# Programms that solves some puzzles

Matteo Nestola [matteo.nestola@studio.unibo.it]

May 10, 2023

**Abstract**

The aim of this project is to find a solution to several questions drawn from the 2021 autumn games. The problems were solved using the paradigm of constraint programming (CP) where relations between variables can be declared in the form of constraints. The language used to implement CP was MiniZinc (a free and open-source constraint modelling language).

## 1 Introduction

The problems to solve are :

1. **Category CE**:
   (a) *RED AND BLACK (1)*
   (b) *THE SMALLEST (4)*
2. **Category C1**:
   (a) *A MAGIC TRIANGLE (5)*
   (b) *HOW MANY 9s! (7)*
3. **Category C2**:
   (a) *MATTEO's AGE (8)*
   (b) *NOW THERE ARE 18! (10)*
4. **Category L1**:
   (a) *THE NINTH (11)*
   (b) *CARLA's BOXES (13)*
5. **Category L2**:
   (a) *LUCA's ALARM CLOCK (16)*
   (b) *A TRIANGLE IN THE WATCH (18)*

# 2 Problems implementation in Minizinc

## 2.1 RED AND BLACK (1)

Text: *In the 52-card deck there are 26 red and 26 black. The deck is separated into two decks: the first of 25 cards, the second of 27.*
Problem: *If the first deck contains 12 red cards, how many black cards are there in the second deck?*

### 2.1.1 Variables

To do this, an array of 52 variables (each variable represents a card) is inserted into the Minizinc programme. Each card can be either only red or only black, which is why the domain of the cards is 0,1 where:

- 0 represents a red card;
- 1 represents a black card.

### 2.1.2 Constraints

To find a solution are written 4 different constraints:

- the maximum number of red cards in the deck is 26;
- the maximum number of black cards in the deck is 26;
- in the first sub-deck of 25 cards there are 12 red;
- in the second sub-deck of 27 there are 26-12 = 14 red.

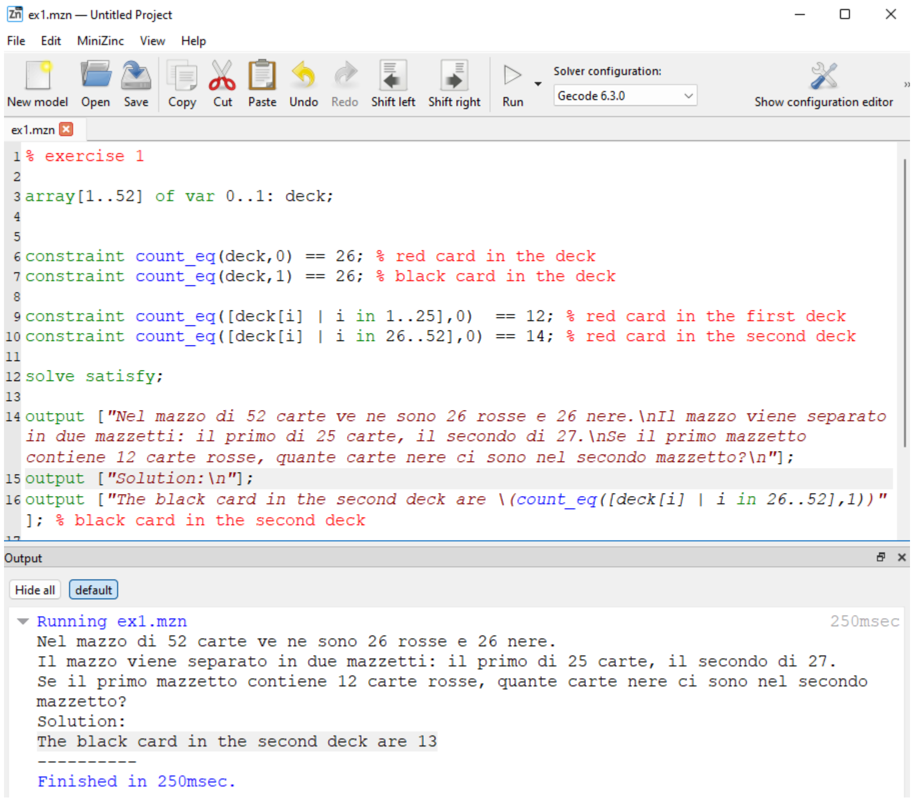
The fourth constraint is a consequence of the third, because if we have 26 red cards of which 12 are in the first sub-deck, the rest will be in the second sub-deck.

### 2.1.3 Solution

In this case it is a satisfaction problem: we wish to find a value for the decision variables that satisfies the constraints but we do not care which one. For that reason in our model we use "solve satisfy". The solution identified by the program is:

$$\textit{The black cards in the second deck are } 13. \tag{1}$$

### 2.1.4 Code



Solver configuration: Gecode 6.3.0

```
% exercise 1

array[1..52] of var 0..1: deck;


constraint count_eq(deck,0) == 26; % red card in the deck
constraint count_eq(deck,1) == 26; % black card in the deck

constraint count_eq([deck[i] | i in 1..25],0)  == 12; % red card in the first deck
constraint count_eq([deck[i] | i in 26..52],0) == 14; % red card in the second deck

solve satisfy;

output ["Nel mazzo di 52 carte ve ne sono 26 rosse e 26 nere.\nIl mazzo viene separato
in due mazzetti: il primo di 25 carte, il secondo di 27.\nSe il primo mazzetto
contiene 12 carte rosse, quante carte nere ci sono nel secondo mazzetto?\n"];
output ["Solution:\n"];
output ["The black card in the second deck are \(count_eq([deck[i] | i in 26..52],1))"
]; % black card in the second deck
```

```
Running ex1.mzn                                                          250msec
Nel mazzo di 52 carte ve ne sono 26 rosse e 26 nere.
Il mazzo viene separato in due mazzetti: il primo di 25 carte, il secondo di 27.
Se il primo mazzetto contiene 12 carte rosse, quante carte nere ci sono nel secondo
mazzetto?
Solution:
The black card in the second deck are 13
----------
Finished in 250msec.
```

Figure 1: Red and black card minizinc implementation.

## 2.2 THE SMALLEST (4)

What is the smallest number (positive integer) of four digits, all even and all different from each other, that is a multiple of 11? (a number cannot begin with the digit 0).

### 2.2.1 Variables

All the following variables domain ranges from 0 to 9:

- c1 represents the first digit;
- c2 represents the second digit;
- c3 represents the third digit;
- c4 represents the fourth digit.

### 2.2.2 Constraints

To find a solution are written 7 different constraints:

- c1 must be different from 0;
- all digits must be different from each other;
- all digits must be divisible by 2;
- the number consisting of the digits c1 - c2 - c3 - c4 must be divisible by 11.

