, obedecendo às seguintes restrições:

- Todas as peças da mesma cor têm de estar ligadas umas às outras, verticalmente ou horizontalmente,
- Não pode haver nenhum grupo de 2x2 em que todas as peças sejam da mesma cor.

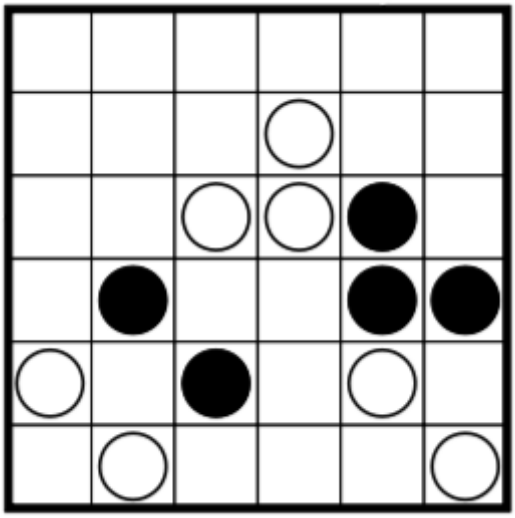

Figura 1 - Exemplo do tabuleiro inicial

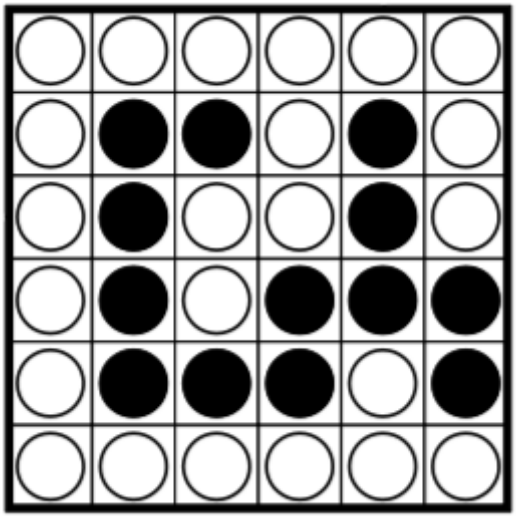

Figura 2 - Exemplo da resolução do tabuleiro da figura 1

Abordagem

Variáveis de Decisão

As variáveis de decisão consistem no tabuleiro em si e pelas diferentes listas que é composto (linhas do tabuleiro), sendo que cada lista é instanciada individualmente.

O domínio do tabuleiro, atribuído individualmente a cada lista dentro dele mesmo e consequentemente a cada célula contida em cada lista, é entre 1 e 2 inclusive. Sendo que o valor 1 representa peças da cor preta e o valor 2 peças de cor branca, não existindo outro tipo de peças. Por fim, a cada linha é atribuída um comprimento (*length*) de acordo com o tamanho do tabuleiro pretendido.

Restrições

Antes de restringir a solução final consoante as regras do jogo, é preciso definir que o tabuleiro final deverá ser completado a partir de um já existente. Assim sendo, inicialmente, e consoante o input do utilizador ao chamar a nossa função de resolução do puzzle, são atribuídos valores às células indicadas.

Começando pela segunda regra referida na descrição do problema, não podemos permitir blocos 2x2 células com peças da mesma cor, por forma a não deixar que tal aconteça, o tabuleiro é percorrido (usando a função *twoByTwo*) e restringindo a que o somatório dos elementos de cada bloco esteja contido entre 4 e 8 (não inclusive).

```
63  twoByTwo([_],[_]).
64  twoByTwo([F1,S1|T1],[F2,S2|T2]):-
65      F1 + S1 + F2 + S2 #> 4,
66      F1 + S1 + F2 + S2 #< 8,
67      twoByTwo([S1|T1],[S2|T2]).
```

Figura 3 - Função que assegura a restrição de não existirem blocos 2x2 da mesma cor.

