Implementing an *Izzi 2* Puzzle Solver with Constraint Logic Programming in Prolog

André Freitas, Rui Gonçalves

FEUP-PLOG, 3MIEIC4, Group 35

Abstract. In the context of PLOG course, this document describes an implementation of a puzzle solver with constraint logic programming in Prolog Language, using Sicstus. This paradigm of programming is very powerful since we just write the rules of the puzzle and the program tries to solve it. Izzi2 is the best example to practice this subject since is a kind of problem that we have some complex constraints to describe how a piece A can connect to piece B and how the solution is valid considering the geometry of the tiles.

Introduction

This project was developed in the Logic Programming course, to apply Constraint Logic Programming using the *clpfd* module in Sicstus Prolog. So, we choose the Izzi2 puzzle because although is simple, is very challenging since the pieces geometry is a diamond. This game is similar to Tangram, so the puzzle box come with the rhombuses and with a list of possible shapes. So, we must connect all the pieces with the same color in the connecting side, fitting a shape we have chosen from the list.

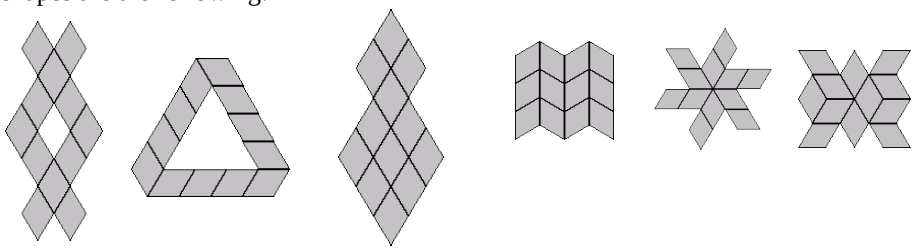
In this article is described the implementation and the steps we did, showing some analysis aspects that are the key of the success and that took a lot of hours to achieve, because this puzzle is very tricky in the implementation.

Problem Description

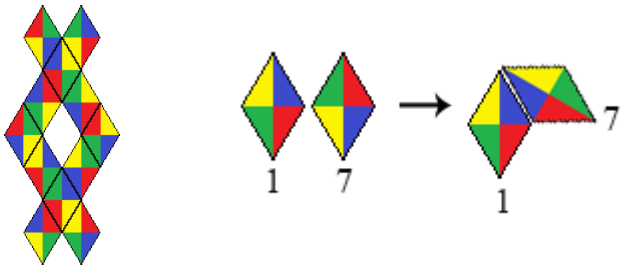
The Puzzle comes with 12 pieces and a list of possible shapes. The pieces are all rhombuses and each rhombus have 4 colors that are Red, Green, Blue and Yellow, so the pieces are all the possible combinations that are 4! divided by 2, because of the symmetry.

Then how to represent a piece in Prolog? It's very easy, is just a list of colors. So for example, for the first diamond, the code is: `p1([y,b,g,r])`

The puzzle have 34 shapes to solve and summing all the possibilities, there are 24!·224 = 10,409,396,852,733,332,453,861,621,760,000 solutions. Some of the 34 shapes are the following:



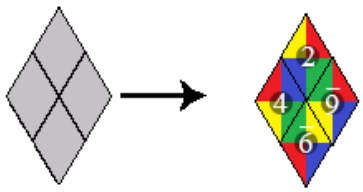
When a person what to solve a puzzle, must choose one shape first and then solve it. For the first shape, a solution can be the following.



Now we must choose a way to represent a connection between two pieces, like the previous illustration in the right. Since the pieces are 1 and 7, the connection is represented by `[1,7,2,4]` following the syntax `[piece1,piece2,piece1colorindex,piece2colorindex]` but this representation isn't enough because the orientation is necessary. So, if the piece is turned upside down, then the orientation is 1, otherwise if is the default position, is 0. Considering this, a more complete representation is `connect([1,7,2,4,0,1])` because the piece 7 is upside down to fit.