### 2.2.3 Solution

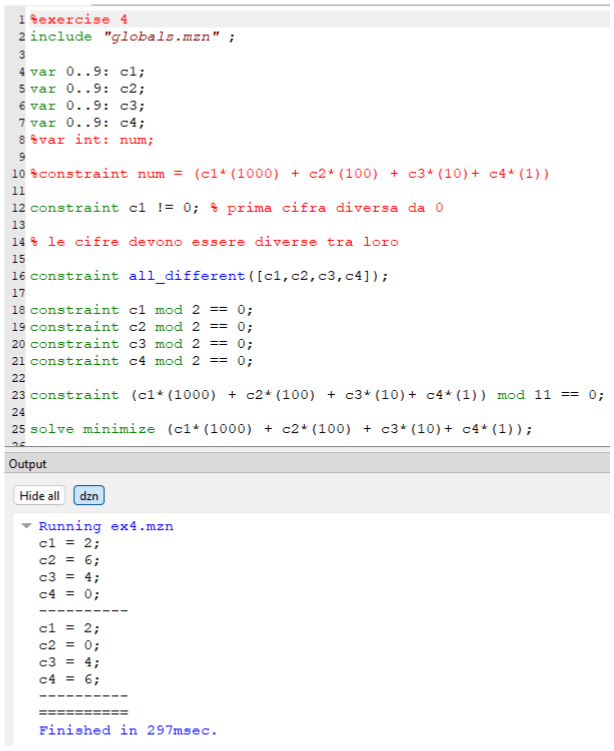
In this case to find a solution it is necessary to minimise

$$c1 * 1000 + c2 * 100 + c3 * 10 + c4$$

. The solution identified by the program is:

$$2046 \tag{2}$$

### 2.2.4 Code



```
%exercise 4
include "globals.mzn" ;

var 0..9: c1;
var 0..9: c2;
var 0..9: c3;
var 0..9: c4;
%var int: num;

%constraint num = (c1*(1000) + c2*(100) + c3*(10)+ c4*(1))

constraint c1 != 0; % prima cifra diversa da 0

% le cifre devono essere diverse tra loro

constraint all_different([c1,c2,c3,c4]);

constraint c1 mod 2 == 0;
constraint c2 mod 2 == 0;
constraint c3 mod 2 == 0;
constraint c4 mod 2 == 0;

constraint (c1*(1000) + c2*(100) + c3*(10)+ c4*(1)) mod 11 == 0;

solve minimize (c1*(1000) + c2*(100) + c3*(10)+ c4*(1));
```

```
Running ex4.mzn
  c1 = 2;
  c2 = 6;
  c3 = 4;
  c4 = 0;
  ----------
  c1 = 2;
  c2 = 0;
  c3 = 4;
  c4 = 6;
  ----------
  ==========
  Finished in 297msec.
```

Figure 2: The smallest

## 2.3 A MAGIC TRIANGLE (5)

The boxes of the triangle in the figure contain all the whole numbers from 1 to 9. Some of the numbers have already been written down. Debora has completed the figure, writing the numbers in each box in such a way that the sum of the numbers on the same side of the triangle always equals 20.

### 2.3.1 Variables

All the following variables domain is described in the variable seq that contains tha values 2,3,6 and 9. The variables are:

- x represents the first number;
- y represents the second number;
- z represents the third number;
- h represents the fourth number.

### 2.3.2 Constraints

To find a solution are written 4 different constraints:

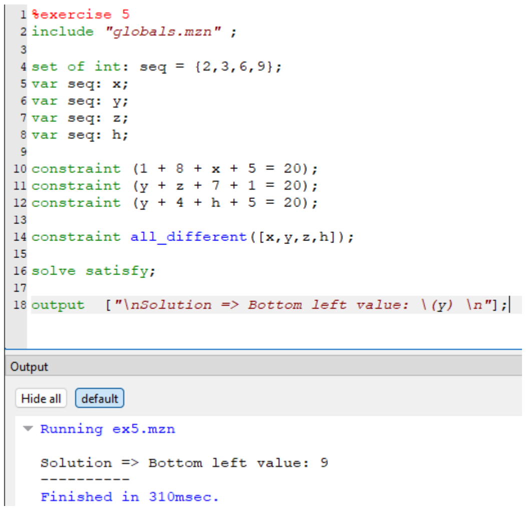
- the first three constraints dictate that each side of the triangle must have as its sum 20;
- the fourth constraint defines that each value of the seq domain can be associated with at most one variable .

### 2.3.3 Solution

This is a satisfiability problem, to find a solution need to be satisfy all the constraints. The solution identified by the program is:

$$Solution => Bottom\ left\ value : 9 \tag{3}$$

### 2.3.4 Code



```
%exercise 5
include "globals.mzn" ;

set of int: seq = {2,3,6,9};
var seq: x;
var seq: y;
var seq: z;
var seq: h;

constraint (1 + 8 + x + 5 = 20);
constraint (y + z + 7 + 1 = 20);
constraint (y + 4 + h + 5 = 20);

constraint all_different([x,y,z,h]);

solve satisfy;

output  ["\nSolution => Bottom left value: \(y) \n"];
```

```
Running ex5.mzn

Solution => Bottom left value: 9
----------
Finished in 310msec.
```

Figure 3: A magic triangle

## 2.4 HOW MANY 9s! (7)

Desiderio considers the first eleven multiples of a two-digit whole number: the number itself, the number multiplied by 2, the number multiplied by 3, ..., up to the number multiplied by 11. At this point he realises that all these eleven multiples contain at least one digit 9. What was the starting number?

### 2.4.1 Variables

- c1: first digit (domain = [0..9]);
- c2: second digit (domain = [0..9]);
- n: represents the starting number to be found.

### 2.4.2 Constraints

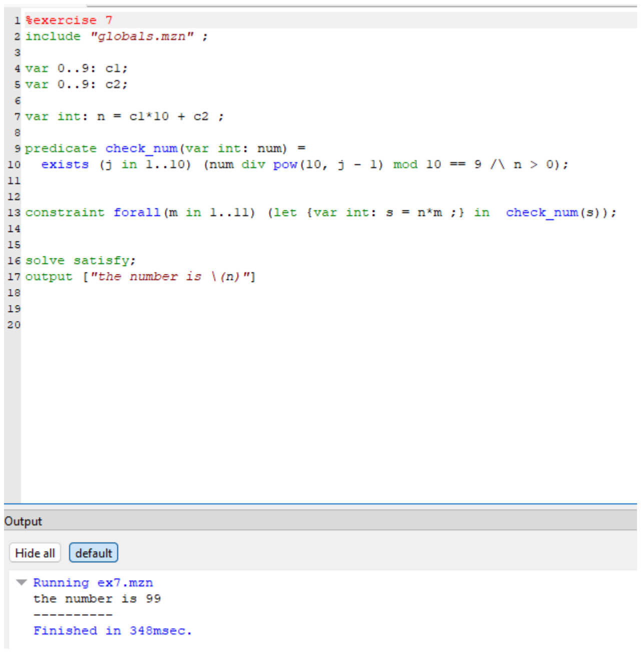
To find a solution is written a single constraint that use a defined predicate to check if all these eleven multiples contain at least one digit 9:

### 2.4.3 Solution

This is a problem of satisfiability and the result is:

$$\textit{the number is } 99 \tag{4}$$

### 2.4.4 Code



```
%exercise 7
include "globals.mzn" ;

var 0..9: c1;
var 0..9: c2;

var int: n = c1*10 + c2 ;

predicate check_num(var int: num) =
  exists (j in 1..10) (num div pow(10, j - 1) mod 10 == 9 /\ n > 0);


constraint forall(m in 1..11) (let {var int: s = n*m ;} in  check_num(s));


solve satisfy;
output ["the number is \(n)"]
```

```
Running ex7.mzn
the number is 99
----------
Finished in 348msec.
```

Figure 4: How many 9s!

## 2.5 MATTEO's AGE (8)

Text: *Matteo was born on 1 January 2000. In 2014 he turned 14 years old and the sum of the digits of that year (2+0+1+4) is equal to 7, half his age.*
Problem: *In which year will the sum of the digits be one fifth of Matthew's age?*

### 2.5.1 Variables

To manage this problem are been defined three different variables :

- y: represent the second digit of the year (domain: $y \in [0, 1]$);
- z: represent the third digit of the year (domain: $z \in [0, 9]$);
- w: represent the fourth digit of the year (domain: $w \in [0, 9]$);

As a consequence of that definition of variables we can rewrite the goal as:

$$(2 + y + z + w) = 1/5 * (z * 10 + w)$$

### 2.5.2 Constraints

- We are considering that Matthew will be older than his current age (14);
- the sum between all digits of the year must be equal to 1/5 of Matthew's age in that year;
- we are assuming that Matteo can be at most 100 years.

