Resolução do puzzle Tents usando programação em lógica com restrições

Mestrado Integrado em Engenharia Informática e Computação

Programação em Lógica

Turma: 3MIEIC02, Grupo: Tents5
Hélder Antunes - up201406163
Inês Proença - 201404228

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

19 de Dezembro de 2016

Resumo

Este trabalho teve como objetivo construir um programa que implementa um solver do puzzle Tents. Para isso, utilizou-se conceitos de programação em lógica com restrições.

1 Introdução

O objetivo principal deste trabalho foi implementar, em liguagem Prolog, um solver do puzzle Tents.

Outro objetivo importante, que serviu como motivação, foi desenvolver aptidões fundamentais no âmbito da programação em lógica com restrições.

O resto do relatório servirá para descrever detalhadamente o problema e sua forma de resolução.

2 Descrição do Problema

É dado um tabuleiro quadrangular com algumas árvores colocadas. O objetivo do puzzle é colocar tendas, de forma a que se cumprem os seguintes objetivos:

- Colocar uma tenda horizontalmente ou verticalmente a cada uma das árvores.
- As tendas não se tocam entre si, nem mesmo diagonalmente.
- Os números fora do tabuleiro indicam o número de tendas por linha e coluna

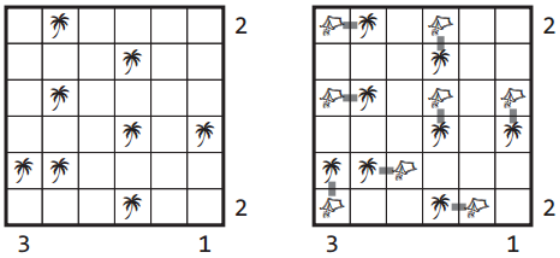

Figura 1: Exemplo de um puzzle

3 Abordagem

3.1 Variáveis de Decisão

As variáveis de entrada são: tamanho do tabuleiro (número de quadrículas na horizontal ou vertical), uma lista com as posições das árvores, duas listas com o número de tents nas linhas e nas colunas.

A variável de decisão é uma lista com as posições dos tents. O domínio de cada posição vai de 0 a tamanhoTabuleiro x tamanhoTabuleiro - 1, inclusive.

Pode-se saber a linha de um tent fazendo posiçãoTent / tamanhoTabuleiro + 1. Para a coluna faz-se posição posiçãoTent mod tamanhoTabuleiro +1.

3.2 Restrições

Restrição "Colocar uma tenda horizontalmente ou verticalmente a cada uma das árvores":

```
tentIsAroundHisTree_restriction([], [], _).
tentIsAroundHisTree_restriction([Tree|Trees], [Tent|Tents], SizeBoard):-
        %Toca horizontalmente
((Tree #= Tent - 1 #\/ Tree #= Tent + 1) #/\ Tree / SizeBoard #= Tent / SizeBoard )
        %Toca Verticalmente
    #\/ (Tree #= Tent - SizeBoard) #\/ (Tree #= Tent + SizeBoard),
    tentIsAroundHisTree_restriction(Trees, Tents, SizeBoard).
```

Restrição "As tendas não se tocam entre si, nem mesmo diagonalmente":

```
% Colaca a restricao: tents nao sao vizinhos entre si.
tentsDontTouchEachOther_restriction([], _, _).
tentsDontTouchEachOther_restriction([T|Ts], Tents, SizeBoard) :-
    tentIsIsolated(T, Tents, SizeBoard),
    tentsDontTouchEachOther_restriction(Ts, Tents, SizeBoard).

% Coloca a restricao: um tent nao e vizinho de nenhum outro tent.
tentIsIsolated(_, [], _).
tentIsIsolated(Tent, [T|Ts], SizeBoard) :-
        % toques horizontais
    (((Tent #= T + 1 #\/ Tent #= T - 1) #/\ Tent / SizeBoard #= T / SizeBoard)
        % toques verticais
     #\/ Tent #= T + SizeBoard #\/ Tent #= T - SizeBoard #\/
        % toques diagonais
    (Tent #= T + SizeBoard + 1 #/\ T / SizeBoard #= Tent / SizeBoard - 1) #\/
    (Tent #= T + SizeBoard - 1 #/\ T / SizeBoard #= Tent / SizeBoard - 1) #\/
    (T #= Tent + SizeBoard + 1 #/\ Tent / SizeBoard #= T / SizeBoard - 1) #\/
    (T #= Tent + SizeBoard - 1 #/\ Tent / SizeBoard #= T / SizeBoard - 1)) #<=> B,
        B #= 0,
    tentIsIsolated(Tent, Ts, SizeBoard).
```

