Logic and Constraint Programming

3- Optimization and more complex models

A.A. 2021/2022

Lorenzo Rossi

lorenzo.rossi@unicam.it

University of Camerino

MINIZINC
≫OPTIMIZATION

```
solve maximize 400*b + 450*c;
```

We want to find a solution that maximises the **expression** in the **solve statement** called the objective

- The objective can be any kind of arithmetic expression
- One can replace the key word maximize by minimize to specify a minimisation problem

```
Prepare 2 banana cakes, and 2chocolate cakes, now!

----------

==========
```

The line ========== is output automatically for optimisation problems when the system has proved that a solution is optimal

MINIZINC
≫DATA FILES

Model input data can be loaded from file (*.dzn*) or from bash

Model:

```
var int:  A;
int:  B;
```

Data file:

```
A = 12;
B = 2;
```

```
minizinc model.mzn data.dzn
```

MINIZINC
≫ASSERTION

Defensive programming suggests that we should check that the values in the data file are reasonable

In case of our example, to check that the quantity of all ingredients is non-negative and generate a run-time error if this is not true

MiniZinc provides a built-in boolean operator

- The form is assert(b,s)

```
constraint assert(flour >= 0, "Invalid data:  flour
must be positive");
```

MINIZINC
≫ASSERTION

Defensive programming suggests that we should check that the values in the data file are reasonable

In case of our example, to check that the quantity of all ingredients is non-negative and generate a run-time error if this is not true

MiniZinc provides a built-in boolean operator

- The form is assert(b,s)

```
constraint assert(flour >= 0, "Invalid data:  flour
must be positive");
```

Let's try the bakery example using data files and asserting the input parameters.

MINIZINC
≫LOAN EXAMPLE

Bob has a problem, he need money for buying a new oven. He would like to take a short loan for one year to be repaid in 4 quarterly instalments. It uses a simple interest calculation to calculate the balance after each quarter

Can we use the same model to answer a number of different questions?

MINIZINC
≫LOAN EXAMPLE

- **Which are the decision variables?**

MINIZINC
≫LOAN EXAMPLE

- **Which are the decision variables?** The interest rate, the borrowed amount, the quarterly repayment, and the balances after each quarter.

MINIZINC
≫LOAN EXAMPLE

- **Which are the decision variables?** The interest rate, the borrowed amount, the quarterly repayment, and the balances after each quarter.
- **Which are the constraints?**

MINIZINC
≫LOAN EXAMPLE

- **Which are the decision variables?** The interest rate, the borrowed amount, the quarterly repayment, and the balances after each quarter.
- **Which are the constraints?** Every balance has to be equal to the remaining borrowed amount times the interest rate, excluding the repayment.

MINIZINC
≫LOAN EXAMPLE

```
% variables
var float: R; % quarterly repayment
var float: P; % principal initially borrowed
var 0.0 .. 10.0: I; % interest rate
% intermediate variables
var float: B1; % balance after one quarter
var float: B2; % balance after two quarters
var float: B3; % balance after three quarters
var float: B4; % balance owing at end
constraint B1 = P * (1.0 + I) - R;
constraint B2 = B1 * (1.0 + I) - R;
constraint B3 = B2 * (1.0 + I) - R;
constraint B4 = B3 * (1.0 + I) - R;
solve satisfy;
output [
"Borrowing ", show_float(0, 2, P), " at ", show(I*100.0),
"% interest, and repaying ", show_float(0, 2, R),
"\nper quarter for 1 year leaves ", show_float(0, 2, B4), " owing\n"
];
```

MINIZINC
≫REAL NUMBERS

Since we wish to use **real number variables and constraint** we need to use a **solver that supports this type of problem**.

The MiniZinc distribution contains such a solver. We can invoke it by selecting COIN-BC from the solver menu in the IDE (the triangle below the Run button), or on the command line using the command minizinc `--solver cbc`:

```
$ minizinc --solver cbc loan.mzn loan1.dzn
```

MINIZINC
≫LOAN EXAMPLE

- **If I borrow 1000 at 4% and repay 260 per quarter, how much do I end up owing?**

MINIZINC
≫LOAN EXAMPLE

- **If I borrow 1000 at 4% and repay 260 per quarter, how much do I end up owing?** `I = 0.04; P = 1000.0; R = 260.0`

