Knowledge Representation and Reasoning Introduction to Constraint Programming with MiniZinc

Antoni Ligęza

ligeza@agh.edu.pl

AGH University of Science and Technology
Kraków, Poland

Knowledge Representation and Reasoning
AGH Kraków
2017

Outline

1. Introduction: An Example to Start
2. Constraint Satisfaction Problems: Examples
3. Constraint Satisfaction Problem
4. Introduction to MiniZinc
5. Example: SEND+MORE=MONEY
6. Constraint Optimization Example
7. Using Data files
8. Constraint Programming with Real Numbers
9. More Complex Structures: Arrays and Sets

Presentation Outline

1. **Introduction: An Example to Start**
2. Constraint Satisfaction Problems: Examples
3. Constraint Satisfaction Problem
4. Introduction to MiniZinc
5. Example: SEND+MORE=MONEY
6. Constraint Optimization Example
7. Using Data files
8. Constraint Programming with Real Numbers
9. More Complex Structures: Arrays and Sets

Introduction: An Example to Start

Abduction

- Abduction — principal way of problem solving — generation of hypotheses,
- Abduction — hypotheses generation performed with backtracking search,
- Abduction — produces numerous, admissible solutions

Abduction: Logical model

$$\frac{\alpha \Longrightarrow \beta, \beta}{\alpha}$$

$$HYP^{+} \cup HYP^{-} \cup KB \models OBS^{+} \cup OBS^{-}$$

$$HYP^{+} \cup HYP^{-} \cup KB \cup OBS^{+} \cup OBS^{-} \not\models \bot$$

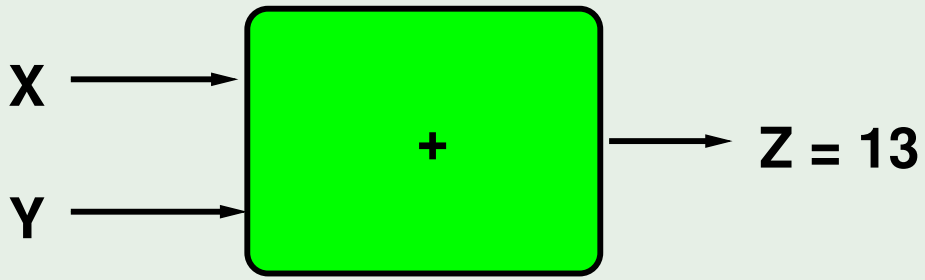
An intuitive example: find explanations for *wet_street*

- $rain \longrightarrow water$
- $sprinkler \longrightarrow water$
- $snow \wedge temperature \longrightarrow water$
- $water \longrightarrow wet\_street$,
- $cleaning \longrightarrow wet\_street$
- $oil \longrightarrow wet\_street$

The role of constraints in abduction

Abductive problem without constraints

- X, Y, Z - variables, $X, Y \in \{0, 1, 2, \ldots, 9\}$, $Z \in \{0, 1, 2, \ldots, 18\}$,
- system: $Z = X + Y$



- Observed: $Z = 13$
- Possible explanations:
  - $(X = 4$ and $Y = 9)$,
  - $(X = 5$ and $Y = 8)$,
  - ... ,
  - $(X = 9$ and $Y = 4)$.
- 6 admissible solutions.

The role of constraints in abduction

Abductive problem with constraints

- X, Y, Z - variables, $X, Y, Z \in \{0, 1, 2, \ldots, 9\}$,

$$Z = X + Y$$

- **Constraint:**

$$Y < X - 3$$

- Observed: $Z = 13$
- Possible explanations: $(X = 9$ and $Y = 4)$,
- 1 admissible solution.