Restrição "Os números fora do tabuleiro indicam o número de tendas por linha e coluna"(é mostrado apenas a restrição nas linhas, a restrição nas colunas é idêntica):

```
% Coloca a restricao: Na linha x existem y tents.
numTentsInRows_restriction(_, _, Row, SizeBoard) :- Row > SizeBoard.
numTentsInRows_restriction(RowTents, Tents, Row, SizeBoard) :-
        Row =< SizeBoard,
        nth1(Row, RowTents, -1), !, % numero de tents nao determinado
        NextRow is Row + 1,
        numTentsInRows_restriction(RowTents, Tents, NextRow, SizeBoard).
numTentsInRows_restriction(RowTents, Tents, Row, SizeBoard) :-
        Row =< SizeBoard,
        countTentsInRow(Tents, Row, SizeBoard, Total),
        element(Row, RowTents, Total),
        NextRow is Row + 1,
        numTentsInRows_restriction(RowTents, Tents, NextRow, SizeBoard).

% Conta o numero de tents na linha Row.
countTentsInRow([], _, _, 0).
countTentsInRow([Tent|Tents], Row, SizeBoard, Total) :-
        Tent / SizeBoard + 1 #= Row #<=> B,
```

```
        Total #= B + TotalAux,
        countTentsInRow(Tents, Row, SizeBoard, TotalAux).
```

3.3 Estratégia de Pesquisa

Na ordenação de variáveis utilizou-se a etiqueta ffc, ou seja, são escolhidas as variáveis de menor domínio, e com mais restrições.

Na seleção de valores de uma variável utilizou-se a etiqueta bisect, evitando, dessa forma, valores outliers.

Na ordenação de valores utilizou-se a opção por defeito (up), em que o domínio é explorado por ordem ascendente.

Para a geração de puzzles aleatórios, utilizou-se uma seleção costumizada de valores em que se escolhe um valor aleatório de entre os valores possíveis do domínio.

```
mySelValores(Var, _Rest, BB, BB1) :-
        fd_set(Var, Set),
        select_best_value(Set, Value),
        (
            first_bound(BB, BB1), Var #= Value
        ;
            later_bound(BB, BB1), Var #\= Value
    ).

select_best_value(Set, BestValue):-
        fdset_to_list(Set, Lista),
        length(Lista, Len),
        random(0, Len, RandomIndex),
        nth0(RandomIndex, Lista, BestValue).
```

4 Visualização da Solução

Para visualizar a solução em modo de texto é preciso colocar as árvores e tents (ambas em formato de listas de posições) num tabuleiro vazio. Utilizaram-se os predicados: createEmptyBoard(SizeBoard, EmptyBoard) e putTentsAndTreesInBoard(EmptyBoard, Trees, Tents, SizeBoard, AuxBoard, Board).

Por fim, criou-se o predicado printBoard(Board, RowTents, ColTents), que desenha o tabuleiro e o número de tents nas linhas e colunas.

Para além disso, é mostrado o tempo de execução do solver, assim como as estatísticas resultantes da busca de solução.