So far we have seen how to represent a piece and how to represent a connection between two pieces. Now, we need to define how a puzzle solution is represented. For example, the solution of the following shape, that doesn't belong to the list of Izzi2 and is just an example, have the representation:

```
[[2,4,3,2,0,0],[2,6,4,4,0,1],[6,9,2,3,1,1],[4,9,4,4,0,1]]
```



Decision Variables

The variables that will appear in the final solution are the number of the pieces from 1 to 12, the color index from 1 to 4 and the orientation that can be 0 or 1 as you saw previously. The following is an example of a solution with 4 connections that is using any four pieces from the diamonds set.

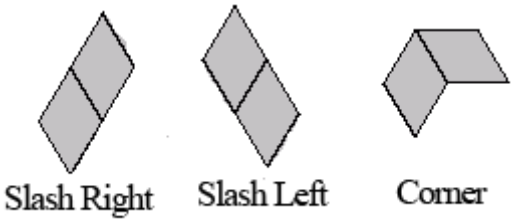
```
(...)

createPieces(Pieces),
Connections=[C1,C2,C3,C4],

C1=[P1,P2,C1I1,C1I2,O1,O2],
C2=[P1,P3,C2I1,C2I2,O1,O3],
C3=[P3,P4,C3I1,C3I2,O3,O4],
C4=[P2,P4,C4I1,C4I2,O2,O4],

(...)

domain([P1,P2,P3,P4],1,12),
domain([C1I1,C1I2,C2I1,C2I2,C3I1,C3I2,C4I1,C4I2],1,4),
domain([O1,O2,O3,O4],0,1),

(...)
```

Constraints

The constraints that were build are for checking: if a connection is valid, if the solution of a shape is valid and if the pieces fit in a certain way. For checking a connection we created the predicate `connect(Pieces,Connection)` that receive the list of pieces and the connection and by comparing the color of the indexes given evaluate if it's true. To check if a complete solution is valid regarding that they are unique, there is a predicate `unique(Connections)`, that receives the list of connections. For example, if the piece 2 is linked to another with the index 4, that index is no more available to another piece and, if there is a connection 1,2, doesn't make sense a connection 2,1 to appear later.

The last constraints and the most important are the constraints to check if the solution respect a certain shape. For example, to describe a shape, we need to say how the pieces fit in each other. We know that some of them fit vertically to the right and to the left, so we need to describe exactly that.

So this conclusion took a hole day starring at the code, solutions and Izzi2 website to find a pattern in the shapes. We found 3 patterns: a slash right connection, a



slash left connection and a corner connection. We can identify any of this connections by the index that two pieces fit together. So if a piece A and piece B fit with 4 and 1, they are a slash left. We named like this because is the best way to describe the pattern that we see and we must allow them to rotate.

```
connectionSlashLeft(Connection):-
        Connection=[ , ,I1,I2,O1,O2],
        (
                (I1#=4, I2#=1, O1#=0,O2#=0);
                (I1#=4, I2#=4, O1#=0,O2#=1);
                (I1#=1, I2#=4, O1#=1,O2#=1);
                (I1#=1, I2#=1, O1#=1,O2#=0)
        ).


connectionSlashRight(Connection):-
        Connection=[ , ,I1,I2,O1,O2],
        (
                (I1#=3, I2#=2, O1#=0,O2#=0);
                (I1#=3, I2#=3, O1#=0,O2#=1);
                (I1#=2, I2#=3, O1#=1,O2#=1);
                (I1#=2, I2#=2,  O1#=1,O2#=0)
        ).

connectionCorner(Connection):-
        Connection=[_,_,I1,I2,O1,O2],
        (
                (I1#=2, I2#=1, O1#=0,O2#=0);
                (I1#=2, I2#=4, O1#=0,O2#=1);
                (I1#=3, I2#=1, O1#=1,O2#=1);
                (I1#=3, I2#=4, O1#=1,O2#=0);
                (I1#=1, I2#=2, O1#=0,O2#=0);
                (I1#=4, I2#=2, O1#=0,O2#=1);
                (I1#=1, I2#=3, O1#=1,O2#=1);
                (I1#=4, I2#=3, O1#=1,O2#=0)
        ).
```