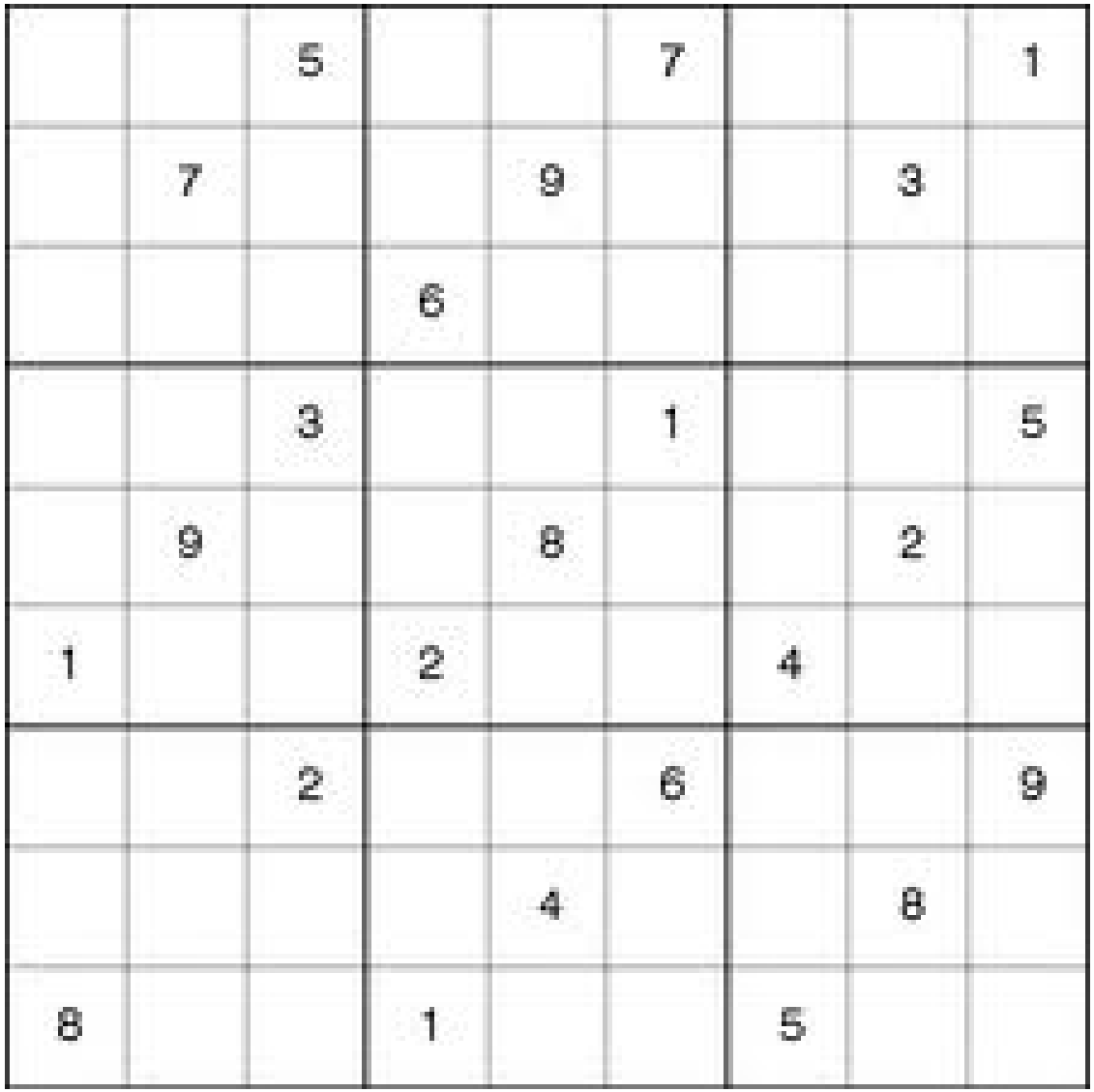
Conclusion

- CONSTRAINTS can refine results of ABDUCTION; less models generated,
- propagation of CONSTRAINTS can reduce computational effort,
- ABDUCTION + CONSTRAINTS = CONSTRUCTIVE ABDUCTION

Presentation Outline

1. Introduction: An Example to Start
2. **Constraint Satisfaction Problems: Examples**
3. Constraint Satisfaction Problem
4. Introduction to MiniZinc
5. Example: SEND+MORE=MONEY
6. Constraint Optimization Example
7. Using Data files
8. Constraint Programming with Real Numbers
9. More Complex Structures: Arrays and Sets