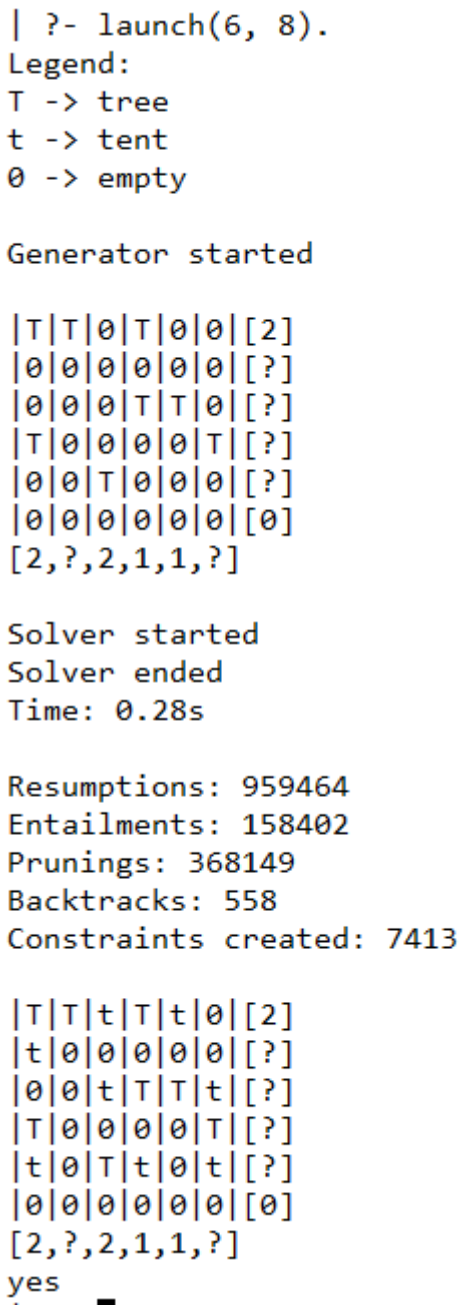

Figura 2: Exemplo de execução do programa

5 Resultados

Testou-se o programa para um número crescente de árvores e tents a colocar para tabuleiro de tamanho 6x6 e 20x20. Com isto percebeu-se que o número de árvores contribui mais significativamente em custo temporal do que a densidade to tabuleiro (número de árvores / tamanho do tabuleiro). Interrompeu-se o programa a partir dos 5 minutos de execução.

Também se testou o programa com várias opções de labeling. Daí pode-se concluir que a opção bissect na seleção de valores melhora muito a eficiência do programa, pois exclui valores outliers. Não se nota muita diferença entre a opção ffc e leftmost(por defeito) na seleção de variáveis, porque o número de restrições das variáveis são iguais entre si ao longo do tempo e os valores de domínio das variáveis são iguais.

Número de árvores	Tempo médio de execução (s)
5	0.0
6	0.0
7	0.08
8	0.04
9	0.03

Figura 3: Resultados num tabuleiro 6x6

Número de árvores	Tempo médio de execução (s)	Número de casos tempo > 5 minutos
9	0.14	0
10	0.49	0
11	0.11	0
12	4.09	0
13	1.6	0
14	5.11	0
15	122.05	0
16	219.33	1
17	377.17	1
18	208.82	0
19	320.01	1
20	275.49	2

Figura 4: Resultados num tabuleiro 20x20

lables\ (tempo (s) /backtracks)	Execução 1	Execução 2	Execução 3	Execução 4	Execução 5	Média
ffc, bisect	0.79/41885	0.91/1086	9.83/3140	10.2/3400	3.86/6374	5.118/11177
ffc, enum	105.73/850521	24.85/753124	293.42/762380	>5minutos	132.42/642052	139.11/752019
ffc, step	>5minutos	>5minutos	>5minutos	>5minutos	>5minutos	?
leftmost, bisect	1.36/43370	18.13/33651	0.78/17277	1.88/996	6.24/2138	5.678/19486