Search Strategy

The labeling uses the list of all variables we said previously and is used like this:

```
Sol=[P1,P2,P3,P4,C1I1,C1I2,C2I1,C2I2,C3I1,C3I2,C4I1,C4I2,O1,O2,O3,O4],
labeling([],Sol).
```

Solution Presentation

This puzzle is very hard to represent in text mode. So, what we do is listing all the existing pieces and giving them a number and returning the connections solution. The user should have the pieces printed, with the same numbers we used in the code and following the connections instructions. To run izzi2 just consult the file izzi2cli.pl and then type "izzi2."

```
| ?- izzi2.
Choose one option:
0 - LIST PEICES
1 - Shape 1
2 - Shape Simple Slash
3 - Shape Simple Corner
4 - shape 4 Corners
5 - shape 10 **WARNING: TAKES A LONG TIME!!**
|: 1

[[1,6,4,1,0,0],[4,6,3,2,0,0],[6,2,3,2,0,0],[6,7,4,4,0,1],[7,5,1,4,1,1],[2,12,3,
2,0,0],[12,8,4,1,0,0],[5,9,2,3,1,1],[8,10,4,1,0,0],[9,10,2,2,1,0],[10,3,3,3,0,1
],[10,11,4,1,0,0]]
```

Results

The time the Prolog spend to return a solution depends in the complexity of the puzzle. If there are a lot of constraints and the shape have corners mixed with slash lefts or slash rights, Sicstus take a lot of hours to compute a solution.

For every different shape we need to implement a predicate that declares all the connection types between each piece. We have built a solution to save time on this using Python to "compile" a shape representation to Prolog. The syntax is this:

```
createCode([[1,3,"SL"],[2,3,"SR"],[3,4,"SR"],[3,5,"SL"],[5,7,"
SL"],[4,6,"SR"],[6,8,"SL"],[7,9,"SR"],[8,10,"SL"],[9,10,"SR"],
[10,11,"SR"],[10,12,"CR"]],1)
```

We just give a list of connections between some pieces and specify the type of connection like Slash Right, Left or Corner. So every time we want to create a shape, we use this to generate code. All the existing shapes are in the file izzi2Puzzles.pl

Conclusion and future work

This project was very challenging and the most difficult that we saw so far. The webpage of Izzi2 have a lot of useful information but is only that. We searched for articles about generic algorithms to implement a puzzle solver but we found nothing good. There is a lack of investigation in the puzzle solving area and we can have another similar problem like building a robot to fill all the tiles in a room, following some constraints or even a machine to build products with some patterns. The information in this area is very poor so we lost days and even weeks trying to find a solution. Since we couldn't use a matrix to represent the diamonds in a board, because of the orientations we achieve a representation that is generic and can be useful to a lot of puzzle problems.

A better environment to help us to develop this solution more efficiently would be an access to a Cluster or Servers to compute the puzzles faster and then confirm that they are right in less time.

References

- Jaap's Puzzle Page. 2010. Izzi2. Visited 25th November 2012. http://www.jaapsch.net/puzzles/izzi2.htm