Constraint Satisfaction Problems: Examples



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

Figure : Sudoku: An example Constraint Satisfaction Problem

Constraint Satisfaction Problems: Examples

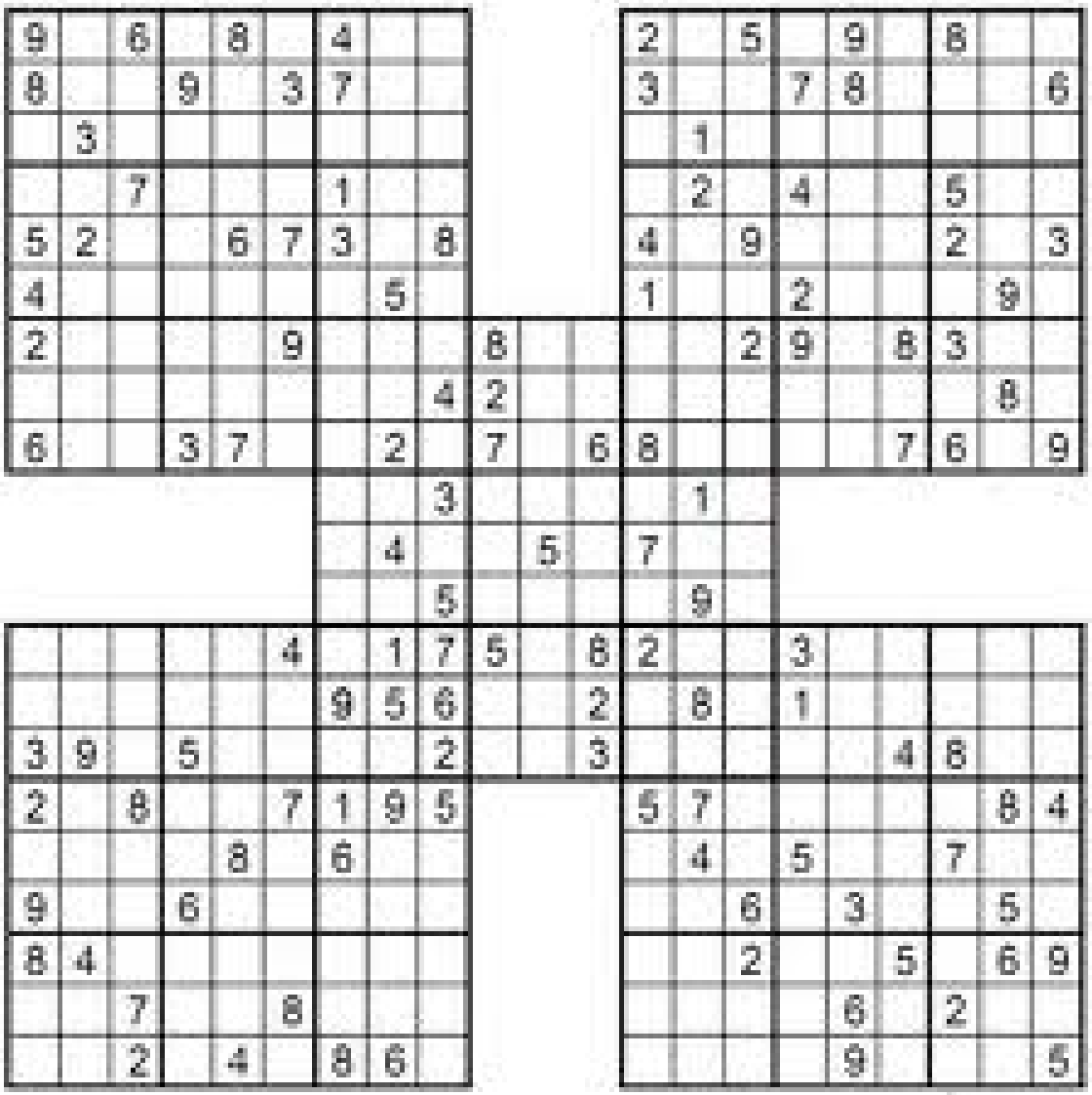


Figure : Sudoku: Yet another example Constraint Satisfaction Problem

Constraint Satisfaction Problems: Examples

Figure : An example Constraint Satisfaction Problem

Constraint Satisfaction Problems: Examples

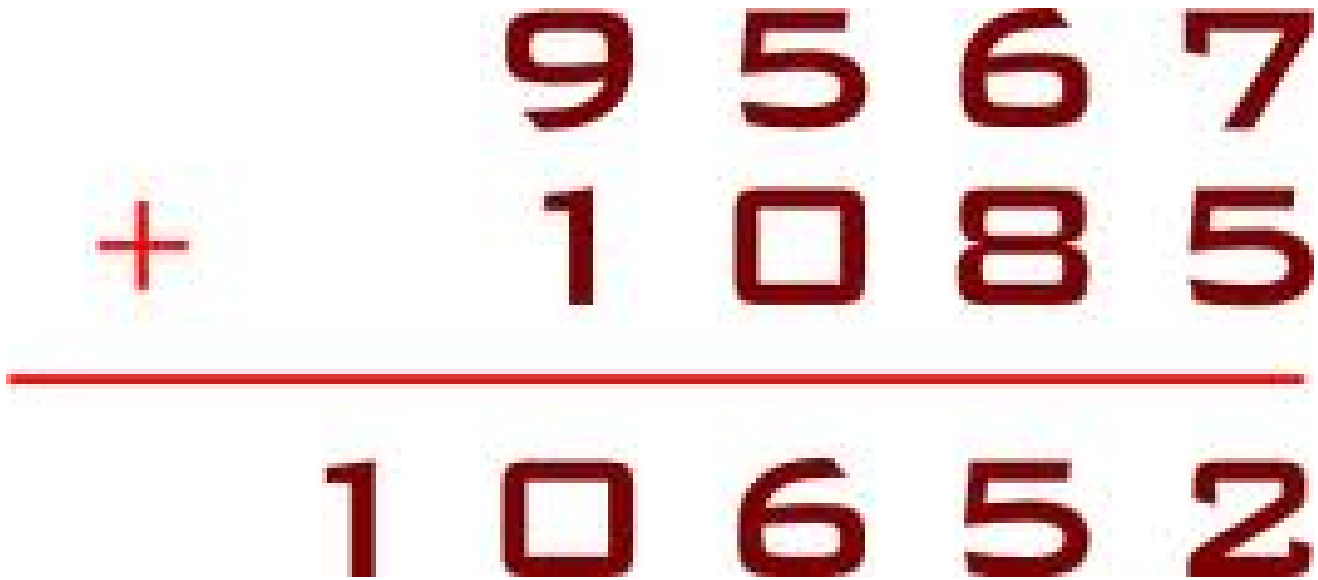


Figure : The unique solution of the example Constraint Satisfaction Problem

Constraint (Logic) Programming

- Declarative Programming + Multi-Purpose Models:
  - simple, transparent statement; practical problem,
  - zero, one, or many solutions,
  - problem: combinatorial explosion.
- Decision factors in CP/CLP:
  - variable — which variable too choose,
  - value — which value to choose,
  - propagation — how to propagate constraints,
  - heuristics — what heuristics can be used.
- CP/CLP basic solution paradigm:
  - select a variable,
  - select a value,
  - propagate constraints,
  - if conflict — backtrack; if unique — return solution; else — loop.