Figura 5: Resultados das diferentes opções do labeling

6 Conclusões e Trabalho Futuro

Com este trabalho aprofundamos o conhecimento em programação em lógica com restrições. A maior dificuldade sentida foi declarar as variáveis de modo a que representasse bem o problema e, ao mesmo tempo, permitisse que aplicação de restrições fosse simples. A primeira maneira pensada e mais intuitiva de declaração de variáveis foi assumir que cada célula do tabuleiro fosse uma variável. Alteramos essa abordagem, pois dessa a maior parte das variáveis declaradas(células vazias) não eram usadas nas restrições e seria mais difícil garantir que não haviam tents sem árvores vizinhas e àrvores sem tents vizinhos. Ao formular o problema da forma evidenciada no relatório, resolveu-se o problema anterior e tornou-se mais fácil ligar cada tent à sua respetiva àrvore.

A Anexo

A.1 tents.pl

```
1  :- use_module(library(clpfd)).
2  :- use_module(library(random)).
3  :- use_module(library(lists)).
4  :- include('restrictions.pl').
5
6  :- dynamic numPositions/1.
7
8
9  /*
10  * FEUP - 2016, Programacao em logica
11  * Autores: Helder Antunes e Ines Proenca
12  * Utilizacao: executar o comando "launch(TamanhoDoTabuleiro,
        NumeroDeArvores)".
13  */
14
15 launch(SizeBoard, NumTrees) :-
16         printLegend,
17         write('Generator started'), nl, nl,
18         generatePuzzle(SizeBoard, NumTrees, Trees, RowTents, ColTents),
19         write('Solver started'), nl,
20         solvePuzzle(SizeBoard, NumTrees, Trees, RowTents, ColTents).
21
22 generatePuzzle(SizeBoard, NumTrees, Trees, RowTents, ColTents) :-
23     length(Trees, NumTrees),
24     length(Tents, NumTrees),
25     NumPositions is SizeBoard * SizeBoard - 1,
26         append([Trees, Tents], Vars),
27     domain(Vars, 0, NumPositions),
28
29     % restricoes
30          all_distinct(Vars),
31     tentIsAroundHisTree_restriction(Trees, Tents, SizeBoard),
32     tentsDontTouchEachOther_restriction(Tents, Tents, SizeBoard),
33         labeling([value(mySelValores)], Vars),!,
34
35         % preparar tabuleiro
36     createEmptyBoard(SizeBoard, EmptyBoard),
37         putTreesInBoard(EmptyBoard, Trees, SizeBoard, _, Board),
38         length(RowTentsAux, SizeBoard),
39         length(ColTentsAux, SizeBoard),
40         setToZero(RowTentsAux),
41         setToZero(ColTentsAux),
42
43         % determinar o numero de tents nas linhas e colunas
44         setNumTentsInRows(RowTentsAux, RowTentsAux2, Tents, SizeBoard
               ),
45         setNumTentsInCols(ColTentsAux, ColTentsAux2, Tents, SizeBoard),
46         makeDisappearSomeValues(RowTentsAux2, RowTents),
```

```
47          makeDisappearSomeValues(ColTentsAux2, ColTents),
48
49      printBoard(Board, RowTents, ColTents), nl.
50
51  solvePuzzle(SizeBoard, NumTrees, Trees, RowTents, ColTents) :-
52          length(Tents, NumTrees),
53          NumPositions is SizeBoard * SizeBoard - 1,
54          domain(Tents, 0, NumPositions),
55
56          % restricoes
57           all_distinct (Tents),
