Logic and Constraint Programming

4- CSP Conclusions

A.A. 2021/2022

Lorenzo Rossi

lorenzo.rossi@unicam.it

University of Camerino

MiniZinc
≫Bakery 2.0

Can we enhance the **bakery model** to be more generic by using arrays, sets, enumerations, aggregation, comprehension?

```
% Products to be produced
enum Products;                                      = {banana, chocolate}
% profit per unit for each product
array[Products] of int: profit;                     = [400,500]
% Resources to be used
enum Resources;                                     = {flour, bananas, sugar, butter, cocoa}
% amount of each resource available
array[Resources] of int: capacity;                  = [4000, 6, 2000, 500, 500]
% units of each resource required to produce 1 unit of
      product
array[Products, Resources] of int: consumption;     = [|250, 2, 75, 100, 0, | 200, 0, 150, 150, 75|]

constraint assert(forall (r in Resources, p in Products)
           (consumption[p,r] >= 0), "Error: negative
                 consumption");
```

MiniZinc
≫Bakery 2.0

```
% bound on number of Products
int: mproducts = max ([(min ([capacity[r] div consumption[p,r] | r
        in Resources where consumption[p,r] > 0])) | p in
        Products]);

% Variables: how much should we make of each product
array[Products] of var 0..mproducts: produce;
array[Resources] of var 0..max(capacity): used;
```

maximum amount of any product that Bob can produce
= max { min {4000 div 250, 6 div 2, ...}, min {4000 div 200, 2000 div 150, ...}}

MiniZinc
≫Bakery 2.0

```
% Production cannot use more than the available Resources:
constraint forall (r in Resources) (used[r] = sum (p in Products)(
        consumption[p, r] * produce[p]) );
constraint forall (r in Resources) (used[r] <= capacity[r]);

% Maximize profit
solve maximize sum (p in Products) (profit[p]*produce[p]);

output [ "\(p) = \(produce[p]);\n" | p in Products ] ++
        [ "\(r) = \(used[r]);\n" | r in Resources ];
```

MINIZINC
≫ALLDIFFERENT

Minizinc provides **global constraints** to use for defining models. (see the handbook for an exhaustive list)

The `alldifferent` constraint requires its argument to be pairwise different.

`alldifferent(set/list/range X)`

$\forall x_i, x_j \in X$, with $i \neq j$, then $x_i \neq x_j$

MINIZINC
≫CRYPTARITHMETIC

```
include "alldifferent.mzn";
var 1..9: S;
var 0..9: E;
var 0..9: N;
var 0..9: D;
var 1..9: M;
var 0..9: O;
var 0..9: R;
var 0..9: Y;
constraint 1000 * S + 100 * E + 10 * N + D
+ 1000 * M + 100 * O + 10 * R + E
= 10000 * M + 1000 * O + 100 * N + 10 * E + Y;
constraint alldifferent([S,E,N,D,M,O,R,Y]);
solve satisfy;
output [show(S),show(E),show(N),show(D),"+ \n",
  show(M),show(O),show(R),show(E),"= \n",
  show(M),show(O),show(N),show(E),show(Y),"\n",];
```

Conditional Expressions

MiniZinc provides a conditional `if-then-else-endif` expression. An example which sets r to x divided by y unless y is zero in which case it sets it to zero, is the following

```
int: r = if y != 0 then x div y else 0 endif;
```

Sudoku

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

Exercise

Implement a model for solving Sudoku puzzles. The model receives a 9x9 array of integer representing the initial schema, and gives as output the solved puzzle.

Sudoku

≫Solution

```
include "alldifferent.mzn";

int: S;
int: N = S * S;

set of int: PuzzleRange = 1..N;
set of int: SubSquareRange = 1..S;

array[1..N,1..N] of 0..N: start; % initial board 0 = empty
array[1..N,1..N] of var PuzzleRange: puzzle;
```

Sudoku
≫Solution

```
% fill initial board
constraint forall(i,j in PuzzleRange)(
if start[i,j] > 0 then puzzle[i,j] = start[i,j] else true endif );

% All different in rows
constraint forall (i in PuzzleRange) (
alldifferent( [ puzzle[i,j] | j in PuzzleRange ]) );

% All different in columns.
constraint forall (j in PuzzleRange) (
alldifferent( [ puzzle[i,j] | i in PuzzleRange ]) );

% All different in sub–squares:
constraint
forall (a, o in SubSquareRange)(
alldifferent( [ puzzle[(a–1) *S + a1, (o–1)*S + o1] | a1, o1 in SubSquareRange ] ) );
```