Constraint (Logic) Programming

- Declarative Programming + Multi-Purpose Models:
  - simple, transparent statement; practical problem,
  - zero, one, or many solutions,
  - problem: combinatorial explosion.
- Decision factors in CP/CLP:
  - variable — which variable too choose,
  - value — which value to choose,
  - propagation — how to propagate constraints,
  - heuristics — what heuristics can be used.
- CP/CLP basic solution paradigm:
  - select a variable,
  - select a value,
  - propagate constraints,
  - if conflict — backtrack; if unique — return solution; else — loop.

Constraint (Logic) Programming

- Declarative Programming + Multi-Purpose Models:
  - simple, transparent statement; practical problem,
  - zero, one, or many solutions,
  - problem: combinatorial explosion.
- Decision factors in CP/CLP:
  - variable — which variable too choose,
  - value — which value to choose,
  - propagation — how to propagate constraints,
  - heuristics — what heuristics can be used.
- CP/CLP basic solution paradigm:
  - select a variable,
  - select a value,
  - propagate constraints,
  - if conflict — backtrack; if unique — return solution; else — loop.

Constraint (Logic) Programming

- Declarative Programming + Multi-Purpose Models:
  - simple, transparent statement; practical problem,
  - zero, one, or many solutions,
  - problem: combinatorial explosion.
- Decision factors in CP/CLP:
  - variable — which variable too choose,
  - value — which value to choose,
  - propagation — how to propagate constraints,
  - heuristics — what heuristics can be used.
- CP/CLP basic solution paradigm:
  - select a variable,
  - select a value,
  - propagate constraints,
  - if conflict — backtrack; if unique — return solution; else — loop.

Constraint (Logic) Programming

- Declarative Programming + Multi-Purpose Models:
  - simple, transparent statement; practical problem,
  - zero, one, or many solutions,
  - problem: combinatorial explosion.
- Decision factors in CP/CLP:
  - variable — which variable too choose,
  - value — which value to choose,
  - propagation — how to propagate constraints,
  - heuristics — what heuristics can be used.
- CP/CLP basic solution paradigm:
  - select a variable,
  - select a value,
  - propagate constraints,
  - if conflict — backtrack; if unique — return solution; else — loop.

Constraint (Logic) Programming

- Declarative Programming + Multi-Purpose Models:
  - simple, transparent statement; practical problem,
  - zero, one, or many solutions,
  - problem: combinatorial explosion.
- Decision factors in CP/CLP:
  - variable — which variable too choose,
  - value — which value to choose,
  - propagation — how to propagate constraints,
  - heuristics — what heuristics can be used.
- CP/CLP basic solution paradigm:
  - select a variable,
  - select a value,
  - propagate constraints,
  - if conflict — backtrack; if unique — return solution; else — loop.

Constraint (Logic) Programming

- Declarative Programming + Multi-Purpose Models:
  - simple, transparent statement; practical problem,
  - zero, one, or many solutions,
  - problem: combinatorial explosion.
- Decision factors in CP/CLP:
  - variable — which variable too choose,
  - value — which value to choose,
  - propagation — how to propagate constraints,
  - heuristics — what heuristics can be used.
- CP/CLP basic solution paradigm:
  - select a variable,
  - select a value,
  - propagate constraints,
  - if conflict — backtrack; if unique — return solution; else — loop.

Constraint (Logic) Programming

- Declarative Programming + Multi-Purpose Models:
  - simple, transparent statement; practical problem,
  - zero, one, or many solutions,
  - problem: combinatorial explosion.
- Decision factors in CP/CLP:
  - variable — which variable too choose,
  - value — which value to choose,
  - propagation — how to propagate constraints,
  - heuristics — what heuristics can be used.
- CP/CLP basic solution paradigm:
  - select a variable,
  - select a value,
  - propagate constraints,
  - if conflict — backtrack; if unique — return solution; else — loop.

Constraint (Logic) Programming

- Declarative Programming + Multi-Purpose Models:
  - simple, transparent statement; practical problem,
  - zero, one, or many solutions,
  - problem: combinatorial explosion.
- Decision factors in CP/CLP:
  - variable — which variable too choose,
  - value — which value to choose,
  - propagation — how to propagate constraints,
  - heuristics — what heuristics can be used.
- CP/CLP basic solution paradigm:
  - select a variable,
  - select a value,
  - propagate constraints,
  - if conflict — backtrack; if unique — return solution; else — loop.

Constraint (Logic) Programming

- Declarative Programming + Multi-Purpose Models:
  - simple, transparent statement; practical problem,
  - zero, one, or many solutions,
  - problem: combinatorial explosion.
- Decision factors in CP/CLP:
  - variable — which variable too choose,
  - value — which value to choose,
  - propagation — how to propagate constraints,
  - heuristics — what heuristics can be used.
- CP/CLP basic solution paradigm:
  - select a variable,
  - select a value,
  - propagate constraints,
  - if conflict — backtrack; if unique — return solution; else — loop.