58          tentIsAroundHisTree_restriction(Trees, Tents, SizeBoard),
59          tentsDontTouchEachOther_restriction(Tents, Tents, SizeBoard),
60          numTentsInRows_restriction(RowTents, Tents, 1, SizeBoard),
61          numTentsInCols_restriction(ColTents, Tents, 1, SizeBoard),
62
63          reset_timer ,
64          labeling ([ ffc , bisect ], Tents),
65          print_time ,
66           fd_statistics , nl,
67
68          % preparar tabuleiro
69          createEmptyBoard(SizeBoard, EmptyBoard),
70          putTentsAndTreesInBoard(EmptyBoard, Trees, Tents, SizeBoard, _,
                Board),
71
72          printBoard(Board, RowTents, ColTents).
73
74  reset_timer :- statistics(walltime,_).
75  print_time :-
76          statistics(walltime,[_,T]),
77          TS is ((T//10)*10)/1000,
78          write('Solver ended'), nl,
79          write('Time: '), write(TS), write('s'), nl, nl.
80
81  mySelValores(Var, _Rest, BB, BB1) :-
82           fd_set (Var, Set),
83           select_best_value (Set, Value),
84          (
85              first_bound(BB, BB1), Var #= Value
86          ;
87              later_bound(BB, BB1), Var #\= Value
88      ).
89
90   select_best_value (Set, BestValue):-
91           fdset_to_list (Set, Lista),
92          length(Lista, Len),
93          random(0, Len, RandomIndex),
94          nth0(RandomIndex, Lista, BestValue).
95
96  % Cria um tabuleiro SizeBoard x SizeBoard com celulas a zero.
97  createEmptyBoard(SizeBoard, EmptyBoard) :-
```

```
98      length(EmptyBoard, SizeBoard),
99      createEmptyBoardAux(SizeBoard, EmptyBoard).
100
101 createEmptyBoardAux(_, []).
102 createEmptyBoardAux(SizeBoard, [H|T]) :-
103     length(H, SizeBoard),
104     fillRow(H),
105     createEmptyBoardAux(SizeBoard, T).
106
107 fillRow ([]) .
108 fillRow ([0| T]) :- fillRow(T).
109
110 % Coloca t's (tents) e T's (trees) num tabuleiro vazio.
111 putTentsAndTreesInBoard(BoardIn, [], [], _, _, BoardIn).
112 putTentsAndTreesInBoard(BoardIn, [Tree|Trees], [Tent|Tents],
113                         SizeBoard, BoardAux, BoardOut) :-
114     RowTree is Tree div SizeBoard + 1,
115     ColTree is Tree mod SizeBoard + 1,
116     RowTent is Tent div SizeBoard + 1,
117     ColTent is Tent mod SizeBoard + 1,
118     setPiece(RowTree, ColTree, 'T', BoardIn, BoardInAux),
119     setPiece(RowTent, ColTent, 't', BoardInAux, BoardAux),
120     putTentsAndTreesInBoard(BoardAux, Trees, Tents, SizeBoard, _,
            BoardOut).
121
122 % Coloca T's (trees) num tabuleiro vazio.
123 putTreesInBoard(BoardIn, [], _, _, BoardIn).
124 putTreesInBoard(BoardIn, [Tree|Trees],
125                         SizeBoard, BoardAux, BoardOut) :-
126     RowTree is Tree div SizeBoard + 1,
127     ColTree is Tree mod SizeBoard + 1,
128     setPiece(RowTree, ColTree, 'T', BoardIn, BoardAux),
129     putTreesInBoard(BoardAux, Trees, SizeBoard, _, BoardOut).
130
131 % Coloca uma peca numa celula do tabuleiro.
132 setPiece(1, Column, Piece, [B|Bs], [ModRow|Bs]):-
133         setPieceInRow(Column, Piece, B, ModRow), !.
