Preface

This article aims to be used by students of the curricular unit of Logic Programming (PLOG) of the Faculty of Engineering of the University of Porto (FEUP). The reader should have basic knowledge of the programming language prolog and knowledge of the use of restrictions in this same language.

For PLOG students the project developed in this article will provide a basis for understanding the techniques of using the prolog language at a more advanced level.

Dezembro 2017

Joel Carneiro
Trid Game

Trid Game solved by Prolog Constraints

ei11053@fe.up.pt
FEUP-PLOG
3MIEIC03
Grupo Trid_5

Faculdade de Engenharia da Universidade do Porto
Home page: http://fe.up.pt

Abstract. The use of the programming language Prolog with constraints allows the elaboration of programs that solve complex problems faster than using other programming languages. This article introduces the development of the Trid game using prolog with constraints. SicStus Prolog is used to test the program. A program capable of simulating the game with n-sized puzzles was achieved.

Keywords: prolog, sicstus, restrictions

1 Introduction

The work developed has the aim of creating a program with the resort to prolog constraints that can solve the trid puzzle, in n-sized puzzles. The program has to be capable of create a new Trid puzzle and then solve it. It was developed a text interface in order to better test and visualize the results.

This article will be divided as follows: Firstly it will be described the Trid rules. After that, it will be specified the approach used to get the constraints, variables of decision and search strategy. Thirdly, it is explained the visualization of the solution and lastly, the results will be analyzed and presented the conclusions.

2 Trid

The goal of the Trid game is to fill all the circles, equivalent to the vertices of each triangle so that there is no number repeated along any line (horizontal or diagonal). In addition, the numbers within the triangles are the result of the sum of the three vertices of that same triangle. There is one last constraint in the game, a restriction that delimits the domain of the values used to fill the vertices. In another way, we give the minimum and maximum values that can be used to fill the circles of the vertices of the triangles [1]. Figure 1 represents a solution with domain values between 1 and 5.

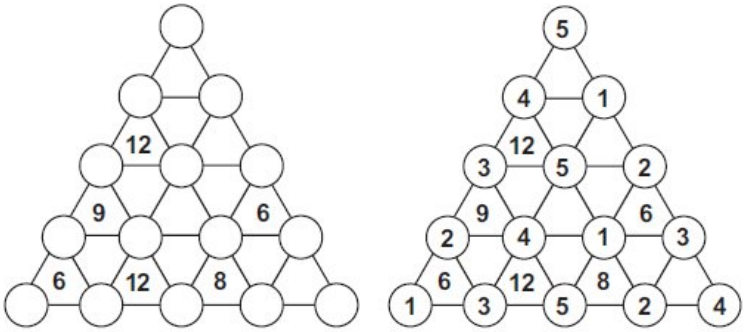

Fig. 1. Initial puzzle and a solution with domain values between 1 and 5.

3 Approach

It is possible to generalize the constraints of this puzzle game in 3 main ones: 1) domain of the values used in the circles (vertices of the triangles) must be in the interval specified; 2) The sum of the three vertices of a triangle is the result presented inside of the same triangle; 3) The values used in the vertices cannot be repeated along any straight line (diagonal or horizontal).

Due to that the program developed has two stages: 1) the program solves the puzzle without any value in the triangles and vertices variables; 2) the program solve a puzzle that has four random inner triangles values of the previous solution. Thus, it is guaranteed that the generated puzzle in 1 has a solution when the program tries to solve him in the 2 stage.

3.1 Decision Variables

To solve correctly the puzzle the decision variables are the triangles' vertices. The domain applied to them is chosen by the user. Thus, that implies if the user inputs a domain that has a smaller length than the longest straight line of the puzzle, there is no solution. This is an intuitive constraint of the game, according to the main ones.