Constraint (Logic) Programming

- Declarative Programming + Multi-Purpose Models:
  - simple, transparent statement; practical problem,
  - zero, one, or many solutions,
  - problem: combinatorial explosion.
- Decision factors in CP/CLP:
  - variable — which variable too choose,
  - value — which value to choose,
  - propagation — how to propagate constraints,
  - heuristics — what heuristics can be used.
- CP/CLP basic solution paradigm:
  - select a variable,
  - select a value,
  - propagate constraints,
  - if conflict — backtrack; if unique — return solution; else — loop.

Constraint (Logic) Programming

- Declarative Programming + Multi-Purpose Models:
  - simple, transparent statement; practical problem,
  - zero, one, or many solutions,
  - problem: combinatorial explosion.
- Decision factors in CP/CLP:
  - variable — which variable too choose,
  - value — which value to choose,
  - propagation — how to propagate constraints,
  - heuristics — what heuristics can be used.
- CP/CLP basic solution paradigm:
  - select a variable,
  - select a value,
  - propagate constraints,
  - if conflict — backtrack; if unique — return solution; else — loop.

Constraint (Logic) Programming

- Declarative Programming + Multi-Purpose Models:
  - simple, transparent statement; practical problem,
  - zero, one, or many solutions,
  - problem: combinatorial explosion.
- Decision factors in CP/CLP:
  - variable — which variable too choose,
  - value — which value to choose,
  - propagation — how to propagate constraints,
  - heuristics — what heuristics can be used.
- CP/CLP basic solution paradigm:
  - select a variable,
  - select a value,
  - propagate constraints,
  - if conflict — backtrack; if unique — return solution; else — loop.

Constraint (Logic) Programming

- Declarative Programming + Multi-Purpose Models:
  - simple, transparent statement; practical problem,
  - zero, one, or many solutions,
  - problem: combinatorial explosion.
- Decision factors in CP/CLP:
  - variable — which variable too choose,
  - value — which value to choose,
  - propagation — how to propagate constraints,
  - heuristics — what heuristics can be used.
- CP/CLP basic solution paradigm:
  - select a variable,
  - select a value,
  - propagate constraints,
  - if conflict — backtrack; if unique — return solution; else — loop.

Presentation Outline

1. Introduction: An Example to Start
2. Constraint Satisfaction Problems: Examples
3. **Constraint Satisfaction Problem**
4. Introduction to MiniZinc
5. Example: SEND+MORE=MONEY
6. Constraint Optimization Example
7. Using Data files
8. Constraint Programming with Real Numbers
9. More Complex Structures: Arrays and Sets

Constraint Satisfaction Problem

CSP statement

- $X = \{X_1, X_2, \ldots, X_k\}$ — a set of variables,
- $D = \{D_1, D_2, \ldots, D_k\}$ — their domains,
- $C = \{(S_i, R_i) : i = 1, 2, \ldots, n\}$ — constraints,
  - S_i — scope — a selection of variables,
  - R_i — relation defined over Cartesian Product of domains appropriate for the scope variables,

CSP solution

A solution to CSP given by (X, D, C) is any assignment of values to variables of X of the form

$$\{X_1 = d_1, X_2 = d_2, \ldots, X_k = d_k\},$$

such that $d_i \in D_i$, and for any constraint in $(S_i, R_i) \in C$, R_i is satisfied by the appropriate projection of the solution vector $(d_1, d_2, \ldots, d_k)$ over variables of S_i.

CP/CSP — Characteristics; CSP vs. OPT

CP vs. OPT

- CSP: first solution counts,
- OPT:best solution counts.

Binary vs. finite domains; SAT

- SAT: binary domains (0 or 1; true or false),
- CSP: finite discrete domains.

CSP: Problems

- large number of variables,
- large domains,
- numerous constraints,
- different types of constraints,
- unpredictable, irregular, hard to trace.

Presentation Outline

1. Introduction: An Example to Start
2. Constraint Satisfaction Problems: Examples
3. Constraint Satisfaction Problem
4. **Introduction to MiniZinc**
5. Example: SEND+MORE=MONEY
6. Constraint Optimization Example
7. Using Data files
8. Constraint Programming with Real Numbers
9. More Complex Structures: Arrays and Sets