134 setPiece(Row, Column, Piece, [B|Bs], [B|Rs]):-
135         Row > 1,
136         R1 is Row -1,
137         setPiece(R1, Column, Piece, Bs, Rs).
138
139 setPieceInRow(1, Piece, [_|RowIn], [Piece|RowIn]):- !.
140 setPieceInRow(Col, Piece, [R|RowIn], [R|RowOut]):-
141         Col > 1,
142         C1 is Col -1,
143         setPieceInRow(C1, Piece, RowIn, RowOut).
144
145 % Coloca uma lista a zero
146 setToZero([]) .
147 setToZero([0|T]) :- setToZero(T).
148
```

```
149 % Set number of tents in each row.
150 setNumTentsInRows(RowIn, RowIn, [], _).
151 setNumTentsInRows(RowIn, RowOut, [Tent|Tents], SizeBoard) :-
152         Row is Tent // SizeBoard,
153         nth0(Row, RowIn, Val),
154         NewVal is Val + 1,
155         setValInList(Row, RowIn, RowAux, NewVal),
156         setNumTentsInRows(RowAux, RowOut, Tents, SizeBoard).
157
158 % Set number of tents in each collumn.
159 setNumTentsInCols(ColIn, ColIn, [], _).
160 setNumTentsInCols(ColIn, ColOut, [Tent|Tents], SizeBoard) :-
161         Col is Tent mod SizeBoard,
162         nth0(Col, ColIn, Val),
163         NewVal is Val + 1,
164         setValInList(Col, ColIn, ColAux, NewVal),
165         setNumTentsInCols(ColAux, ColOut, Tents, SizeBoard).
166
167 % Cria uma lista Out, que e copia de uma lista In em todos os indices
168 % exceto o especificado em Index que vai ter o valor Value.
169 setValInList(0, [_|Tin], [Value|Tin], Value).
170 setValInList(Index, [Hin|Tin],[Hin|Tout], Value) :-
171                 Index > 0,
172                 NextIndex is Index - 1,
173                 setValInList(NextIndex, Tin, Tout, Value).
174
175 % Substitui aleatoriamente alguns valores da lista por -1.
176 makeDisappearSomeValues([], []).
177 makeDisappearSomeValues([H|T], [H_out|T_out]) :-
178         random(0, 3, Num),
179         (Num =:= 0,
180                 H_out = -1;
181         Num >= 1,
182                 H_out = H
183         ),
184         makeDisappearSomeValues(T, T_out).
185
186 % Desenha o tabuleiro
187 printBoard(Board, RowTents, ColTents) :-
188     printAuxBoard(Board, RowTents),
189         write('['), printDownBorder(ColTents).
190
191 printAuxBoard([], []).
192 printAuxBoard([H|T], [TentsRow|TentsRows]) :-
193     write('|'), printRow(H, TentsRow),
194     printAuxBoard(T, TentsRows).
195
196 printRow([], -1) :- write('['), write('?'), write(']'), nl.
197 printRow([], TentsRow) :- write('['), write(TentsRow), write(']'), nl.
198 printRow([H|T], TentsRow) :-
199     write(H), write('|'),
200     printRow(T, TentsRow).
```

```
201
202 printDownBorder([-1]) :- write('?]'), nl.
203 printDownBorder([H]) :- write(H), write(']'), nl.
204 printDownBorder([-1|T]) :-
205         write('?,'),
206         printDownBorder(T).
207 printDownBorder([H|T]) :-
208         write(H), write(','),
209         printDownBorder(T).
210
211 printLegend :-
212         write('Legend:'), nl,
213         write('T -> tree'), nl,
214         write('t -> tent'), nl,
215         write('0 -> empty'), nl, nl.
```