### 2.5.3 Solution

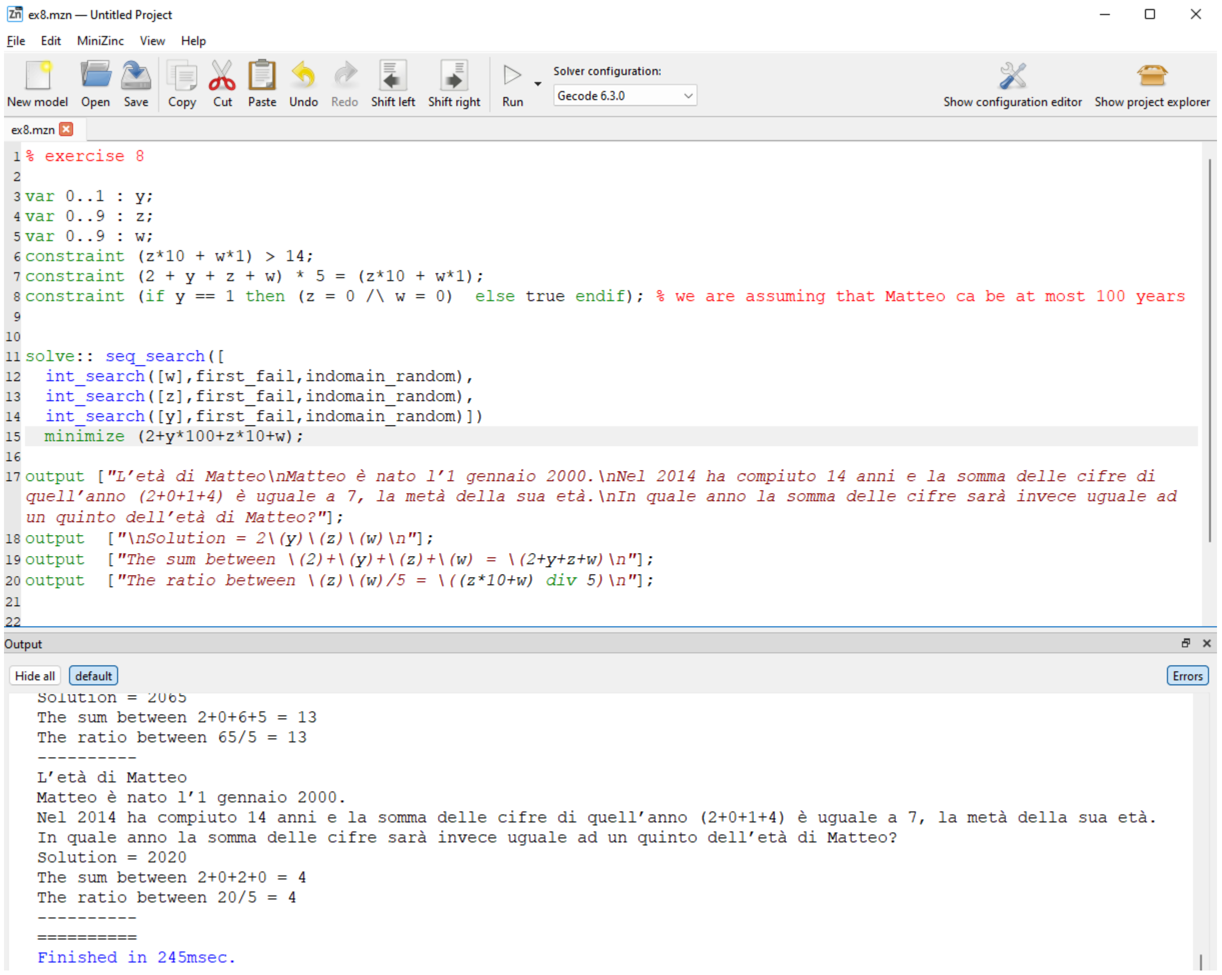
Search_strategy was chosen as the solution search strategy for that problem. The sequential search constructor first undertakes the search given by the first annotation in its list, when all variables in this annotation are fixed it undertakes the second search annotation, etc. until all search annotations are complete.
In that specific case there are three annotation (one for each variable) in which we assign the variable its smallest domain value (*indomain_random*) and we choose the variable with the smallest domain size ($first\_fail$). The goal of that search strategy is to minimize the year $(2 + y * 100 + z * 10 + w)$.

The solution found is 20:

$$2020 => 2 + 0 + 2 + 0 = 4 \; and \; 20/5 = 4$$

As we can see from the image below this search strategy identifies two possible solution 2065 and 2020 but we set to minimize the year and for that reason the best solution found is 2020.

### 2.5.4 Code


ex8.mzn — Untitled Project
File Edit MiniZinc View Help
New model Open Save Copy Cut Paste Undo Redo Shift left Shift right Run
Show configuration editor Show project explorer
ex8.mzn
Output
Hide all default Errors

Solver configuration: Gecode 6.3.0

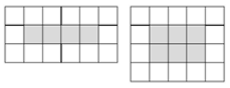
```
% exercise 8

var 0..1 : y;
var 0..9 : z;
var 0..9 : w;
constraint (z*10 + w*1) > 14;
constraint (2 + y + z + w) * 5 = (z*10 + w*1);
constraint (if y == 1 then (z = 0 /\ w = 0)  else true endif); % we are assuming that Matteo ca be at most 100 years


solve:: seq_search([
  int_search([w],first_fail,indomain_random),
  int_search([z],first_fail,indomain_random),
  int_search([y],first_fail,indomain_random)])
  minimize (2+y*100+z*10+w);

output ["L’età di Matteo\nMatteo è nato l’1 gennaio 2000.\nNel 2014 ha compiuto 14 anni e la somma delle cifre di
quell’anno (2+0+1+4) è uguale a 7, la metà della sua età.\nIn quale anno la somma delle cifre sarà invece uguale ad
un quinto dell’età di Matteo?"];
output  ["\nSolution = 2\(y)\(z)\(w)\n"];
output  ["The sum between \(2)+\(y)+\(z)+\(w)  = \(2+y+z+w)\n"];
output  ["The ratio between \(z)\(w)/5 = \((z*10+w) div 5)\n"];
```

```
Solution = 2065
The sum between 2+0+6+5 = 13
The ratio between 65/5 = 13
----------
L’età di Matteo
Matteo è nato l’1 gennaio 2000.
Nel 2014 ha compiuto 14 anni e la somma delle cifre di quell’anno (2+0+1+4) è uguale a 7, la metà della sua età.
In quale anno la somma delle cifre sarà invece uguale ad un quinto dell’età di Matteo?
Solution = 2020
The sum between 2+0+2+0 = 4
The ratio between 20/5 = 4
----------
==========
Finished in 245msec.
```

Figure 5: Matthew’s age minizinc implementation.

## 2.6 NOW THERE ARE 18! (10)

If you have 14 white squares around the perimeter of a rectangle, you can ”encircle” 4 or 6 grey squares (as shown in the figure). Liliana has no less than 18 white squares to place around the perimeter of an appropriately sized rectangle.



How many grey squares can Liliana ’encircle’ at most?