3.2 Constraints

To translate the constraints to the Prolog program it was used predicates that receive a list of vertices variables an then apply the restrictions. The variables' list is a list of list. Each list inside of the main list has the size of the row at the puzzle: the first list has 1 variable, the second list has 2, the third list has 3, and so on. Using this list are applied the constraints over the lines (all different numbers), and the constraints that force that the sum of the 3 vertices of a triangle is the number inside it.

Furthermore, the minimum and the maximum values that user inserted in the program are used to restrict the variables used in the labeling predicate.

To a better analysis, figures 2,3,4 and 5 have the predicates used.

```
innerRestrictions(Vertices,X,Y,TridSize):- Tridsizemenos1 is TridSize - 1,
        X == Tridsizemenos1,
        diagonal1(Vertices,X,1,TridSize,ResultList),
        all_different(ResultList),
        diagonal2(Vertices,X,Y,TridSize,ResultList2),
        all_different(ResultList2).
innerRestrictions(Vertices,X,Y,TridSize):- Tridsizemenos1 is TridSize - 1,
        X < Tridsizemenos1,
        diagonal1(Vertices,X,1,TridSize,ResultList),
        all_different(ResultList),
        diagonal2(Vertices,X,Y,TridSize,ResultList2),
        all_different(ResultList2),
        X1 is X + 1,
        Y1 is Y + 1,
        innerRestrictions(Vertices,X1,Y1,TridSize).
```

Fig. 2. Predicate that gets the constraints in the inner of the puzzle, using other predicates too.

```
allfinals([],[]).
allfinals([Head|Tail],[H|T]):-
        length(Head,SizeH),
        nth1(SizeH,Head,H),
        allfinals(Tail,T).

%d:- diagonal1([['A'],['B','C'],['D','E','F'],['G','H','I','J'],['K','L','M','N','O']],2,1,5, Rs), write(Rs
diagonal1(Vertices,X,Y,TridSize,[H|[]]):- X == TridSize,
        nth1(X,Vertices,ListaR),
        nth1(Y,ListaR,H).
diagonal1(Vertices,X,Y,TridSize,[H|T]):- X < TridSize,
        nth1(X,Vertices,ListaR),
        nth1(Y,ListaR,H),
        X1 is X + 1,
        Y1 is Y + 1,
        diagonal1(Vertices,X1,Y1,TridSize, T).

%d:- diagonal2([['A'],['B','C'],['D','E','F'],['G','H','I','J'],['K','L','M','N','O']],2,1,5, Rs), write(Rs
diagonal2(Vertices,X,Y,TridSize,[H|[]]):- X == TridSize,
        nth1(X,Vertices,ListaR),
        nth1(Y,ListaR,H).
diagonal2(Vertices,X,Y,TridSize,[H|T]):- X < TridSize,
        nth1(X,Vertices,ListaR),
        nth1(Y,ListaR,H),
        X1 is X + 1,
        diagonal2(Vertices,X1,Y,TridSize, T).
```

Fig. 3. allfinals predicate gets the constraints in the right diagonal line of the puzzle. diagonal1 and diagonal2 predicates get the constraints of all diagonals in the inner of the puzzle.

```
restrictions([Hvertices,H2ver|_],[Htriangles|_], 1):-
        nth1(1,Hvertices,A),
        nth1(1,H2ver,B),
        nth1(2,H2ver,C),
        nth1(1,Htriangles,A1),
        A+B+C #= A1.

restrictions(Vertices,Triangles, N):-
        nth1(N,Vertices,ListaVerticesN),
        N1 is N + 1,
        nth1(N1,Vertices,ListaVerticesN1),
        nth1(N,Triangles,ListaTriangles),
        length(ListaTriangles,X),
        restrictionsAux(ListaVerticesN,ListaVerticesN1, ListaTriangles,X, 1).

restrictionsAux([],[],[],_,_).
restrictionsAux([A,B|TVerticesN],[C,D|TVerticesN1], [A1,B1|TListaTriangles],X, Tinicial):- Tinicial < X,
        A+C+D #= A1,
        A+B+D #= B1,
        Tinicial1 is Tinicial + 2,
        restrictionsAux([B|TVerticesN],[D|TVerticesN1],TListaTriangles,X, Tinicial1).

restrictionsAux([A|_],[C,D|_], [A1|_],X, Tinicial):- Tinicial == X,
        A+C+D #= A1.
```