Attachments

```
%   _ _____________ ___
%  (_)___  /___  (_)__ \
%    / /  / / / / _  ) |
%   | | / /  / / / | |/ /
%   | |/ /__ / /__| |/ /_
%   |_/_____/_____|_|____|
%

:- use_module(library(clpfd)).
:- use_module(library(lists)).
% Pieces database
% Example: for p1([y,r,g,b]):
%   yr
%  yyrr
% yyyrrr
% gggbbb
%  ggbb
%   gb

p1([y,r,g,b]).
p2([y,r,b,g]).
p3([r,b,g,y]).
p4([r,b,y,g]).
p5([b,y,g,r]).
p6([b,y,r,g]).
p7([r,y,b,g]).
p8([r,y,g,b]).
p9([b,r,y,g]).
p10([b,r,g,y]).
p11([y,b,r,g]).
p12([y,b,g,r]).

% Get piece color
getColor(Piece,Index,Color):-
        Piece=[C1,C2,C3,C4],
        (
                Index#=1,
                Color#=C1;

                Index#=2,
                Color#=C2;

                Index#=3,
                Color#=C3;

                Index#=4,
                Color#=C4

        ).

% Create Pieces List
```

```
createPieces(Pieces):-

p1(P1),p2(P2),p3(P3),p4(P4),p5(P5),p6(P6),p7(P7),p8(P8),p9(P9),p
10(P10),p11(P11),p12(P12),
        Pieces=[P1,P2,P3,P4,P5,P6,P7,P8,P9,P10,P11,P12].

% Get Piece from piece list
getPiece(Pieces,Index,Piece):-
        Pieces=[H|_],
        Index#=1,
        Piece=H.

getPiece(Pieces,Index,Piece):-
        Index#>1,
        NewIndex#= Index-1,
        Pieces=[_|T],
        getPiece(T,NewIndex,Piece).

% Connection
connect(Pieces,Connection):-
        Connection=[Piece1Index,Piece2Index,P1ColorIndex,P2Col-
orIndex,_,_],
        Piece1Index#\=Piece2Index,
        getPiece(Pieces,Piece1Index,Piece1),
        getPiece(Pieces,Piece2Index,Piece2),
        getColor(Piece1,P1ColorIndex,Piece1IndexColor),
        getColor(Piece2,P2ColorIndex,Piece2IndexColor),
        Piece1IndexColor#=Piece2IndexColor.

% Extract element from a list
unique([]).
unique(Connections):-
        Connections=[Connection|T],
        delete(Connections, _, WithoutList),
        connectionUnique(Connection,WithoutList),
        unique(T).

% Check if with existing connections can create a connection
connectionUnique(Connection,Connections):-
        Connection=[P1,P2,I1,I2,_,_],
        \+member([_,P2,_,I2,_,_],Connections),
        \+member([P2,_,I2,_,_,_],Connections),
        \+member([P1,P2,_,_,_,_],Connections),
        \+member([P2,P1,_,_,_,_],Connections),
        \+member([_,P1,_,I1,_,_],Connections),
        \+member([P1,_,I1,_,_,_],Connections).

% Connection types
connectionSlashLeft(Connection):-
        Connection=[_,_,I1,I2,O1,O2],
        (
                (I1#=4, I2#=1, O1#=0,O2#=0);
                (I1#=4, I2#=4, O1#=0,O2#=1);
```

```
                (I1#=1, I2#=4, O1#=1,O2#=1);
                (I1#=1, I2#=1, O1#=1,O2#=0)
        ).


connectionSlashRight(Connection):-
        Connection=[_,_,I1,I2,O1,O2],
        (
                (I1#=3, I2#=2, O1#=0,O2#=0);
                (I1#=3, I2#=3, O1#=0,O2#=1);
                (I1#=2, I2#=3, O1#=1,O2#=1);
                (I1#=2, I2#=2,  O1#=1,O2#=0)
        ).