### 2.6.1 Variables

- side1: base of the outside perimeter;
- side2: height of the outside perimeter;
- outside_area: area of the outside rectangle;
- inside1: base of the inside rectangle;
- inside2: height of the inside rectangle;
- inside_area: area of the inside rectangle;

### 2.6.2 Constraints

A number of constraints have been defined:

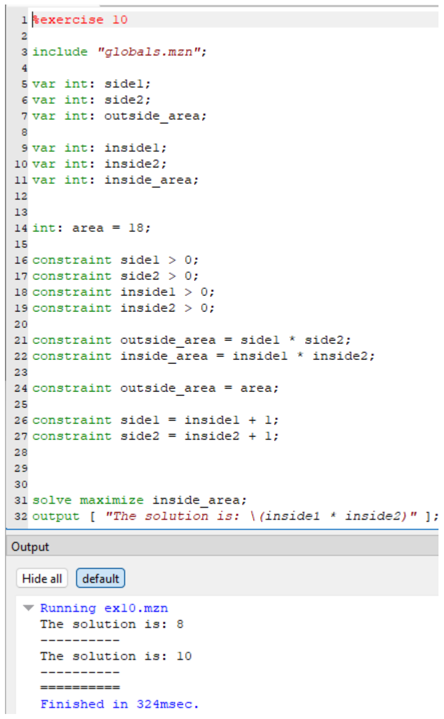
- four constraints to define that the dimensions of rectangles must be greater than 0;
- two constraints to define the areas of the two rectangles ;
- a constraint defining the value of the area of the outer rectangle;
- two constraints defining the relationships between the two rectangles, since they are one inside the other the outer one will have the same dimensions as the inner one increased by 1.

### 2.6.3 Solution

To find a solution to this problem, one must maximise `inside_area` variable:

$$\textit{The solution is}: \ 10 \tag{5}$$

### 2.6.4 Code



```
%exercise 10

include "globals.mzn";

var int: side1;
var int: side2;
var int: outside_area;

var int: inside1;
var int: inside2;
var int: inside_area;


int: area = 18;

constraint side1 > 0;
constraint side2 > 0;
constraint inside1 > 0;
constraint inside2 > 0;

constraint outside_area = side1 * side2;
constraint inside_area = inside1 * inside2;

constraint outside_area = area;

constraint side1 = inside1 + 1;
constraint side2 = inside2 + 1;



solve maximize inside_area;
output [ "The solution is: \(inside1 * inside2)" ];
```

```
Running ex10.mzn
The solution is: 8
----------
The solution is: 10
----------
==========
Finished in 324msec.
```

Figure 6: Now there are 18!

## 2.7 The ninth (11)

Deleting the digit 0 from the number 405 yields 45, which is its ninth (and still divisible by 9). What is the smallest (positive integer) four-digit number such that deleting a 0 yields its ninth?

### 2.7.1 Variables

- c1: the first digit;
- c2: the second digit;
- c3: the third digit;
- c4: the fourth digit;
- n: the number that represents the goal.

### 2.7.2 Constraints

Three constraints have been defined:

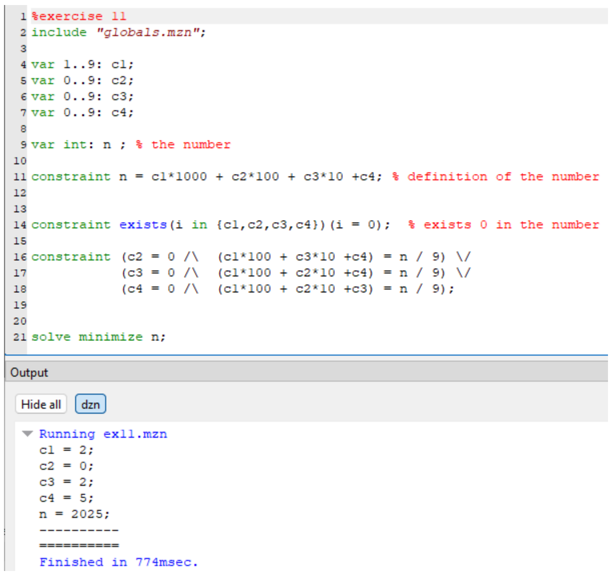
- the first constraint defines `n` as a number composed by the four digits;
- the second constraint check if exist a 0 in the number ;
- the last constraint is the most important one: check if removing the digit 0 from the number results in its ninth.

### 2.7.3 Solution

To find a solution to this problem, one must minimize `n`.

$$\begin{aligned} The\ solution\ is:& \\ c1 = 2;& \\ c2 = 0;& \\ c3 = 2;& \\ c4 = 5;& \\ n = 2025;& \end{aligned}$$

### 2.7.4 Code



```
%exercise 11
include "globals.mzn";

var 1..9: c1;
var 0..9: c2;
var 0..9: c3;
var 0..9: c4;

var int: n ; % the number

constraint n = c1*1000 + c2*100 + c3*10 +c4; % definition of the number


constraint exists(i in {c1,c2,c3,c4})(i = 0);  % exists 0 in the number

constraint (c2 = 0 /\  (c1*100 + c3*10 +c4) = n / 9) \/
           (c3 = 0 /\  (c1*100 + c2*10 +c4) = n / 9) \/
           (c4 = 0 /\  (c1*100 + c2*10 +c3) = n / 9);


solve minimize n;
```

```
Running ex11.mzn
c1 = 2;
c2 = 0;
c3 = 2;
c4 = 5;
n = 2025;
----------
==========
Finished in 774msec.
```

Figure 7: The ninth

## 2.8 CARLA's BOXES (13)

Carla has the two boxes in the picture at her disposal (they are cubes: the first has a side of 10 cm, the second 20 cm). She fills the small one with water several times, to the brim, and then pours it into the second box, without losing a single drop.
How many decantings can Carla do, at most, to fill the second box?

### 2.8.1 Variables

There is only one variable `travasi` which represents the number of racking to be done.

### 2.8.2 Constraints

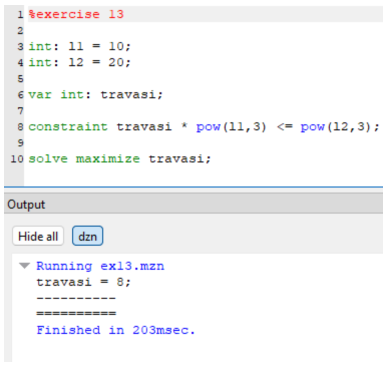
A single constraint is defined:
The number of decantations depends on the volume of the small box and the large box.

### 2.8.3 Solution

To find a solution to this problem, one must maximise `travasi` variable:

$$The\ solution\ is:\ travasi\ =\ 8; \tag{6}$$

### 2.8.4 Code



```
%exercise 13

int: l1 = 10;
int: l2 = 20;

var int: travasi;

constraint travasi * pow(l1,3) <= pow(l2,3);

solve maximize travasi;
```

```
Running ex13.mzn
travasi = 8;
----------
==========
Finished in 203msec.
```

Figure 8: Carla's boxes

## 2.9 LUCA's ALARM CLOCK (16)

Luca has an alarm clock that marks all hours, minutes and seconds from 00:00:00 until 23:59:59. How many times in 24 hours does the alarm clock simultaneously indicate a 0, a 1, a 2, a 3, a 4 and a 5?

### 2.9.1 Variables

`clock` represents the digits on a six-digit digital clock, where each digit can take a value between 0 and 9.