Fig. 4. restrictions predicate gets the constraints of the sum of the vertices of each triangle in an n-sized puzzle.

```
%faz as restricoes das linhas que fazem as arestas de todos os triangulos
lineRestrictions(Vertices,TridSize):- innerRestrictions(Vertices,2,2,TridSize),
        allfirsts(Vertices,ResultF),
        all_different(ResultF),
        allLinesDistinct(Vertices),
        allfinals(Vertices,ResultFinals),
        all_different(ResultFinals).
```

Fig. 5. linerestrictions predicate gets the constraints of the lines of the puzzle using the other predicates above too.

3.3 Evaluation Function

In order to evaluate the obtained solution the way found is looking to the text interface of the program and confirm is the values are right. However, to puzzles with a big size, it will be more complicated to evaluate them.

3.4 Search Strategy

The search strategy used in this program is the default. When labeling is called the parameters passed to it is an empty list. With that, sicstus assume the default strategy, that is, the domain is explored in ascending order [2]. The down strategy was tested, however, the results, in terms of time, are not good. The bisect strategy was also tested and the time required for the solution increased in about 15 seconds when compared with the default. The step strategy was the one that had a performance almost equal to the default strategy. All tests were made with a puzzle that has 100 variables in the largest straight line.

4 Solution Presentation

To print the solution was developed the printBoard predicate that uses the printListV and the printListT predicates to print the solution on the screen. It also uses the espaces predicate to align the triangle in the output. The predicate receives two lists of lists. One represents the vertices and the other the triangles variables. The predicate print the line n of each list respectively until prints all the variables in the list. To a better comprehension, figure 6 presents the predicates.

```
/*Printing an entire board 5*5*/
printBoard([HVar|[]],_,TridSize):- espaces(TridSize),
        printListV(HVar).
printBoard([HVar|Tvar], [HTri|TTri],TridSize) :- espaces(TridSize),
        printListV(HVar),
        espaces(TridSize),
        write(' '),
        printListT(HTri),
        FinalSize is TridSize - 1,
        printBoard(Tvar,TTri, FinalSize).

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

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

espaces(0).
espaces(N):- write(' '),
        N1 is N - 1,
        espaces(N1).
```

Fig. 6. Printing predicates.

5 Results

In this section, it will be present some results using different sizes and domains. When it is requested a puzzle with n equal to 50 and the minimum and maximum of the domain are 1 and 75, respectively, the results are:

Firstly, the representations of the puzzle and the following statistics:
The execution took 489 ms. The execution took 0.489 seconds.
Resumptions: 57103
Entailments: 2548
Prunings: 98822
Backtracks: 0
Constraints created: 2548

When it is requested a puzzle with n equal to 100 and the minimum and maximum of the domain are 1 and 128, respectively, the results are:

Firstly, the representations of the puzzle and the following statistics:
The execution took 6388 ms. The execution took 6.388 seconds.
Resumptions: 357544
Entailments: 10098
Prunings: 703113
Backtracks: 0
Constraints created: 10098

When it is requested a puzzle with n equal to 200 and the minimum and maximum of the domain are 1 and 290, respectively, the results are:

Firstly, the representations of the puzzle and the following statistics:
The execution took 77819 ms. The execution took 77.819 seconds.
Resumptions: 2370822
Entailments: 40198
Prunings: 5176174
Backtracks: 0
Constraints created: 40198

Analyzing this results, and the other ones not presented here, we can say that the program is fast even when we have a lot of variables to calculate. 77.819 seconds for 200 variables, based on the 3 main constrictions of the game is a really good result. Thus, it is possible to see the power of Prolog language with constraints.