Introduction to MiniZinc

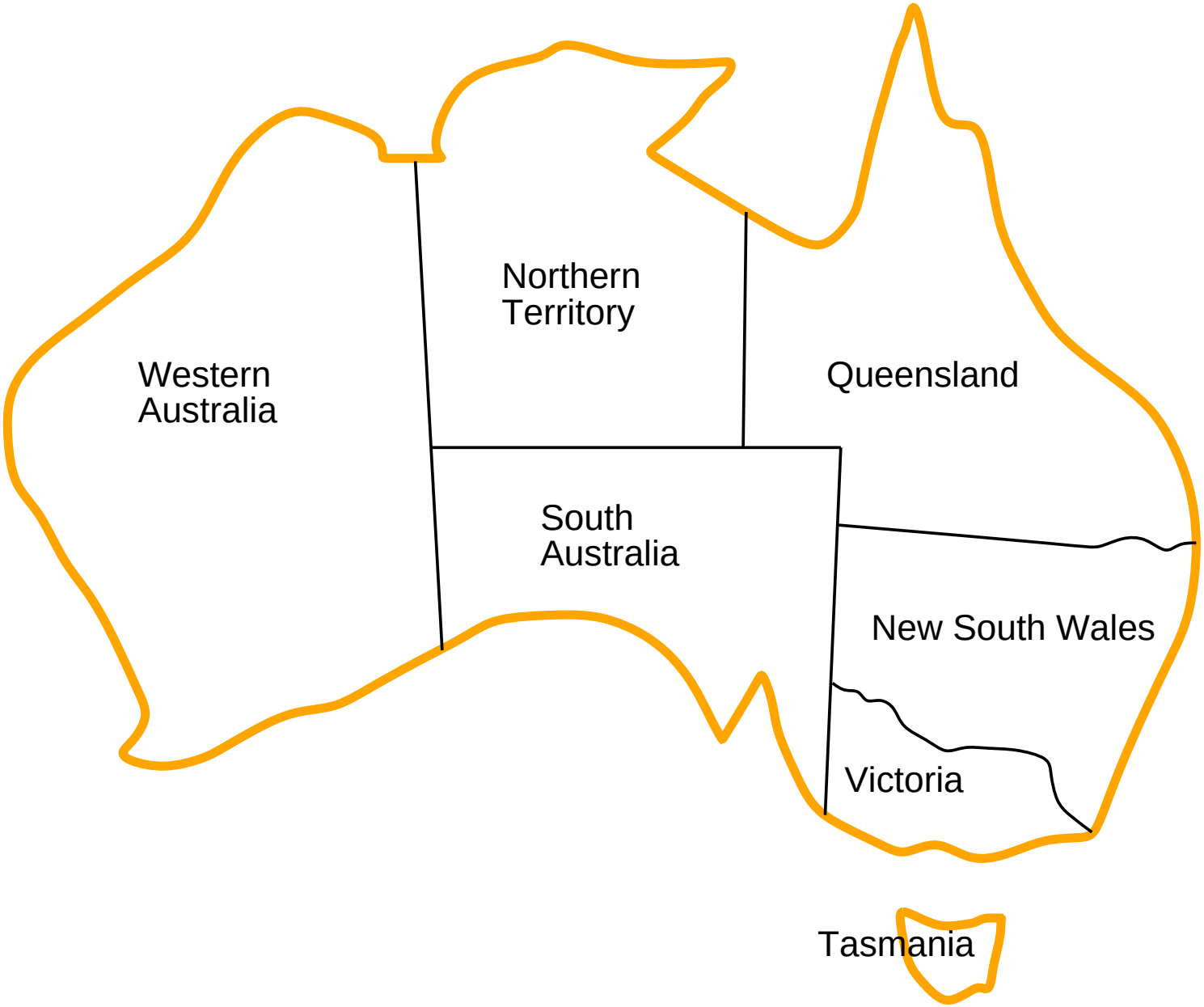
Some important ideas:

- MiniZinc belongs to Declarative Programming Paradigm,
- MiniZinc provides a high=level language for constraint specification,
- the constraints are translated into FlatZinc model,
- the constraints can be processed with several lower-level tools (*backend solvers*; model once, solve everywhere.
- the same MODEL can be processed with several data/goals (*backend solvers*; model once, solve what-you-need.