Sudoku
≫Solution

```
solve satisfy;
output [ show(puzzle[i,j]) ++ " " ++
if j mod S == 0 then " " else "" endif ++
if j == N then
if i != N then
if i mod S == 0 then "\n\n" else "\n" endif
else "" endif else "" endif
| i,j in PuzzleRange ] ++ ["\n"];
```

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

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

Constraints Programming: History and Applications

Constraint Programming
≫History

Constraint Programming has a long tradition, the initial ideas leading to CP appeared in 1960s and 1970s.

In early 1960, Ivan Sutherland (1988 Turing award) developed, the first application for interactive graphics, i.e., **Sketchpad**. The application allows users to draw and manipulate **constrained geometric figures** on computer's display, at that time, a **constraint language for graphical interaction** was introduced

https://www.youtube.com/watch?v=6orsmFndx_o

Constraint Programming
≫History

Since 1980s, no robust academic or commercial constraint systems were available ready to be used in industrial applications

In the 1990s, the first constraint programming systems emerged, and practical application of constraint programming, like **circuit verification**, **scheduling**, or **resource allocation**, arose

Constraint Programming
≫History

The ILOG Solver was one of the first CSP solvers delivered as a C++ library. It is now part of the IBM ILOG CPLEX Studio

In the first decade of 2000, constraint systems like Choco, JaCoP, Gecode, Minion, or Google CP Solver turned up and became more and more mature

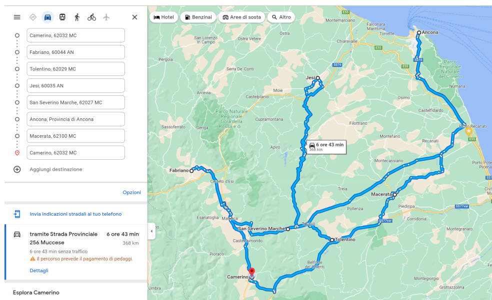
In the last years, constraint systems are available as **libraries** and can read constraint **models specified in quasi standard languages** like MiniZinc or the Java Constraint Programming API.

Constraint Programming Applications
≫Railway Interlocking Systems

In railway signaling, an interlocking system is an arrangement of signals and other equipment that prevents conflicting movements of trains in a network of tracks, switches, and crossings

Constraint Programming Applications
≫Vehicle Routing

The Vehicle Routing Problem optimizes the routes of delivery trucks, cargo lorries, public transportation (buses, taxis and airplanes) or technicians on the road, by improving the order of the visits



https://goo.gl/maps/chxSsZQzLP8fx6qQ9

Constraint Programming Applications

≫School timetabling

In universities, schools, and other similar contexts, the timetabling problem is to provide timetables for lessons respecting constraints such as room capacity and availability, professors availability, lessons overlapping avoidance, and so on.

	Monday	Room	Tuesday	Room	Wednesday	Room	Thursday	Room	Friday	Room
9am - 10am	PM	AB1	KE	AB1	LCP	AB1	ESI	AB1	LCP	AB1
10am - 11am	PM	AB1	KE	AB1	LCP	AB1	ESI	AB1	LCP	AB1
11am - 12pm	ESI	AB1	KE ISF	AB1 LB1	ESA	LB1	PM	AB1	SVL	AB1
12pm - 1pm	ESI	AB1	KE ISF	AB1 LB1	ESA	LB1	PM	AB1	SVL	AB1
2pm - 3pm	KE	AB1	DS	LB1	SVL	AV1	ISF	AB1	ESA	LB1
3pm - 4pm	KE	AB1	DS	LB1	SVL	AB1	ISF	AB1	ESA	LB1
4pm - 5pm	KE	AB1	FMS	LB1	FMS	LB1	DS	LB1	ESA	LB1
5pm - 6pm	KE	AB1	FMS	LB1	FMS	LB1	DS	LB1		
6pm - 7pm										

First assignment

Bob is a fan of retro games. He is intended to program a CSP model generating minesweeper mazes form a distribution of mines.