6 Conclusions and Future Work

After analyzing the results it is possible to assume that the program presented is capable of solving the n-sized puzzles of the Trid game. Also by using the constraints in the Prolog language the program solve the problems in a really

fast way, comparing to the normal way of solving this kind of problems. To get a completely amazing solution only is need to align better the triangles in the program's output. Maybe if the labeling options are studied in a more deep way a faster result can be obtained.

References

1. The Logical World of Puzzles: Rules of 'Trid'.
2. SICStus Prolog - Constraint Logic Programming over Finite Domains.

7 Code

```
:- include('print.pl').
:- include('generictrid.pl').
:- use_module(library(system)).
:- use_module(library(lists)).
:- use_module(library(random)).
:- use_module(library(clpfd)).


trid:-
        write('************************************************'),nl,
        write('********   Trid - PLOG - Version 1.0   ***********'),nl,
        write('************************************************'),nl,
        nl,nl,
        nl, write('1 - Play Game'),
        nl, write('2 - Exit Game'),nl,
        write('Choose : '),nl,nl,
        read(Choice),
        menu(Choice).

menu(Choice):- Choice == 1,
        genericChoose.
menu(Choice):- Choice == 2,
        exit(_).

endGame(_):-
        write('************************************************'),nl,
        write('**************  Trid Version 1.0 *****************'),nl,
        write('************************************************'),nl,
        nl,nl,nl.

exit(_) :- nl,nl,write('See you later !!!! '),nl,nl.



/*
   L is the current list,
   P is the position where the element will be insert,
   E is the element to insert and
   R is the return, the new list
*/
replaceInThePosition(L, P, E, R) :-
```

```
        findall(X, (nth0(I,L,Y), (I == P -> X=E ; X=Y)), R).


randomvalues(R1,R2,R3,R4, TrianglesSize):- Final is TrianglesSize - 1,
        random(1,Final, R1),
        random(1,Final, R2),
        random(1,Final, R3),
        random(1,Final, R4),
        all_different([R1,R2,R3,R4]),
        labeling([],[R1,R2,R3,R4]).



:- use_module(library(clpfd)).
:- use_module(library(random)).

genericChoose:-
        write('Choose the Game Size: '),nl,
        read(TridSize),
        write('Choose the MinValue: '),nl,
        read(MinValue),
        write('Choose the MaxValue: '),nl,
        read(MaxValue),
        tridplayerGeneric(MinValue,MaxValue,TridSize).

%gera as listas e chama os predicados que , primeiro vao gerar o problema e depi
tridplayerGeneric(MinValue,MaxValue,TridSize) :-
        TridSizemenos1 is TridSize - 1,
        write('A gerar problema...'),nl,nl,
        createTrid(MinValue,MaxValue,VariablesList1,TrianglesList,Values,
        TridSize,TridSizemenos1,Total,TotalTri,ExecutionTime,Seconds),
        % imporar numeros random da solucao obtida
        randomvalues(R1,R2,R3,R4,TotalTri),
        nth1(R1,TrianglesList,Valor1),
        nth1(R2,TrianglesList,Valor2),
        nth1(R3,TrianglesList,Valor3),
        nth1(R4,TrianglesList,Valor4),
        update(R1,R2,R3,R4,C1,C2,C3,C4),
        length(VariablesListPlay,Total),
        length(TrianglesListPlay,TotalTri),
