COMP3620/COMP6320 Artificial Intelligence
Lab 3: Constraint Programming

April 27 - May 1, 2017

The lab exercises go through the basics of coding problems in MiniZinc. The following three problems are not particularly difficult, but they illustrate many of the features of MiniZinc which are required for the assignment. They also serve as examples of how to run MiniZinc with its default solver and how to think about turning problems into CSP specifications.

Problem 1: Integer Inequalities

Able, Baker, Charlie and Dog are all integers.

- Able is positive (greater than zero), Baker is less than 7 and Charlie is greater than 3.
- Dog is less than the sum of Able and Baker.
- Dog times 2 is greater than Able, Baker and Charlie all added together.
- Baker minus Charlie is greater than Able.

Find Able, Baker, Charlie and Dog.

This problem is intended as an introduction to MiniZinc syntax: to represent it, you need to declare decision variables, state simple constraints and include a `solve` item. The example on page 4 of the MiniZinc tutorial should serve as a guide.

Begin by creating a file called `ABCD.mzn`. You can edit it using any text editor you prefer, as long as it produces plain text. It is possible to use the MiniZinc IDE for this, but we do not recommend it, as it takes a while to lesarn, and it may make assumptions about default solvers etc. which are not appropriate for our purposes. Write your MiniZinc model in the file and save it. Open a terminal for command line input and go to the directory where your file is. To solve the problem, simply type

```
minizinc ABCD.mzn
```

To generate all solutions, and check that there is only one (this problem only has one solution), type:

```
minizinc -a ABCD.mzn
```

Problem 2: Office Blocked

Six hardworking employees of the Paper Circulation Division of the Department of Miscellanea occupy offices on two floors of the Administration Building: three on the upper floor (offices U1, U2 and U3) and the other three directly below them on the lower floor (L1, L2 and L3).

1. Arthur is directly above Bella, who works next to Duncan.

2. Elizabeth's office number is smaller than Francesca's.
3. Craig and Francesca are in adjacent rooms.

Who works where?

You can start your MiniZinc encoding by enumerating (listing) the six employees. Then you need to declare some arrays of decision variables in order to give yourself the vocabulary to state the constraints. So the first three items in your MiniZinc file could be:

```
enum person = {A, B, C, D, E, F};
array[person] of var 1..2: floor;
array[person] of var 1..3: roomNumber;
. . .
```

Again, there is only one solution for this problem. Run the MiniZinc solver with the `-a` option to make sure you get only one solution.

Problem 3: Magic Squares

A magic square of order N consists of all the integers from 1 to N^2 arranged in an $N \times N$ square, in such a way that every row, every column and each of the diagonals sums to the same value $\mathcal{V}$. Elementary arithmetic tells us that $\mathcal{V} = (N^3 + N)/2$. Magic squares exist for every positive integer N except $N = 2$.

Write a MiniZinc file `magic.mzn` to generate a magic square of order N and print the square as output. For example, for $N = 3$ it might print

```
2  7  6
9  5  1
4  3  8
```

Write a MiniZinc data file `magic4.dzn` containing the one line

```
N = 4;
```

Then type the command

```
minizinc -a magic.mzn magic4.dzn
```

to see 7040 solutions, or omit the `-a` switch to see just one. Try with another `.dzn` file setting N to 5, but don't use `-a` when evaluating this one! How big can your magic squares get before the solver takes too long to find one?

Now try something a little different. Let's define the "badness" of a magic square to be the sum of its four corner values, and with that definition find the best magic square of a given size. Alter the `solve` item of your `magic.mzn` to generate a solution minimising the sum of the corner cells.