connectionCorner(Connection):-
        Connection=[_,_,I1,I2,O1,O2],
        (
                (I1#=2, I2#=1, O1#=0,O2#=0);
                (I1#=2, I2#=4, O1#=0,O2#=1);
                (I1#=3, I2#=1, O1#=1,O2#=1);
                (I1#=3, I2#=4, O1#=1,O2#=0);

                (I1#=1, I2#=2, O1#=0,O2#=0);
                (I1#=4, I2#=2, O1#=0,O2#=1);
                (I1#=1, I2#=3, O1#=1,O2#=1);
                (I1#=4, I2#=3, O1#=1,O2#=0)
        ).

:- consult(izzi2puzzles).

%write vertical piece
printVPiece(P):-
        printPiece(P).
printPiece(P):-
        getColor(P,1,C1),
        getColor(P,2,C2),
        getColor(P,3,C3),
        getColor(P,4,C4),
        write('   '),write(C1),write(C2),nl,
        write('1
'),write(C1),write(C1),write(C2),write(C2),write(' 2'),nl,
        write('
'),write(C1),write(C1),write(C1),write(C2),write(C2),write(C2),n
l,
        write('
'),write(C3),write(C3),write(C3),write(C4),write(C4),write(C4),n
l,
        write('3
'),write(C3),write(C3),write(C4),write(C4),write(' 4'),nl,
        write('   '),write(C3),write(C4),nl.
```

```
izzi2 :-
        write('Choose one option:'),nl,
        write('0 - LIST PEICES'),nl,
        write('1 - Shape 1'),nl,
        write('2 - Shape Simple Slash'),nl,
        write('3 - Shape Simple Corner'),nl,
        write('4 - shape 4 Corners'),nl,
        write('5 - shape 10 **WARNING: TAKES A LONG
TIME!!**'),nl,
        read(Input),(
        Input = 1,(
                shape1(C),write(C)
                );
        Input = 2,(
                shapeSimpleSlash(C),write(C)
                );
        Input = 3,(
                shapeSimpleCorner(C),write(C)
                );
        Input = 4,(
                shape4Corners(C),write(C)
                );
        Input = 5,(
                shape10(C),write(C)
                );
        Input = 0,(
                write('---P1---'),nl,nl,
                p1(P1),printPiece(P1),nl,
                write('---P2---'),nl,nl,
                p2(P2),printPiece(P2),nl,
                write('---P3---'),nl,nl,
                p3(P3),printPiece(P3),nl,
                write('---P4---'),nl,nl,
                p4(P4),printPiece(P4),nl,
                write('---P5---'),nl,nl,
                p5(P5),printPiece(P5),nl,
                write('---P6---'),nl,nl,
                p6(P6),printPiece(P6),nl,
                write('---P7---'),nl,nl,
                p7(P7),printPiece(P7),nl,
                write('---P8---'),nl,nl,
                p8(P8),printPiece(P8),nl,
                write('---P9---'),nl,nl,
                p9(P9),printPiece(P9),nl,
                write('---P10---'),nl,nl,
                p10(P10),printPiece(P10),nl,
                write('---P11---'),nl,nl,
                p11(P11),printPiece(P11),nl,
                write('---P12---'),nl,nl,
                p12(P12),printPiece(P12),nl
                )
        ).
```

```
:- consult(izzi2).

shape4Corners(Connections):-