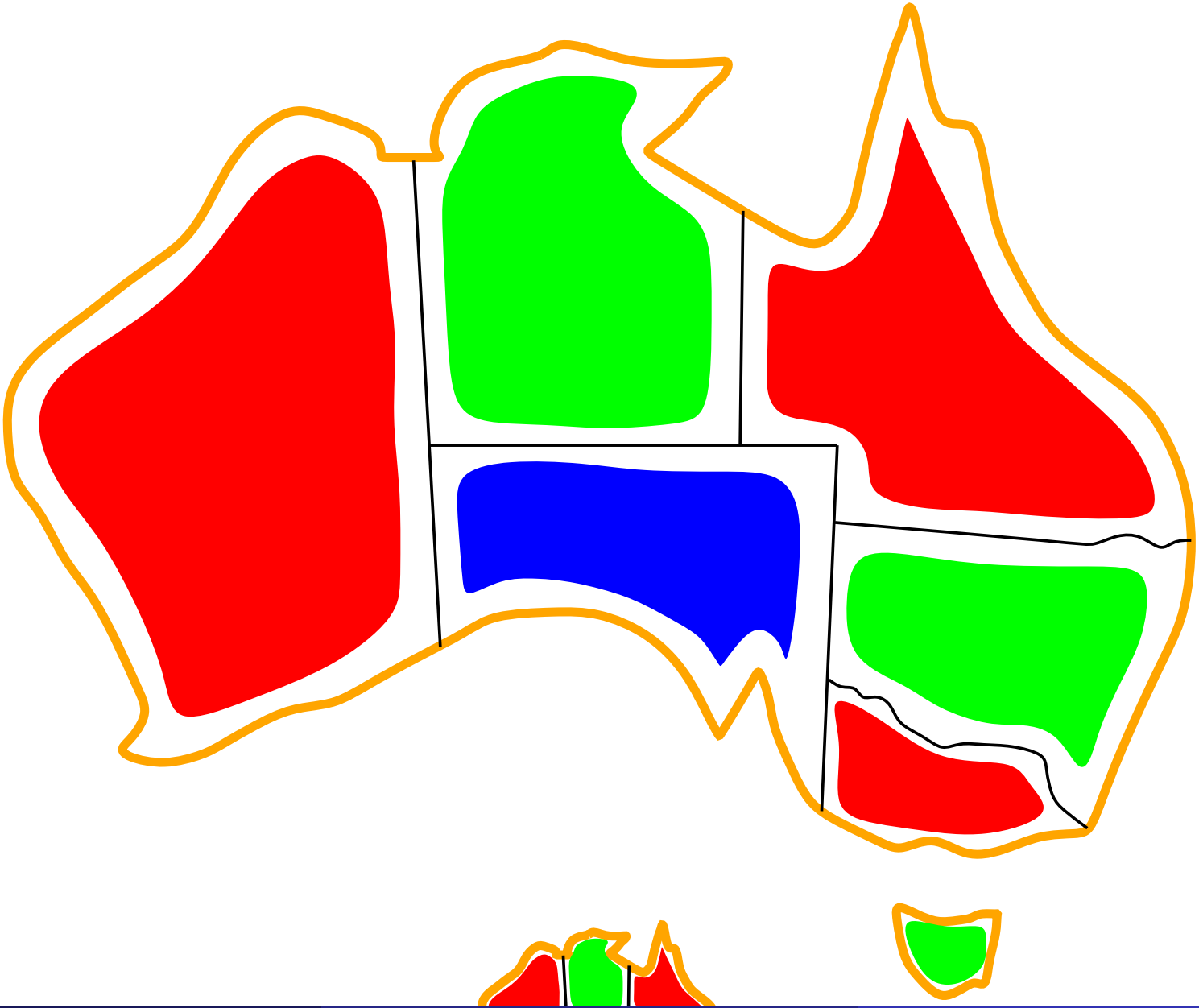
Composition of MiniZinc program:

- **parameters** definition — if any,
- **variables** definition,
- **constraints** definitions,
- **objective function** definition — if any (other than SAT),
- **solve** command and parameters,
- output specification.