A.2 restrictions.pl

```
1  :- use_module(library(clpfd)).
2  :- use_module(library(random)).
3  :- use_module(library(lists)).
4
5  % Coloca a restricao: o tent esta a tocar a sua arvore horizontalmente ou
       verticalmente.
6  tentIsAroundHisTree_restriction([], [], _).
7  tentIsAroundHisTree_restriction([Tree|Trees], [Tent|Tents], SizeBoard):-
8          %Toca horizontalmente
9  ((Tree #= Tent - 1 #\/ Tree #= Tent + 1) #/\ Tree / SizeBoard #= Tent
       / SizeBoard )
10         %Toca Verticalmente
11     #\/ (Tree #= Tent - SizeBoard) #\/ (Tree #= Tent + SizeBoard),
12     tentIsAroundHisTree_restriction(Trees, Tents, SizeBoard).
13
14 % Colaca a restricao: tents nao sao vizinhos entre si.
15 tentsDontTouchEachOther_restriction([], _, _).
16 tentsDontTouchEachOther_restriction([T|Ts], Tents, SizeBoard) :-
17     tentIsIsolated(T, Tents, SizeBoard),
18     tentsDontTouchEachOther_restriction(Ts, Tents, SizeBoard).
19
20 % Coloca a restricao: um tent nao e vizinho de nenhum outro tent.
21 tentIsIsolated(_, [], _).
22 tentIsIsolated(Tent, [T|Ts], SizeBoard) :-
23         % toques horizontais
24     (((Tent #= T + 1 #\/ Tent #= T - 1) #/\ Tent / SizeBoard #= T /
           SizeBoard)
25         % toques verticais
26      #\/ Tent #= T + SizeBoard #\/ Tent #= T - SizeBoard #\/
27         % toques diagonais
28     (Tent #= T + SizeBoard + 1 #/\ T / SizeBoard #= Tent / SizeBoard
           - 1) #\/
```

```
29      (Tent #= T + SizeBoard - 1 #/\ T / SizeBoard #= Tent / SizeBoard
            - 1) #\/
30      (T #= Tent + SizeBoard + 1 #/\ Tent / SizeBoard #= T / SizeBoard
            - 1) #\/
31      (T #= Tent + SizeBoard - 1 #/\ Tent / SizeBoard #= T / SizeBoard
            - 1)) #<=> B,
32          B #= 0,
33      tentIsIsolated(Tent, Ts, SizeBoard).
34
35  % Coloca a restricao: Na linha x existem y tents.
36  numTentsInRows_restriction(_, _, Row, SizeBoard) :- Row > SizeBoard.
37  numTentsInRows_restriction(RowTents, Tents, Row, SizeBoard) :-
38          Row =< SizeBoard,
39          nth1(Row, RowTents, -1), !, % numero de tents nao determinado
40          NextRow is Row + 1,
41          numTentsInRows_restriction(RowTents, Tents, NextRow, SizeBoard).
42  numTentsInRows_restriction(RowTents, Tents, Row, SizeBoard) :-
43          Row =< SizeBoard,
44          countTentsInRow(Tents, Row, SizeBoard, Total),
45          element(Row, RowTents, Total),
46          NextRow is Row + 1,
47          numTentsInRows_restriction(RowTents, Tents, NextRow, SizeBoard).
48
49  % Conta o numero de tents na linha Row.
50  countTentsInRow([], _, _, 0).
51  countTentsInRow([Tent|Tents], Row, SizeBoard, Total) :-
52          Tent / SizeBoard + 1 #= Row #<=> B,
53          Total #= B + TotalAux,
54          countTentsInRow(Tents, Row, SizeBoard, TotalAux).
55
56  % Coloca a restricao: Na coluna x existem y tents.
57  numTentsInCols_restriction(_, _, Col, SizeBoard) :- Col > SizeBoard.
58  numTentsInCols_restriction(ColTents, Tents, Col, SizeBoard) :-
59          Col =< SizeBoard,
60          nth1(Col, ColTents, -1), !, % numero de tents nao determinado
61          NextCol is Col + 1,
62          numTentsInCols_restriction(ColTents, Tents, NextCol, SizeBoard).
63  numTentsInCols_restriction(ColTents, Tents, Col, SizeBoard) :-
64          Col =< SizeBoard,
65          countTentsInCol(Tents, Col, SizeBoard, Total),
66          element(Col, ColTents, Total),
67          NextCol is Col + 1,
68          numTentsInCols_restriction(ColTents, Tents, NextCol, SizeBoard).
69
70  % Conta o numero de tents na coluna Col.
71  countTentsInCol([], _, _, 0).
72  countTentsInCol([Tent|Tents], Col, SizeBoard, Total) :-
73          Tent mod SizeBoard + 1 #= Col #<=> B,
74          Total #= B + TotalAux,
75          countTentsInCol(Tents, Col, SizeBoard, TotalAux).
```