Por fim, e por forma a respeitar a primeira regra referida acima, foram definidas 3 funções (sendo estas *restrictPath*, *restrictLastCol* e *restrictLastLine*). Basicamente o que cada uma destas faz é ter a certeza que para cada peça no tabuleiro existe pelo menos uma peça da mesma cor adjacente. Exemplo de como assegurámos esta restrição:

```
15  restrictLastCol(FirstLine, SecondLine):-
16      element(6, SecondLine, E1),
17      element(5, SecondLine, E2),
18      element(6, FirstLine, E3),
19      E1 #= E2 #\/ E1 #= E3.
```

Figura 4 – Função que força ligação entre peças (para a ultima coluna de cada linha)

Por fim, é relevante referir que outra restrição foi criada, mas não está a ser utilizada pelo grupo pois não consta nas regras do jogo. Esta sendo que tem de existir o mesmo número de peças pretas e brancas na solução do tabuleiro, que inicialmente o grupo achava que o jogo era completado colocando uma peça de cada cor alternadamente até completo o tabuleiro, mas mais tarde foi observado que não existe nenhuma referencia a esta regra. Assim sendo, esta encontra-se comentada no código enviado.

Função de Avaliação

A solução obtida é avaliada pelos próprios elementos do grupo ao visualizar o resultado final do tabuleiro e verificando que a solução respeita as regras. Além disto, não achámos necessária a implementação de uma função de avaliação visto que as restrições permitem que a condição de completação do puzzle falhe, caso esta exista.

Estratégia de Pesquisa

A estratégia de pesquisa consiste nas restrições serem aplicadas da seguinte forma: primeiro não permitindo que existam espaços 2x2 com peças da mesma cor e depois verificando se todas as peças estão ligadas a pelo menos outra da sua cor. O *labeling* foi deixado vazio, sendo que foi, de todas as experiencias, a forma como obteve melhores resultados em termos de tempo.

Visualização da Solução

Tendo em conta que, tal como no primeiro projeto, o resultado a obter é um tabuleiro, em que números representam tipos de peças diferentes, utilizamos os predicados feitos nesse mesmo projeto, mas com pequenas alterações.

Todos os predicados para este efeito encontram-se nu ficheiro separado (board.pl). No predicado printBoard([H|T]) a lista é passada como argumento e a função printSpaces é responsável por imprimir as linhas verticais entre as casas e as peças no seu interior.

Por cada elemento da lista recebida por printSpaces é chamada a função translatePrint que imprime o valor da peça (0 se for uma casa vazia, 1 se for uma peça branca e 2 se for uma peça preta). Para além do tabuleiro em si são também imprimidas estatísticas relativas à resolução, tópico que será mais aprofundado no próximo ponto.

```
| 1 | 1 | 1 | 1 | 1 | 1 |
| 1 | 2 | 1 | 2 | 2 | 2 |
| 1 | 2 | 1 | 1 | 1 | 1 |
| 1 | 2 | 1 | 2 | 2 | 1 |
| 1 | 2 | 1 | 1 | 2 | 1 |
| 1 | 2 | 2 | 1 | 2 | 1 |


Execution time: 312 milliseconds.
Local Stack: 496 bytes.
Trail Stack: 15700 bytes.
Bactrack Stack: 888 bytes.
Resumptions: 843
Entailments: 351
Prunings: 492
Backtracks: 3
Constraints created: 260
Input = [],
Output = [[1,1,1,1,1,1],[1,2,1,2,2,2],[1,2,1,1,1,1],[1,2,1,2,2,1],[1,2,1,1,2,1]
,[1,2,2,1,2|...]] ? yes
```

Figura 5 - Exemplo do output do programa

```
13  printBoard([]):-
14    nl, nl.
15  printBoard([H|T]):-
16    printSpaces(H),
17    printBoard(T).
18  printSpaces([]):-
19    write('|'),
20    nl.
21  printSpaces([H|T]):-
22    write('|'),
23    translatePrint(H),
24    printSpaces(T).
25
26  translatePrint(0):-
27    write('   ').
28  translatePrint(X):-
29    write(' '),
30    write(X),
31    write(' ').
```

Figura 6 - Código referente à impressão do tabuleiro

Resultados

Foram efetuados testes com tabuleiros de 3 tamanhos diferentes: 4x4, 5x5 e 6x6, e anotadas diferentes estatísticas sobre a execução. Sendo que as diferenças dos tempos de execução variaram pouco entre cada experiencia (na ordem das centésimas de segundo), mas, tal como esperado, aumenta à medida que o tamanho do tabuleiro é maior.