        createPieces(Pieces),
        Connections=[C1,C2,C3,C4],

        C1=[P1,P2,C1I1,C1I2,O1,O2],
        C2=[P1,P3,C2I1,C2I2,O1,O3],
        C3=[P2,P4,C3I1,C3I2,O2,O4],
        C4=[P3,P4,C4I1,C4I2,O3,O4],
        connectionCorner(C1),
        connectionSlashLeft(C2),
        connectionSlashRight(C3),
        connectionCorner(C4),

        connect(Pieces,C1),
        connect(Pieces,C2),
        connect(Pieces,C3),
        connect(Pieces,C4),

        domain([P1,P2,P3,P4],1,12),
        domain([C1I1,C1I2,C2I1,C2I2,C3I1,C3I2,C4I1,C4I2],1,4),
        domain([O1,O2,O3,O4],0,1),

        all_different([P1,P2,P3,P4]),
        unique(Connections),

Sol=[P1,P2,P3,P4,C1I1,C1I2,C2I1,C2I2,C3I1,C3I2,C4I1,C4I2,O1,O2,O
3,O4],
        labeling([],Sol).

shape10(Connections):-
        createPieces(Pieces),
        Connections=[C1,C2,C3,C4,C5,C6,C7,C8,C9,C10,C11,C12],

        C1=[P1,P2,C1I1,C1I2,O1,O2],
        C2=[P4,P3,C2I1,C2I2,O4,O3],
        C3=[P6,P5,C3I1,C3I2,O6,O5],
        C4=[P2,P3,C4I1,C4I2,O2,O3],
        C5=[P5,P2,C5I1,C5I2,O5,O2],
        C6=[P3,P8,C6I1,C6I2,O3,O8],
        C7=[P5,P10,C7I1,C7I2,O5,O10],
        C8=[P8,P7,C8I1,C8I2,O8,O7],
        C9=[P8,P9,C9I1,C9I2,O8,O9],
        C10=[P9,P11,C10I1,C10I2,O9,O11],
        C11=[P9,P10,C11I1,C11I2,O9,O10],
        C12=[P10,P12,C12I1,C12I2,O10,O12],

        connectionSlashRight(C1),
        connectionSlashRight(C2),
        connectionSlashRight(C3),
        connectionSlashRight(C10),
        connectionSlashRight(C8),
```

```
        connectionSlashRight(C12),
        connectionCorner(C11),
        connectionCorner(C4),
        connectionCorner(C5),
        connectionCorner(C6),
        connectionCorner(C7),
        connectionCorner(C9),



        connect(Pieces,C1),
        connect(Pieces,C2),
        connect(Pieces,C3),
        connect(Pieces,C4),
        connect(Pieces,C5),
        connect(Pieces,C6),
        connect(Pieces,C7),
        connect(Pieces,C8),
        connect(Pieces,C9),
        connect(Pieces,C10),
        connect(Pieces,C11),
        connect(Pieces,C12),

        domain([P1,P2,P3,P4,P5,P6,P7,P8,P9,P10,P11,P12],1,12),

domain([C1I1,C1I2,C2I1,C2I2,C3I1,C3I2,C4I1,C4I2,C5I1,C5I2,C6I1,C
6I2,C7I1,C7I2,C8I1,C8I2,C9I1,C9I2,C10I1,C10I2,C11I1,C11I2,C12I1,
C12I2],1,4),
        domain([O1,O2,O3,O4,O5,O6,O7,O8,O9,O10,O11,O12],0,1),

        all_different([P1,P2,P3,P4,P5,P6,P7,P8,P9,P10,P11,P12]),
        unique(Connections),

Sol=[P1,P2,P3,P4,P5,P6,P7,P8,P9,P10,P11,P12,C1I1,C1I2,C2I1,C2I2,
C3I1,C3I2,C4I1,C4I2,C5I1,C5I2,C6I1,C6I2,C7I1,C7I2,C8I1,C8I2,C9I1
,C9I2,C10I1,C10I2,C11I1,C11I2,C12I1,C12I2,O1,O2,O3,O4,O5,O6,O7,O
8,O9,O10,O11,O12],
        labeling([],Sol).


shapeSimpleSlash(Connections):-
        createPieces(Pieces),
        Connections=[C1,C2,C3,C4],

        C1=[P1,P2,C1I1,C1I2,O1,O2],
        C2=[P1,P3,C2I1,C2I2,O1,O3],
        C3=[P3,P4,C3I1,C3I2,O3,O4],
        C4=[P2,P4,C4I1,C4I2,O2,O4],