        replaceInThePosition(TrianglesListPlay, C1, Valor1, RList),
        replaceInThePosition(TrianglesListPlay, C2, Valor2, RList),
        replaceInThePosition(TrianglesListPlay, C3, Valor3, RList),
        replaceInThePosition(TrianglesListPlay, C4, Valor4, RList),
        generateTrianglesListFrom(RList,1,RT,TridSizemenos1),
        generateVariablesListFrom(VariablesList1,1,RV,TridSize),
```

```
        %print do tabuleiro a resolver
        printBoard(RV,RT, TridSize),nl,nl,
        write('Create the Trid puzzle took '),
        write(ExecutionTime), write(' ms.'),nl,
        write('Create the Trid puzzle took '),
        write(Seconds), write(' seconds.'),nl,
        nl,fd_statistics,nl,nl,
        write('A resolver o problema...'),nl,nl,
        %resolver trid com base no ja calculado
        solveTrid(VariablesListPlay,RList,Values,TridSize,TridSizemenos1).


%gera todo o problema
createTrid(MinValue,MaxValue,VariablesList1,TrianglesList,Values,TridSize,
TridSizemenos1,Total,TotalTri,ExecutionTime,Seconds):-
        count(TridSize,Total),
        length(VariablesList,Total),
        length(VariablesList1,Total),
        countp(TridSizemenos1,TotalTri),
        length(TrianglesList,TotalTri),
        Values = [MinValue,MaxValue],
        statistics(walltime, [_ | [_]]),
        playGameGenericAuto(VariablesList,TrianglesList,Values,TridSize),
        statistics(walltime, [_ | [ExecutionTime]]),
        Seconds is ExecutionTime / 1000.

%da a solicao final
solveTrid(VariablesListPlay,RList,Values,TridSize,TridSizemenos1):-
statistics(walltime, [_ | [_]]),
        playGameGeneric(VariablesListPlay,RList,Values,TridSize),
        statistics(walltime, [_ | [ExecutionTime]]),
        generateVariablesListFrom(VariablesListPlay,1,RV1,TridSize),
        generateTrianglesListFrom(RList,1,RT1,TridSizemenos1),
        printBoard(RV1,RT1,TridSize),nl,nl,nl,
        write('Execution took '), write(ExecutionTime), write(' ms.'),nl,
        Seconds is ExecutionTime / 1000,
        write('Execution took '), write(Seconds), write(' seconds.'),nl,
        nl,fd_statistics.


%gera o problema
playGameGenericAuto(Vertices,Triangles,[MinValue,MaxValue], TamanhoTrid) :-
        domain(Vertices,MinValue,MaxValue),
        TamanhoTridMenos1 is TamanhoTrid - 1,
        generateVariablesListFrom(Vertices,1,RV,TamanhoTrid),
```

```
        generateTrianglesListFrom(Triangles,1,RT,TamanhoTridMenos1),
        restricoes(RV,RT,TamanhoTridMenos1),
        lineRestrictions(RV,TamanhoTrid),!,
        labeling([],Vertices).

%resolve o problema
playGameGeneric(Vertices,Triangles,[MinValue,MaxValue], TamanhoTrid) :-
        domain(Vertices,MinValue,MaxValue),
        TamanhoTridMenos1 is TamanhoTrid - 1,
        generateVariablesListFrom(Vertices,1,RV,TamanhoTrid),
        generateTrianglesListFrom(Triangles,1,RT,TamanhoTridMenos1),
        restricoes(RV,RT,TamanhoTridMenos1),
        lineRestrictions(RV,TamanhoTrid),!,
        labeling([],Vertices).



%-------------------------

%faz as restricoes das linhas que fazem as arestas de todos os triangulos
lineRestrictions(Vertices,TridSize):- innerRestrictions(Vertices,2,2,TridSize),
        allfirsts(Vertices,ResultF),
        all_different(ResultF),