MINIZINC
≫LOAN EXAMPLE

- **If I borrow 1000 at 4% and repay 260 per quarter, how much do I end up owing?** `I = 0.04; P = 1000.0; R = 260.0`
- **If I want to borrow 1000 at 4% and owe nothing at the end, how much do I need to repay?**

MINIZINC
≫LOAN EXAMPLE

- **If I borrow 1000 at 4% and repay 260 per quarter, how much do I end up owing?** `I = 0.04; P = 1000.0; R = 260.0`
- **If I want to borrow 1000 at 4% and owe nothing at the end, how much do I need to repay?** `I = 0.04; P = 1000.0; B4 = 0.0;`

MINIZINC
≫LOAN EXAMPLE

- **If I borrow 1000 at 4% and repay 260 per quarter, how much do I end up owing?** `I = 0.04; P = 1000.0; R = 260.0`
- **If I want to borrow 1000 at 4% and owe nothing at the end, how much do I need to repay?** `I = 0.04; P = 1000.0; B4 = 0.0;`
- **If I can repay 250 a quarter, how much can I borrow at 4% to end up owing nothing?**

MINIZINC
≫LOAN EXAMPLE

- **If I borrow 1000 at 4% and repay 260 per quarter, how much do I end up owing?** `I = 0.04; P = 1000.0; R = 260.0`
- **If I want to borrow 1000 at 4% and owe nothing at the end, how much do I need to repay?** `I = 0.04; P = 1000.0; B4 = 0.0;`
- **If I can repay 250 a quarter, how much can I borrow at 4% to end up owing nothing?** `I = 0.04; R = 250.0; B4 = 0.0;`

Minizinc. More Complex Models

COMPREHENSION

Minizinc permits to build a list/set of elements from other lists/sets (Use {} for sets and [] for lists)

```
[a[i] != a[j] | i,j in 1..100 where i<j]
```

Example

`[i + j | i, j in 1..3 where j < i]` evaluates to `[2 + 1, 3 + 1, 3 + 2]` which is `[3, 4, 5]`

AGGREGATION

Aggregation functions for arithmetic arrays are: `sum`, `product`, `min`, `max`; for booleans are `forall`, `exists`, `xorall`, `iffall`

Example

`product( [i | i in 1..10])` evaluates to `sum([1..10])` which is `10!`

MINIZINC
≫MORE COMPLEX MODELS

Consider a model for calculating temperatures on a rectangular sheet of metal. We approximate the temperatures across the sheet by breaking the sheet into a finite number of elements in a 2D matrix

0	100	100	100	0
0	?	?	?	0
0	?	?	?	0
0	?	?	?	0
0	0	0	0	0

MINIZINC
≫TEMPERATURE MODEL

```
set of int: HEIGHT = 0..h;
set of int: CHEIGHT = 1..h−1;
set of int: WIDTH = 0..w;
set of int: CWIDTH = 1..w−1;
array[HEIGHT,WIDTH] of var float: t; % temperature at point (i,j)
```

Laplace's equation states that when the plate reaches a steady state the temperature at each internal point is the average of its orthogonal neighbours.

```
% Laplace equation: each internal temp. is average of its neighbours
constraint forall(i in CHEIGHT, j in CWIDTH)(4.0*t[i,j] = t[i−1,j] + t[i,j−1] + t[i+1,j] + t[i,j+1]);
```

MINIZINC
≫TEMPERATURE MODEL

We restrict the temperatures on each edge to be equal, and ensure that the corners (which are irrelevant) are set to 0.0

```
% edge constraints
constraint forall(i in CHEIGHT)(t[i,0] = left);
constraint forall(i in CHEIGHT)(t[i,w] = right);
constraint forall(j in CWIDTH)(t[0,j] = top);
constraint forall(j in CWIDTH)(t[h,j] = bottom);
% corner constraints
constraint t[0,0]=0.0;
constraint t[0,w]=0.0;
constraint t[h,0]=0.0;
constraint t[h,w]=0.0
```

MINIZINC
≫TEMPERATURE MODEL

We print the solution

```
solve satisfy;
output [ show_float(6, 2, t[i,j]) ++
if j == w then "\n" else " " endif |
i in HEIGHT, j in WIDTH
];
```

-0.00	100.00	100.00	100.00	-0.00
-0.00	42.86	52.68	42.86	-0.00
-0.00	18.75	25.00	18.75	-0.00
-0.00	7.14	9.82	7.14	-0.00
-0.00	-0.00	-0.00	-0.00	-0.00