### 2.9.2 Constraints

To find a solution several constraints have been defined:

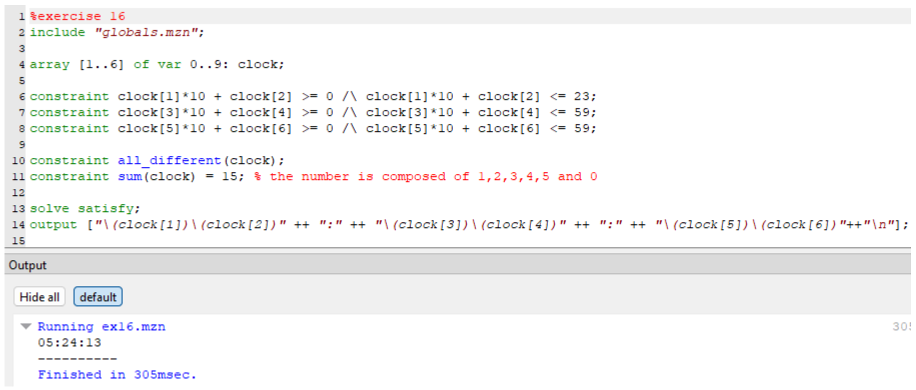
- the first constraint ensures that the first two digits of the clock represent a valid hour value between 0 and 23 inclusive.;
- the second constraint ensures that the third and fourth digits of the clock represent a valid minute value between 0 and 59 inclusive;
- the third constraint ensures that the fifth and sixth digits of the clock represent a valid second value between 0 and 59 inclusive ;
- the fourth constraint ensures that all six elements of the `clock` array are distinct from each other. This means that no digit appears more than once on the clock;
- the fifth constraint ensures that the sum of all six elements of the clock array is equal to 15. Since the digits 1, 2, 3, 4, 5, and 0 are used to form the clock, this constraint ensures that the clock displays a time that includes each of these digits exactly once.

### 2.9.3 Solution

To solve this problem, it is sufficient to find a solution that satisfies all the constraints:

$$The\ solution\ is:\ 05:24:13 \tag{7}$$

### 2.9.4 Code



```
%exercise 16
include "globals.mzn";

array [1..6] of var 0..9: clock;

constraint clock[1]*10 + clock[2] >= 0 /\ clock[1]*10 + clock[2] <= 23;
constraint clock[3]*10 + clock[4] >= 0 /\ clock[3]*10 + clock[4] <= 59;
constraint clock[5]*10 + clock[6] >= 0 /\ clock[5]*10 + clock[6] <= 59;

constraint all_different(clock);
constraint sum(clock) = 15; % the number is composed of 1,2,3,4,5 and 0

solve satisfy;
output ["\(clock[1])\(clock[2])" ++ ":" ++ "\(clock[3])\(clock[4])" ++ ":" ++ "\(clock[5])\(clock[6])"++"\n"];
```

```
Running ex16.mzn
05:24:13
----------
Finished in 305msec.
```

Figure 9: Luca's alarm clock

## 2.10 A TRIANGLE IN THE WATCH (18)

A clock has three hands mounted on the same axis, for the hours, minutes and seconds, all of the same length. Starting at 00:00, with the three hands overlapping, how many times in a 12-hour period do the three ends of the hands lie at the vertices of an equilateral triangle?

### 2.10.1 Variables

- `h`: an integer variable between 0 and 11 (inclusive) representing the hour hand.
- `m`: an integer variable between 0 and 59 (inclusive) representing the minute hand.
- `s`: an integer variable between 0 and 59 (inclusive) representing the second hand.
- `h_degree`: an integer variable between 0 and 360 (inclusive) representing the degree position of the hour hand.
- `m_degree`: an integer variable between 0 and 360 (inclusive) representing the degree position of the minute hand.
- `s_degree`: an integer variable between 0 and 360 (inclusive) representing the degree position of the second hand.

### 2.10.2 Constraints

A number of constraints have been defined:

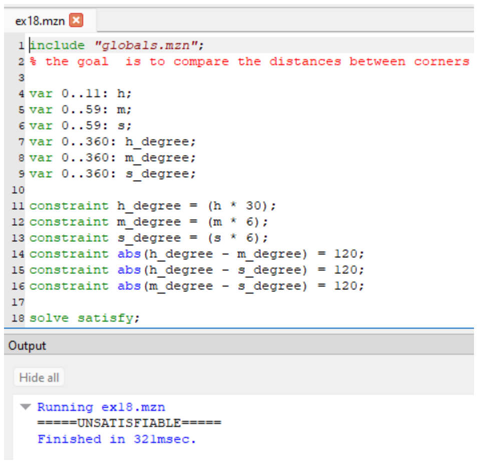
- `h_degree = (h * 30)` : Converts the hour value to its corresponding degree position (there are 12 hours in a clock, and 360 degrees in a circle, so each hour hand moves by 30 degrees).
- `m_degree = (m * 6)` : Converts the minute value to its corresponding degree position (there are 60 minutes in an hour, and 360 degrees in a circle, so each minute hand moves by 6 degrees).
- `s_degree = (s * 6)` : Converts the second value to its corresponding degree position (there are 60 seconds in a minute, and 360 degrees in a circle, so each second hand moves by 6 degrees).
- `abs(h_degree - m_degree) = 120` : Ensures that the difference in degree positions between the hour and minute hand is equal to 120 degrees (the angle of an equilateral triangle).
- `abs(h_degree - s_degree) = 120` : Ensures that the difference in degree positions between the hour and second hand is equal to 120 degrees (the angle of an equilateral triangle).
- `abs(m_degree - s_degree) = 120` : Ensures that the difference in degree positions between the minute and second hand is equal to 120 degrees (the angle of an equilateral triangle).

### 2.10.3 Solution

This is a satisfiability problem, for this specific problem there is no solution so the number of times is 0:

$$The\ solution\ is : ===== UNSATISFIABLE ===== \tag{8}$$

### 2.10.4 Code



```
include "globals.mzn";
% the goal  is to compare the distances between corners

var 0..11: h;
var 0..59: m;
var 0..59: s;
var 0..360: h_degree;
var 0..360: m_degree;
var 0..360: s_degree;

constraint h_degree = (h * 30);
constraint m_degree = (m * 6);
constraint s_degree = (s * 6);
constraint abs(h_degree - m_degree) = 120;
constraint abs(h_degree - s_degree) = 120;
constraint abs(m_degree - s_degree) = 120;

solve satisfy;
```

```
Running ex18.mzn
=====UNSATISFIABLE=====
Finished in 321msec.
```

Figure 10: A triangle in the watch

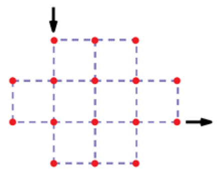
## 3 External resources

Below is the link to the GitHub repository containing extra exercises, and all the minizinc files of each exercise are also preset in the repository: Github

# 4 EXTRA exercises

## 4.1 THE MAZE (2)

In the following drawing, you see a diagram of a maze. Maximus enters the maze at the top left (where the arrow is) and exits to the right (at the other arrow), following the dotted lines and never passing the same red dot more than once.



Knowing that the distance between two pellets is 10 metres, what is the maximum distance Massimo can travel?

### 4.1.1 Variables

The code starts by defining the number of nodes and edges in the graph, the starting node, and the ending node:

```
int: n = 16; % number of nodes
int: num_edges = 42; % number of links
int: start = 1; % starting node
int: target = 13; % ending node
```

### 4.1.2 Constraints

Next, an array of variables representing the nodes in the path is declared, with the domain of each variable ranging from 0 to the number of nodes. The path length is computed as the sum of all nodes that are greater than 0:

```
array[1..n] of var 0..n: x;
var int: path_length = sum(i in 1..n) (x[i] > 0);
```

Two predicates are defined: `last_not_0` and `all_paths`. The `last_not_0` predicate checks if the last element in the path is greater than 0 and if all the elements after that are 0:

```
predicate last_not_0(array[int] of var int: x, var int: y) =
exists(i in 1..length(x)) (
x[i] = y /\
forall(j in i+1..n) (x[j] = 0) /\ % all elements after y are 0
forall(j in 1..i) (x[j] > 0) % all elements before y are > 0
);
```

The `all_paths` predicate generates all possible paths in the maze by checking if there is a link between each consecutive pair of nodes:

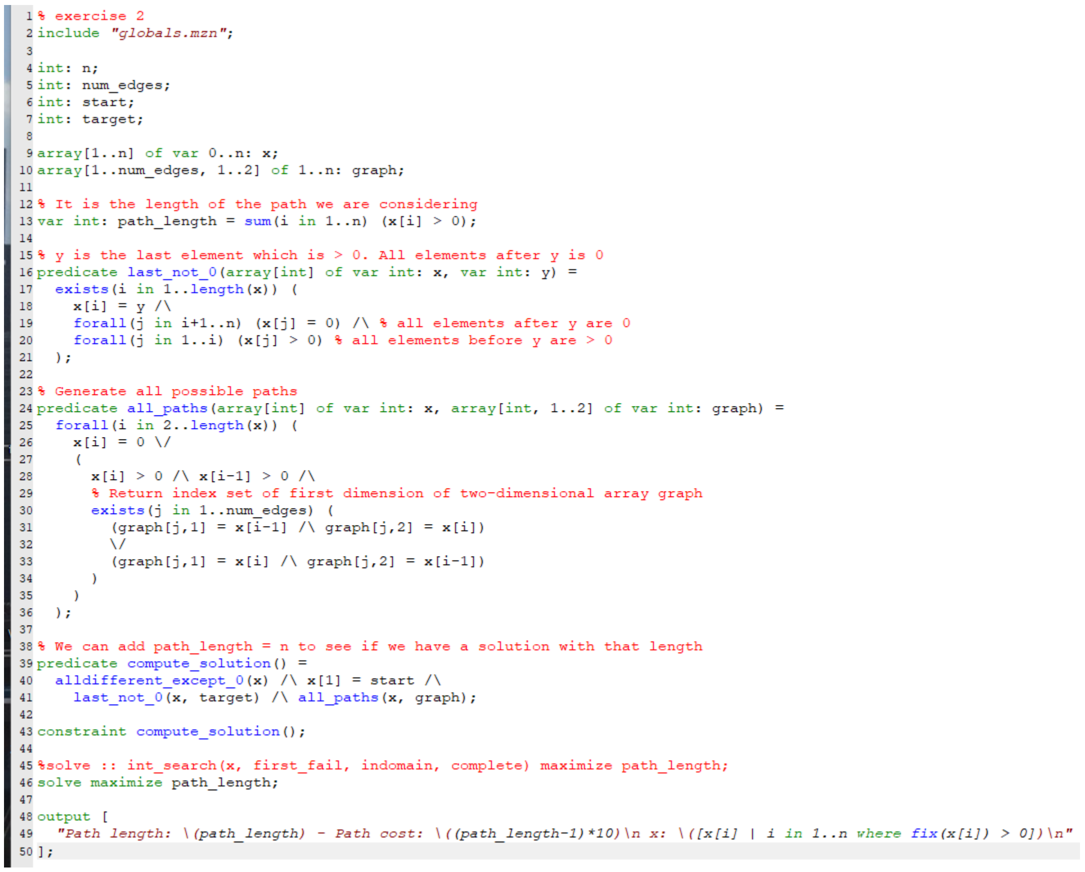
```
predicate all_paths(array[int] of var int: x, array[int, 1..2] of var int: graph) =
    forall(i in 2..length(x)) (
        x[i] = 0 \/
( x[i] > 0 /\ x[i-1] > 0 \/ exists(j in 1..num_edges) (
(graph[j,1] = x[i-1] /\ graph[j,2] = x[i])
/
(graph[j,1] = x[i] /\ graph[j,2] = x[i-1])
)
)
);
```

### 4.1.3 Solution

The `compute_solution` predicate uses the `alldifferent_except_0` constraint to ensure that each node in the path is unique and not equal to 0. It also sets the first node to be the starting node and ensures that the last node is the ending node using the `last_not_0` predicate. The `all_paths` predicate is then used to generate all possible paths in the maze. Finally, the "solve" statement maximizes the path length, and the output statement displays the path length and the path cost in meters:

```
predicate compute\_solution() =
alldifferent\_except_0(x) /\ x[1] = start /\
last\_not\_0(x, target) /\ all_paths(x, graph);

constraint compute\_solution();
solve maximize path\_length;
output [
"Path length: (path_length) - Path cost: ((path_length-1)*10)\
```

### 4.1.4 Code



```
% exercise 2
include "globals.mzn";

int: n;
int: num_edges;
int: start;
int: target;

array[1..n] of var 0..n: x;
array[1..num_edges, 1..2] of 1..n: graph;

% It is the length of the path we are considering
var int: path_length = sum(i in 1..n) (x[i] > 0);

% y is the last element which is > 0. All elements after y is 0
predicate last_not_0(array[int] of var int: x, var int: y) =
  exists(i in 1..length(x)) (
    x[i] = y /\
    forall(j in i+1..n) (x[j] = 0) /\ % all elements after y are 0
    forall(j in 1..i) (x[j] > 0) % all elements before y are > 0
  );

% Generate all possible paths
predicate all_paths(array[int] of var int: x, array[int, 1..2] of var int: graph) =
  forall(i in 2..length(x)) (
    x[i] = 0 \/
    (
      x[i] > 0 /\ x[i-1] > 0 /\
      % Return index set of first dimension of two-dimensional array graph
      exists(j in 1..num_edges) (
        (graph[j,1] = x[i-1] /\ graph[j,2] = x[i])
        \/
        (graph[j,1] = x[i] /\ graph[j,2] = x[i-1])
      )
    )
  );

% We can add path_length = n to see if we have a solution with that length
predicate compute_solution() =
  alldifferent_except_0(x) /\ x[1] = start /\
    last_not_0(x, target) /\ all_paths(x, graph);

constraint compute_solution();

%solve :: int_search(x, first_fail, indomain, complete) maximize path_length;
solve maximize path_length;

output [
  "Path length: \(path_length) - Path cost: \((path_length-1)*10)\n x: \([x[i] | i in 1..n where fix(x[i]) > 0])\n"
];
```

Figure 11: The maze

## 4.2 A MAXIMUM TRIANGLE (6)

Debora, after crossing out all nine numbers (see previous question), issues a challenge to Alessandro: ”Write the nine numbers, one in each box, so that the sum of the three sums of the numbers on the same side of the triangle is the maximum possible. How much is this sum worth?

### 4.2.1 Variables

- side1: is an array of 4 variables each of them with a domain from 1 to 9;
- side2: is an array of 4 variables each of them with a domain from 1 to 9;
- side3: is an array of 4 variables each of them with a domain from 1 to 9;
- sum1: this variable represents the sum along side1;
- sum2: this variable represents the sum along side2;
- sum3: this variable represents the sum along side3;
- f_sum: this variable represents the sum between all the sides of the triangle.

### 4.2.2 Constraints

To find a solution are written 6 different constraints:

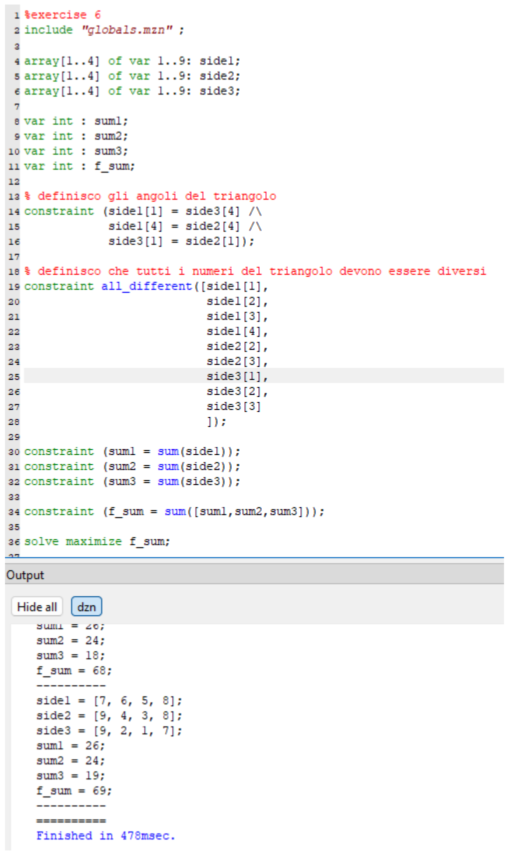
- the first constraint is for the definition of the corner values;
- the second constraint defines that all numbers in the triangle must be different, except for the values in the angles;
- the last fourth constraints are used to calculate the sum of all the numbers in the triangle.