	Execution Time	Local Stack	Trail Stack	Backtrack Stack	Resumptions	Entailments	Prunings	Backtracks	Constraints Created
4x4	282 ms	800 bytes	33944 bytes	1448 bytes	559	189	371	4	262
5x5	297 ms	896 bytes	18280 bytes	1640 bytes	624	173	387	5	312
6x6	313 ms	992 bytes	27848 bytes	1832 bytes	827	350	489	3	262

Tabela 1 - Tabela de estatísticas para diferentes tabuleiros

Todas as estatísticas retiradas das diferentes experiencias foram obtidas o predicado fd_statistics, que é um predicado da biblioteca clpfd, bem como o predicado statistics/2 que nos permitiu obter o tempo de execução, *local stack*, *trail stack* e *backtrack stack*.

Foram ainda realizadas experiencias com diferentes variações da função de *labeling*, no entanto, em todos os casos para todas as experiencias foi observado que o *labeling* quando deixado vazio. É de salientar que não existe qualquer tipo de valor a minimizar ou maximizar para a resolução deste puzzle sendo que qualquer solução é tão válida como qualquer outra. Uma das experiencias realizadas foram com um tabuleiro inicial mais complexo (Figura 1), por forma a comparar o tempo de execução do programa. O *labeling* usado foi [ff,bisect], que acreditávamos ser apropriado, usando o principio de *first-fail* e dividindo o domínio em duas partes optando apenas por uma, pois diminuiria o tempo total de execução. No entanto, quando comparado com o *labeling* vazio, o tempo de execução foi (em média) maior, obtendo ambos, exatamente a mesma solução e constantes valores em *stack*.

Labeling	*Média tempo de execução (ms)*
[]	219
[ff,bisect]	265

Conclusões e Trabalho Futuro

Este projeto serviu para melhor compreendermos a utilidades da programação em lógica com restrições na resolução de problemas deste género.

Relativamente ao uso de Prolog com restrições concluímos que, com o seu uso, o desenvolvimento de programas complexos é facilitado, apesar de em certas tarefas, imprimir o tabuleiro por exemplo, não ser tão intuitivo e eficaz como noutras linguagens. Uma das grandes dificuldades do grupo na realização deste projeto foi a regra que todas as peças da mesma cor têm de estar ligadas umas às outras, sendo que noutra linguagem, seria mais fácil percorrer o tabuleiro usando pesquisa em largura e verificar que tal acontecia, sendo que foi um desafio implementar esta regra, não estando totalmente correta.

Quanto à nossa implementação em concreto, existem aspetos que poderiam ser melhorados, nomeadamente a forma como o executamos em tabuleiros de diferentes tamanhos visto ser uma alteração do próprio programa replicado.

Bibliografia

- https://sicstus.sics.se/sicstus/docs/latest4/html/sicstus.html. SICS Swedish ICT AB, 2001.
- https://moodle.up.pt/pluginfile.php/55023/mod_resource/content/5/PLR_SICStus.pdf. Henrique Lopes Cardoso, Novembro, 2015.