        allLinesDistinct(Vertices),
        allfinals(Vertices,ResultFinals),
        all_different(ResultFinals).


%     restricoes([['A'],['B','C'],['D','E','F'],['G','H','I','J']],[['A1'],['B1',
%     restricoes([['A'],['B','C'],['D','E','F'],['G','H','I','J'], ['K','L','M','
restricoes(_,_,0).
restricoes(Vertices,Triangles,TamanhoTridMenos1):- restrictions(Vertices,
Triangles,TamanhoTridMenos1),
        Aux is TamanhoTridMenos1 - 1,
        restricoes(Vertices,Triangles,Aux).

%       restrictions([['A'],['B','C'],['D','E','F'],['G','H','I','J']],[['A1'],[
restrictions([Hvertices,H2ver|_],[Htriangles|_], 1):-
        nth1(1,Hvertices,A),
        nth1(1,H2ver,B),
        nth1(2,H2ver,C),
        nth1(1,Htriangles,A1),
        A+B+C #= A1.

restrictions(Vertices,Triangles, N):-
```

```
        nth1(N, Vertices, ListaVerticesN),
        N1 is N + 1,
        nth1(N1, Vertices, ListaVerticesN1),
        nth1(N, Triangles, ListaTriangles),
        length(ListaTriangles,X),
        restrictionsAux(ListaVerticesN, ListaVerticesN1, ListaTriangles,X, 1).

restrictionsAux([],[],[],_,_).
restrictionsAux([A,B|TVerticesN],[C,D|TVerticesN1], [A1,B1|TListaTriangles],X,
    Tinicial):- Tinicial < X,
        A+C+D #= A1,
        A+B+D #= B1,
        Tinicial1 is Tinicial + 2,
        restrictionsAux([B|TVerticesN],[D|TVerticesN1],TListaTriangles,X,
        Tinicial1).

restrictionsAux([A|_],[C,D|_], [A1|_],X, Tinicial):- Tinicial == X,
        A+C+D #= A1.


%c:- allfirsts([['A'],['B','C'],['D','E','F'],['G','H','I','J'], ['K','L','M','N
allfirsts(Ls, Rs) :-
        maplist(member, Rs, Ls).

%l:- allLinesDistinct([['A'],['B','C'],['D','E','F'],['G','H','I','J'], ['K','L'
allLinesDistinct([]).
allLinesDistinct([Head|Tail]):- all_different(Head),
        allLinesDistinct(Tail).


%f:- allfinals([['A'],['B','C'],['D','E','F'],['G','H','I','J'], ['K','L','M','N
allfinals([],[]).
allfinals([Head|Tail],[H|T]):-
        length(Head,SizeH),
        nth1(SizeH,Head,H),
        allfinals(Tail,T).

%d:- diagonal1([['A'],['B','C'],['D','E','F'],['G','H','I','J'],['K','L','M','N'
diagonal1(Vertices,X,Y,TridSize,[H|[]]):- X == TridSize,
        nth1(X, Vertices, ListaR),
        nth1(Y,ListaR,H).
diagonal1(Vertices,X,Y,TridSize,[H|T]):- X < TridSize,
        nth1(X, Vertices, ListaR),
        nth1(Y,ListaR,H),
        X1 is X + 1,
```

```
        Y1 is Y + 1,
        diagonal1(Vertices,X1,Y1,TridSize, T).

%d:- diagonal2([['A'],['B','C'],['D','E','F'],['G','H','I','J'],['K','L','M','N'
diagonal2(Vertices,X,Y,TridSize,[H|[]]):- X == TridSize,
        nth1(X,Vertices,ListaR),
        nth1(Y,ListaR,H).
diagonal2(Vertices,X,Y,TridSize,[H|T]):- X < TridSize,
        nth1(X,Vertices,ListaR),
        nth1(Y,ListaR,H),
        X1 is X + 1,
        diagonal2(Vertices,X1,Y,TridSize, T).


%i:- innerRestrictions([['A'],['B','C'],['D','E','F'],['G','H','I','J'],['K','L'
innerRestrictions(Vertices,X,Y,TridSize):- Tridsizemenos1 is TridSize - 1,
        X == Tridsizemenos1,
        diagonal1(Vertices,X,1,TridSize,ResultList),
        all_different(ResultList),
        diagonal2(Vertices,X,Y,TridSize,ResultList2),
        all_different(ResultList2).
innerRestrictions(Vertices,X,Y,TridSize):- Tridsizemenos1 is TridSize - 1,