        connectionSlashRight(C1),
        connectionSlashLeft(C2),
        connectionSlashRight(C3),
```

```
        connectionSlashLeft(C4),

        connect(Pieces,C1),
        connect(Pieces,C2),
        connect(Pieces,C3),
        connect(Pieces,C4),

        domain([P1,P2,P3,P4],1,12),
        domain([C1I1,C1I2,C2I1,C2I2,C3I1,C3I2,C4I1,C4I2],1,4),
        domain([O1,O2,O3,O4],0,1),

        all_different([P1,P2,P3,P4]),
        unique(Connections),

Sol=[P1,P2,P3,P4,C1I1,C1I2,C2I1,C2I2,C3I1,C3I2,C4I1,C4I2,O1,O2,O
3,O4],
        labeling([],Sol).



shapeSimpleCorner(Connections):-
        createPieces(Pieces),
        Connections=[C1,C2],

        C1=[P1,P2,C1I1,C1I2,O1,O2],
        C2=[P2,P3,C2I1,C2I2,O2,O3],

        connectionCorner(C1),
        connectionSlashLeft(C2),



        connect(Pieces,C1),
        connect(Pieces,C2),

        domain([P1,P2,P3],1,12),
        domain([C1I1,C1I2,C2I1,C2I2],1,4),
        domain([O1,O2,O3],0,1),

        all_different([P1,P2,P3]),
        unique(Connections),
        Sol=[P1,P2,P3,C1I1,C1I2,C2I1,C2I2,O1,O2,O3],
        labeling([],Sol).

shape1(Connections):-
        createPieces(Pieces),
        Connections=[C1,C2,C3,C4,C5,C6,C7,C8,C9,C10,C11,C12],

        C1=[P1,P3,C1I1,C1I2,O1,O3],
        C2=[P2,P3,C2I1,C2I2,O2,O3],
        C3=[P3,P4,C3I1,C3I2,O3,O4],
        C4=[P3,P5,C4I1,C4I2,O3,O5],
```

```
        C5=[P5,P7,C5I1,C5I2,O5,O7],
        C6=[P4,P6,C6I1,C6I2,O4,O6],
        C7=[P6,P8,C7I1,C7I2,O6,O8],
        C8=[P7,P9,C8I1,C8I2,O7,O9],
        C9=[P8,P10,C9I1,C9I2,O8,O10],
        C10=[P9,P10,C10I1,C10I2,O9,O10],
        C11=[P10,P11,C11I1,C11I2,O10,O11],
        C12=[P10,P12,C12I1,C12I2,O10,O12],

        connectionSlashLeft(C1),
        connectionSlashRight(C2),
        connectionSlashRight(C3),
        connectionSlashLeft(C4),
        connectionSlashLeft(C5),
        connectionSlashRight(C6),
        connectionSlashLeft(C7),
        connectionSlashRight(C8),
        connectionSlashLeft(C9),
        connectionSlashRight(C10),
        connectionSlashRight(C11),
        connectionSlashLeft(C12),

        connect(Pieces,C1),
        connect(Pieces,C2),
        connect(Pieces,C3),
        connect(Pieces,C4),
        connect(Pieces,C5),
        connect(Pieces,C6),
        connect(Pieces,C7),
        connect(Pieces,C8),
        connect(Pieces,C9),
        connect(Pieces,C10),
        connect(Pieces,C11),
        connect(Pieces,C12),

        domain([P1,P2,P3,P4,P5,P6,P7,P8,P9,P10,P11,P12],1,12),

domain([C1I1,C1I2,C2I1,C2I2,C3I1,C3I2,C4I1,C4I2,C5I1,C5I2,C6I1,C
6I2,C7I1,C7I2,C8I1,C8I2,C9I1,C9I2,C10I1,C10I2,C11I1,C11I2,C12I1,
C12I2],1,4),
        domain([O1,O2,O3,O4,O5,O6,O7,O8,O9,O10,O11,O12],0,1),

        all_different([P1,P2,P3,P4,P5,P6,P7,P8,P9,P10,P11,P12]),
        unique(Connections),

Sol=[P1,P2,P3,P4,P5,P6,P7,P8,P9,P10,P11,P12,C1I1,C1I2,C2I1,C2I2,
C3I1,C3I2,C4I1,C4I2,C5I1,C5I2,C6I1,C6I2,C7I1,C7I2,C8I1,C8I2,C9I1