### 4.2.3 Solution

Starting with the last constraint defined to find a solution to the problem, f_sum must be maximised:

$$side1 = [7,6,5,8]; side2 = [9,4,3,8]; side3 = [9,2,1,7]; sum1 = 26; sum2 = 24; sum3 = 19; f_sum = 69; \quad (9)$$

### 4.2.4 Code



```
%exercise 6
include "globals.mzn" ;

array[1..4] of var 1..9: side1;
array[1..4] of var 1..9: side2;
array[1..4] of var 1..9: side3;

var int : sum1;
var int : sum2;
var int : sum3;
var int : f_sum;

% definisco gli angoli del triangolo
constraint (side1[1] = side3[4] /\
            side1[4] = side2[4] /\
            side3[1] = side2[1]);

% definisco che tutti i numeri del triangolo devono essere diversi
constraint all_different([side1[1],
                          side1[2],
                          side1[3],
                          side1[4],
                          side2[2],
                          side2[3],
                          side3[1],
                          side3[2],
                          side3[3]
                          ]);

constraint (sum1 = sum(side1));
constraint (sum2 = sum(side2));
constraint (sum3 = sum(side3));

constraint (f_sum = sum([sum1,sum2,sum3]));

solve maximize f_sum;
```

```
sum1 = 26;
sum2 = 24;
sum3 = 18;
f_sum = 68;
----------
side1 = [7, 6, 5, 8];
side2 = [9, 4, 3, 8];
side3 = [9, 2, 1, 7];
sum1 = 26;
sum2 = 24;
sum3 = 19;
f_sum = 69;
----------
==========
Finished in 478msec.
```

Figure 12: A maximum triangle

## 4.3 FOUR DIGITS FOR A DATE (9)

19.09.2021 (19 September 2021) is written using four digits, each used twice. What is the next date (written with the same format as the previous one) that has the same property of being written with four digits, each used twice?

### 4.3.1 Variables

- date: array representing the digits to be found, each element has the following domain of values D = (0..9);
- g_date: variable that redefines the date as a single integer;
- old_date: variable that redefines the old date as a single integer.

### 4.3.2 Constraints

A number of constraints have been defined:

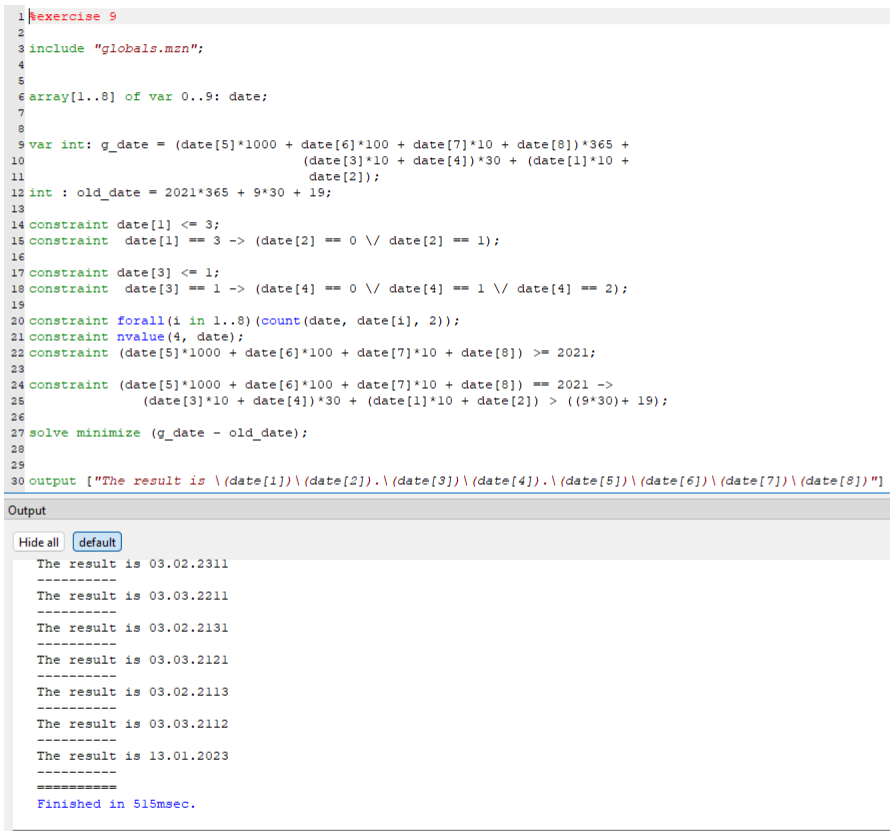
- two constraints for the first digit, because the days are at most 31 so `date[1]` it should be at most 3 and if the first digit is 3 the second one it should be 0 or 1;
- two constraints on the third digit, because the month are at most 12 so `date[3]` it should be at most 1 and if it is 1 `date[4]` it should be 0, 1 or 2;
- a constraint dictates that digits may be repeated no more than once.
- a constraint checks if the amount of different digits are equal to 4;
- a constraint requiring that the year of the new date must be later or equal than 2021;
- if it is indeed 2021, that the day of the month be later than 19.09.

### 4.3.3 Solution

This is an optimisation problem where the distance between `g_date - old_date` must be minimised:

$$\textit{The result is } 13.01.2023 \tag{10}$$

### 4.3.4 Code



```
%exercise 9

include "globals.mzn";


array[1..8] of var 0..9: date;


var int: g_date = (date[5]*1000 + date[6]*100 + date[7]*10 + date[8])*365 +
                                 (date[3]*10 + date[4])*30 + (date[1]*10 +
                                  date[2]);
int : old_date = 2021*365 + 9*30 + 19;

constraint date[1] <= 3;
constraint  date[1] == 3 -> (date[2] == 0 \/ date[2] == 1);

constraint date[3] <= 1;
constraint  date[3] == 1 -> (date[4] == 0 \/ date[4] == 1 \/ date[4] == 2);

constraint forall(i in 1..8)(count(date, date[i], 2));
constraint nvalue(4, date);
constraint (date[5]*1000 + date[6]*100 + date[7]*10 + date[8]) >= 2021;

constraint (date[5]*1000 + date[6]*100 + date[7]*10 + date[8]) == 2021 ->
              (date[3]*10 + date[4])*30 + (date[1]*10 + date[2]) > ((9*30)+ 19);

solve minimize (g_date - old_date);


output ["The result is \(date[1])\(date[2]).\(date[3])\(date[4]).\(date[5])\(date[6])\(date[7])\(date[8])"]
```

```
The result is 03.02.2311
----------
The result is 03.03.2211
----------
The result is 03.02.2131
----------
The result is 03.03.2121
----------
The result is 03.02.2113
----------
The result is 03.03.2112
----------
The result is 13.01.2023
----------
==========
Finished in 515msec.
```

Figure 13: Four digits for a date

## 4.4 MAGIC! (12)

Text: *Carla takes a (positive integer) two-digit number, multiplies it by 4 and then subtracts 3 from the result thus obtained. Magic: the final number obtained by Carla is written with the same digits as the starting number, but in reverse order. starting number, but in reverse order.*
Problem: *What was the starting number?*

### 4.4.1 Variables

The only variables to be defined are the digits of the number to be guessed:

- n1: represent the first digit of the number (domain: $n1 \in [1, 9]$);
- n2: represent the second digit of the number (domain: $n2 \in [0, 9]$).