        X < Tridsizemenos1,
        diagonal1(Vertices,X,1,TridSize,ResultList),
        all_different(ResultList),
        diagonal2(Vertices,X,Y,TridSize,ResultList2),
        all_different(ResultList2),
        X1 is X + 1,
        Y1 is Y + 1,
        innerRestrictions(Vertices,X1,Y1,TridSize).


%-------------------------

%generateVariablesListFrom([1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16],1,R,5).
generateVariablesListFrom([], _, [], _).
generateVariablesListFrom(L, N, [DL|DLTail],Size) :-
        length(DL, N),
        append(DL, LTail, L),
        N1 is N + 1,
        generateVariablesListFrom(LTail, N1, DLTail,Size).

generateTrianglesListFrom([], _, [], _).
generateTrianglesListFrom(L, 1, [DL|DLTail],Size) :-
        length(DL, 1),
```

```
        append(DL, LTail, L),
        generateTrianglesListFrom(LTail, 2, DLTail,Size).

%generateTrianglesListFrom([1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16],1,R,4).
generateTrianglesListFrom(L, N, [DL|DLTail],Sizemenos1) :-
        N1 is N - 1,
        SizeT is 2*N1 + 1,
        length(DL, SizeT),
        append(DL, LTail, L),
        NR is N + 1,
        generateTrianglesListFrom(LTail, NR, DLTail,Sizemenos1).

%-------------------------------------


%test:- generateVariablesList(5,VariablesList,1), write(VariablesList).
generateVariablesList(X,[Head|[]],X):-length(Head,X).
generateVariablesList(TridSize,[Head|Tail],N):- length(Head,N),
        N1 is N + 1,
        generateVariablesList(TridSize,Tail,N1).

%test:- generateTrianglesList(4,VariablesList,1), write(VariablesList).
generateTrianglesList(X,[Head|[]],X):- Nimpar is X - 1,
        Size is 2*Nimpar + 1,
        length(Head,Size).

generateTrianglesList(TridSizemenos1,[Head|Tail],1):- length(Head,1),
        generateTrianglesList(TridSizemenos1,Tail,2).

generateTrianglesList(TridSizemenos1,[Head|Tail],N):- Nimpar is N - 1,
        Size is 2*Nimpar + 1,
        length(Head,Size),
        N1 is N + 1,
        generateTrianglesList(TridSizemenos1,Tail,N1).

count(0,0).
count(TridSize,R):- New is TridSize - 1,
        count(New,R1),
        R is R1 + TridSize.

countp(0,0).
countp(TridSizemenos1,R):- New is TridSizemenos1 - 1,
        countp(New,R1),
        R is R1 + 2*New +1.
```

```
update(R1,R2,R3,R4,C1,C2,C3,C4):-
        C1 is R1 - 1,
        C2 is R2 - 1,
        C3 is R3 - 1,
        C4 is R4 - 1.



/*************************
* OUTPUT RELATED FUNCTIONS *
*************************/

%p:- printBoard([['A'],['B','C'],['D','E','F'],['G','H','I','J'], ['K','L','M','

/*Printing an entire board 5*5*/
printBoard([HVar|[]],_,TridSize):- espaces(TridSize),
        printListV(HVar).
printBoard([HVar|Tvar], [HTri|TTri],TridSize) :- espaces(TridSize),
        printListV(HVar),
        espaces(TridSize),
        write('␣'),
        printListT(HTri),
        FinalSize is TridSize - 1,
        printBoard(Tvar,TTri, FinalSize).

printListV([]):- nl.
printListV([H|T]):- write('('),
        write(H),
        write(')'),
        write('␣'),
        printListV(T).

printListT([]):- nl.
printListT([H|T]):- write(H),
        write('␣'),
        printListT(T).

espaces(0).
espaces(N):- write('␣'),
        N1 is N - 1,
        espaces(N1).
```