,C9I2,C10I1,C10I2,C11I1,C11I2,C12I1,C12I2,O1,O2,O3,O4,O5,O6,O7,O
8,O9,O10,O11,O12],
        labeling([],Sol).
```

```
def createCode(connections,shape,filename):
    fo = open(filename+".pl", "a")
    fo.write("shape"+str(shape)+"(Connections):-\n")
    fo.write("\tcreatePieces(Pieces),\n")

    # Write List of connections
    fo.write("\tConnections=[")
    for i in range(1,len(connections)+1):
        fo.write("C"+str(i))
        if(i!=(len(connections))):
            fo.write(",")

    fo.write("],\n\n")

    #Write connections
    for i in range(len(connections)):
        p1=connections[i][0]
        p2=connections[i][1]
        istr=str(i+1)

fo.write("\tC"+istr+"=[P"+str(p1)+",P"+str(p2)+",C"+istr+"I1,C"+
istr+"I2,O"+str(p1)+",O"+str(p2)+"],\n")

    # Write topology
    for i in range(len(connections)):
        top=connections[i][2]
        istr=str(i+1)
        if(top=="SR"):
            fo.write("\tconnectionSlashRight(C"+istr+"),\n")
        if(top=="SL"):
            fo.write("\tconnectionSlashLeft(C"+istr+"),\n")
        if(top=="CR"):
            fo.write("\tconnectionCorner(C"+istr+"),\n")

    #Write connnect
    fo.write("\n")
    for i in range(1,len(connections)+1):
        fo.write("\tconnect(Pieces,C"+str(i)+"),\n")

    #Write pieces
    fo.write("\n\tdo-
main([P1,P2,P3,P4,P5,P6,P7,P8,P9,P10,P11,P12],1,12),\n")
    #  Write Indexes
    fo.write("\tdomain([")
    for i in range(1,len(connections)+1):
        fo.write("C"+str(i)+"I1,C"+str(i)+"I2")
        if(i!=(len(connections))):
            fo.write(",")
    fo.write("],1,4),\n")


fo.write("\tdomain([O1,O2,O3,O4,O5,O6,O7,O8,O9,O10,O11,O12],0,1)
,\n")
```

```
    fo.write("\n\tall_differ-
ent([P1,P2,P3,P4,P5,P6,P7,P8,P9,P10,P11,P12]),\n")
    fo.write("\tunique(Connections),\n")
    # Write sol to labeling
    fo.write("\tSol=[P1,P2,P3,P4,P5,P6,P7,P8,P9,P10,P11,P12,")
    for i in range(1,len(connections)+1):
        fo.write("C"+str(i)+"I1,C"+str(i)+"I2")
        if(i!=(len(connections))):
            fo.write(",")
    fo.write(",O1,O2,O3,O4,O5,O6,O7,O8,O9,O10,O11,O12],\n")

    fo.write("\tlabeling([],Sol).\n")
    fo.close()

createCode([
        [1,2,"CR"],
        [1,3,"CR"],
        [2,3,"CR"],
        [2,4,"SR"],
        [3,5,"SL"],
        [4,5,"CR"],
        [4,6,"CR"],
        [5,7,"CR"],
        [6,8,"CR"],
        [7,9,"CR"],
        [8,9,"CR"],
        [8,10,"SL"],
        [9,11,"SR"],
        [10,11,"CR"],
        [10,12,"CR"],
        [11,12,"CR"]
        ],3,"izzi2Puzzles")

createCode([
        [1,2,"SR"],
        [4,3,"SR"],
        [6,5,"SR"],
        [2,3,"CR"],
        [5,2,"CR"],
        [3,8,"CR"],
        [5,10,"CR"],
        [8,7,"SR"],
        [8,9,"CR"],
        [9,11,"SR"],
        [9,10,"CR"],
        [10,12,"SR"]
        ],10,"izzi2Puzzles")


createCode([
        [1,2,"CR"],
        [1,3,"SL"],
        [2,4,"SR"],
```

```
[3,4,"CR"]
],"4Corners","izzi2Puzzles")
```