### 4.4.2 Constraints

To solve that problem only one constraint is implemented,the following mathematical constraint on the number can be extracted from the text:

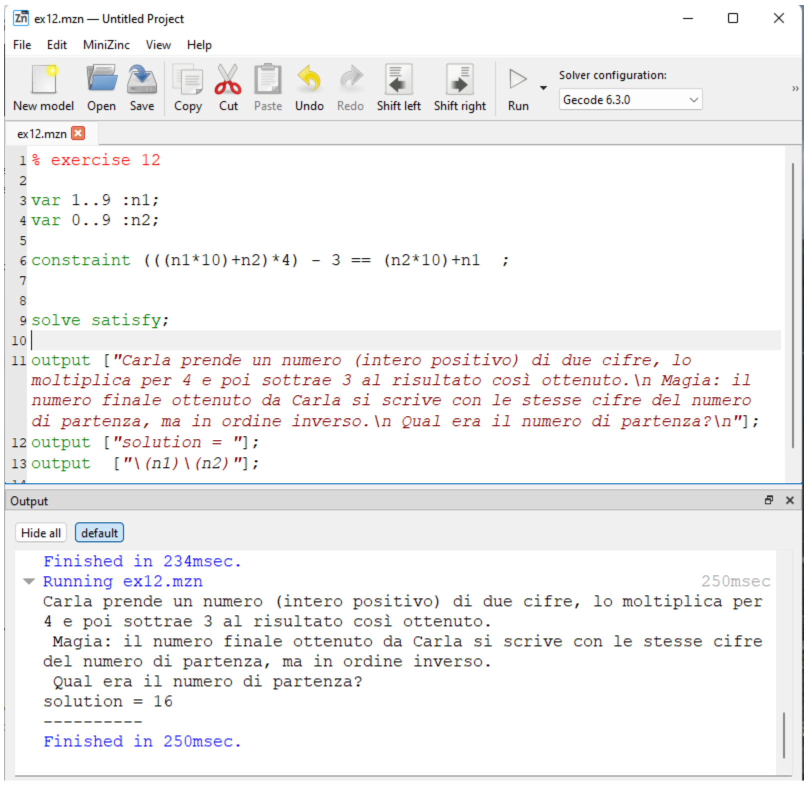
$$[(n1 * 10 + n2) * 4] - 3 == n2 * 10 + n1$$

### 4.4.3 Solution

It is a satisfaction problem and the solution that we have found is : 16

$$(16 * 4) - 3 = 61$$

### 4.4.4 Code


ex12.mzn — Untitled Project
File Edit MiniZinc View Help
New model Open Save Copy Cut Paste Undo Redo Shift left Shift right Run
Solver configuration:
Gecode 6.3.0
ex12.mzn
Output
Hide all default

```
% exercise 12

var 1..9 :n1;
var 0..9 :n2;

constraint (((n1*10)+n2)*4) - 3 == (n2*10)+n1  ;


solve satisfy;

output ["Carla prende un numero (intero positivo) di due cifre, lo
moltiplica per 4 e poi sottrae 3 al risultato così ottenuto.\n Magia: il
numero finale ottenuto da Carla si scrive con le stesse cifre del numero
di partenza, ma in ordine inverso.\n Qual era il numero di partenza?\n"];
output ["solution = "];
output  ["\(n1)\(n2)"];
```

```
Finished in 234msec.
Running ex12.mzn                                          250msec
Carla prende un numero (intero positivo) di due cifre, lo moltiplica per
4 e poi sottrae 3 al risultato così ottenuto.
 Magia: il numero finale ottenuto da Carla si scrive con le stesse cifre
del numero di partenza, ma in ordine inverso.
 Qual era il numero di partenza?
solution = 16
----------
Finished in 250msec.
```

Figure 14: Magic! minizinc implementation.

## 4.5 WITH THE TEN DIGITS (17)

A positive integer number N is such that all ten digits from 0 to 9 are used to write its third power, N3, and its fourth power, N4. What is the value of N?

### 4.5.1 Variables

- N: is the positive integer to be found;
- power3: is the third power of N;
- power4: is the fourth power of N;
- power3_digits: is an array of 10 element, each of them represents a digit of power3;
- power4_digits: is an array of 10 element, each of them represents a digit of power4.

### 4.5.2 Constraints

A number of constraints have been defined:

- The sum of the powers of 10 multiplied by each digit in the power3_digits array must be equal to the value of power3.
- The sum of the powers of 10 multiplied by each digit in the power4_digits array must be equal to the value of power4.
- The value of power3 must be equal to N raised to the power of 3.
- The value of power4 must be equal to N raised to the power of 4.
- The digits in the power3_digits and power4_digits arrays must be pairwise distinct, except for the value 0.

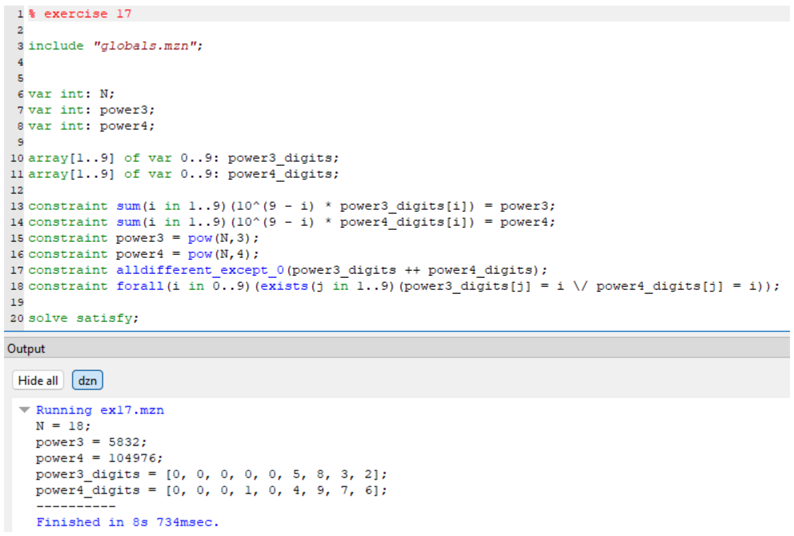
- For each digit between 0 and 9, there must be at least one index in the power3_digits or power4_digits arrays that contains that digit.

### 4.5.3 Solution

This is a problem of satisfiability and the solution is:

$$\begin{aligned} N &= 18; \\ power3 &= 5832; \\ power4 &= 104976; \\ power3\_digits &= [0,0,0,0,0,5,8,3,2]; \\ power4\_digits &= [0,0,0,1,0,4,9,7,6]; \end{aligned}$$

### 4.5.4 Code



```
% exercise 17

include "globals.mzn";


var int: N;
var int: power3;
var int: power4;

array[1..9] of var 0..9: power3_digits;
array[1..9] of var 0..9: power4_digits;

constraint sum(i in 1..9)(10^(9 - i) * power3_digits[i]) = power3;
constraint sum(i in 1..9)(10^(9 - i) * power4_digits[i]) = power4;
constraint power3 = pow(N,3);
constraint power4 = pow(N,4);
constraint alldifferent_except_0(power3_digits ++ power4_digits);
constraint forall(i in 0..9)(exists(j in 1..9)(power3_digits[j] = i \/ power4_digits[j] = i));

solve satisfy;
```

```
Running ex17.mzn
N = 18;
power3 = 5832;
power4 = 104976;
power3_digits = [0, 0, 0, 0, 0, 5, 8, 3, 2];
power4_digits = [0, 0, 0, 1, 0, 4, 9, 7, 6];
----------
Finished in 8s 734msec.
```

Figure 15: With the ten digits