Anexos

1. Ficheiro com código fonte responsável pela resolução do puzzle Yin Yang: yin-yang.pl

```
:- use_module(library(clpfd)).
:- use_module(library(lists)).
:- use_module(library(system)).
:- [board].

%Make sure every piece is connected to another piece
restrictPath(_,_,6).
restrictPath(FirstLine, SecondLine, Inc):-
        Inc2 is Inc + 1,
        element(Inc, FirstLine, E1),
        element(Inc2, FirstLine, E2),
        element(Inc, SecondLine, E3),
        E1 #= E2 #\/ E1 #= E3,
```

```
        restrictPath(FirstLine, SecondLine, Inc2).
restrictLastCol(FirstLine, SecondLine):-
        element(6, SecondLine, E1),
        element(5, SecondLine, E2),
        element(6, FirstLine, E3),
        E1 #= E2 #\/ E1 #= E3.
restrictLastLine(_,_,6).
restrictLastLine(SecondLine, FirstLine, Inc):-
        Inc2 is Inc + 1,
        element(Inc, SecondLine, E1),
        element(Inc, FirstLine, E2),
        element(Inc2, SecondLine, E3),
        E1 #= E2 #\/ E1 #= E3,
        restrictPath(FirstLine, SecondLine, Inc2).

%Gets input from user and adds pieces to final board
restrictFromInput([],_,_,_,_,_,_).
restrictFromInput([[Colour,X,0]|T],L1,L2,L3,L4,L5,L6):-
        X2 is X + 1,
        element(X2, L1, Colour), write('Input at line 0'), nl,
        restrictFromInput(T,L1,L2,L3,L4,L5,L6).
restrictFromInput([[Colour,X,1]|T],L1,L2,L3,L4,L5,L6):-
        X2 is X + 1,
        element(X2, L2, Colour), write('Input at line 1'), nl,
        restrictFromInput(T,L1,L2,L3,L4,L5,L6).
restrictFromInput([[Colour,X,2]|T],L1,L2,L3,L4,L5,L6):-
        X2 is X + 1,
        element(X2, L3, Colour), write('Input at line 2'), nl,
        restrictFromInput(T,L1,L2,L3,L4,L5,L6).
restrictFromInput([[Colour,X,3]|T],L1,L2,L3,L4,L5,L6):-
        X2 is X + 1,
        element(X2, L4, Colour), write('Input at line 3'), nl,
        restrictFromInput(T,L1,L2,L3,L4,L5,L6).
restrictFromInput([[Colour,X,4]|T],L1,L2,L3,L4,L5,L6):-
        X2 is X + 1,
        element(X2, L5, Colour), write('Input at line 4'), nl,
        restrictFromInput(T,L1,L2,L3,L4,L5,L6).
restrictFromInput([[Colour,X,5]|T],L1,L2,L3,L4,L5,L6):-
        X2 is X + 1,
        element(X2, L6, Colour), write('Input at line 5'), nl,
        restrictFromInput(T,L1,L2,L3,L4,L5,L6).

%Not allow 2x2of the same colour
twoByTwo([_],[_]).
twoByTwo([F1,S1|T1],[F2,S2|T2]):-
        F1 + S1 + F2 + S2 #> 4,
        F1 + S1 + F2 + S2 #< 8,
```

```
        twoByTwo([S1|T1],[S2|T2]).

    %Set length and domain for each line
    defineLines([]).
    defineLines([L|T]):-
        length(L,6),
        domain(L,1,2),
        defineLines(T).

    %Input são coordenadas das peças ja na board do tipo [1,0,4] (peça de cor preta(1) de coordenadas (0,4), sendo cor branca = 2)
        %Example 1: solveGame([[1,0,4],[2,1,1]], Board). Example 2: solveGame([[2,0,4],[2,1,5],[2,5,5],[2,4,4],[2,2,2],[2,3,1],[1,2,4],[1,1,3],[1,4,3],[1,4,2],[1,5,3]], Board).
    solveGame(Input, Output):-
        nl,
        Output = [L1,L2,L3,L4,L5,L6],
        defineLines(Output),
        restrictFromInput(Input,L1,L2,L3,L4,L5,L6), !,
        twoByTwo(L1,L2),
        twoByTwo(L2,L3),
        twoByTwo(L3,L4),
        twoByTwo(L4,L5),
        twoByTwo(L5,L6),
        restrictPath(L1,L2, 1),
        restrictPath(L2,L3, 1),
        restrictPath(L3,L4, 1),
        restrictPath(L4,L5, 1),
        restrictPath(L5,L6, 1),
        restrictLastCol(L1, L2),
        restrictLastCol(L2, L3),
        restrictLastCol(L3, L4),
        restrictLastCol(L4, L5),
        restrictLastCol(L5, L6),
        restrictLastLine(L6, L5, 1),
        append(Output, NewOutput),
        %sum(NewOutput, #=,54), %Restriction not used
        %Statisctics calculators
        statistics(runtime, [TimeOfExec,_]),
        statistics(local_stack, [LocalStack,_]),
        statistics(trail, [TrailStack,_]),
        statistics(choice, [BackTrackStack,_]),
        %Labeling
        labeling([], NewOutput), nl, printBoard(Output),
        %Statistics
        nl, write('Execution time: '), write(TimeOfExec), write(' milliseconds.'),
        nl, write('Local Stack: '), write(LocalStack), write(' bytes.'),
```

```
        nl, write('Trail Stack: '), write(TrailStack), write(' bytes.'),
        nl, write('Bactrack Stack: '), write(BackTrackStack), write(' bytes.'), nl,
        fd_statistics.
```

2. Ficheiro com código fonte para desenhar o tabuleiro do puzzle Yin Yang: board.pl.

```
printBoard([]):-
        nl, nl.
printBoard([H|T]):-
        printSpaces(H),
        printBoard(T).

printSpaces([]):-
        write('|'),
        nl.
printSpaces([H|T]):-
        write('|'),
        translatePrint(H),
        printSpaces(T).

translatePrint(0):-
        write('   ').
translatePrint(X):-
        write(' '),
        write(X),
        write(' ').
```