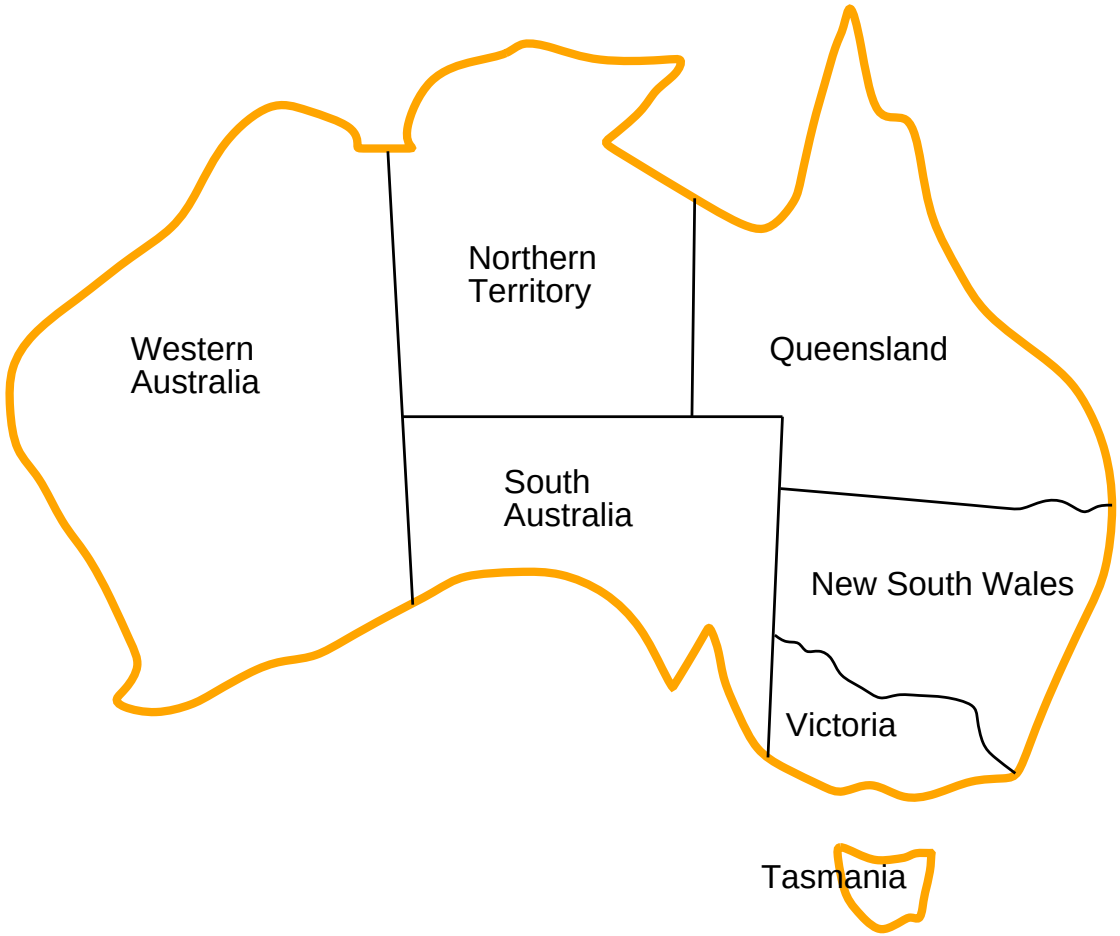
A Map Coloring Problem

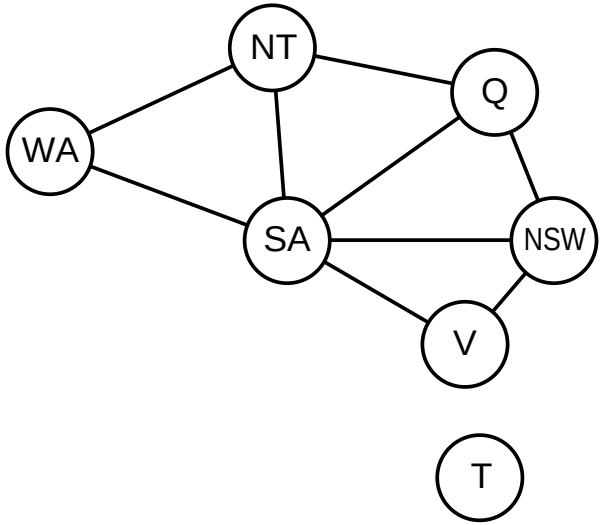


A Map Co-louring Problem Solved



What about Constraints?





A Map Coloring Example I

```
% Colouring Australia using nc colours
int: nc = 3; % a single parameter

% variables
var 1..nc: wa;   var 1..nc: nt;  var 1..nc: sa;   var 1..nc: q;
var 1..nc: nsw;  var 1..nc: v;   var 1..nc: t;

%constraints
constraint wa != nt;
constraint wa != sa;
constraint nt != sa;
constraint nt != q;
constraint sa != q;
constraint sa != nsw;
constraint sa != v;
constraint q != nsw;
constraint nsw != v;
```

A Map Coloring Example II

```
% solution type declaration
solve satisfy;

% output specification
output ["wa=", show(wa), "\t nt=", show(nt),
        "\t sa=", show(sa), "\n", "q=", show(q),
        "\t nsw=", show(nsw), "\t v=", show(v), "\n",
         "t=", show(t),  "\n"];
```

Code Specification: Some Basic Ideas

- a `parameter` — type and value,
- a parameter cannot be changed (but re-specified for a next run),
- parameters can be specified in a separate file, given by hand, or modified **before** compilation,
- supported types: `int`, `float`, `bool`, `string`; **also** `array`, **and** `set` ,
- a `variable` is assigned *domain* (or type),
- variables can be: `bool`, `int`, `float`, `set`,
- arrays of variables are accessible,
- a variable can be *instantiated* with a value of an appropriate type only!
- `constraints` (basic): $=$ ($==$), $>$, $<$, $<=$, $>=$,
- **constraints** are Boolean expressions – *what does this mean?*,
- `solve satisfy;` defines the goal,
- output specification (long strings can be split over lines with the ++ for concatenation).

Presentation Outline

1. Introduction: An Example to Start
2. Constraint Satisfaction Problems: Examples
3. Constraint Satisfaction Problem
4. Introduction to MiniZinc
5. **Example: SEND+MORE=MONEY**
6. Constraint Optimization Example
7. Using Data files
8. Constraint Programming with Real Numbers
9. More Complex Structures: Arrays and Sets

Example: SEND+MORE=MONEY

Figure : An example Constraint Satisfaction Problem

- 8 variables: `S, E, N, D, M, O, R, Y`,
- 10-values in each domain,
- alldifferent(S, E, N, D, M, O, R, Y),
- basic search-space size: 10^8,
- reduced search-space size:
  $9 * 8 * 7 * 6 * 5 * 4 * 3 * 2 = 362880$

A simple CLP code I

```
sendmoremoney(Vars) :-
    Vars = [S,E,N,D,M,O,R,Y],
    Vars ins 0..9,
    S #\= 0,
    M #\= 0,
    all_different(Vars),
                 1000*S + 100*E + 10*N + D
    +            1000*M + 100*O + 10*R + E
    #= 10000*M + 1000*O + 100*N + 10*E + Y.

solve(Vars):- Vars=[S,E,N,D,M,O,R,Y],
              sendmoremoney([S,E,N,D,M,O,R,Y]),
              label(Vars).
```

Pretty good results! I

```
?- time(sendmoremoney(V)).
% 6,758 inferences, 0.00 CPU in 0.00 seconds (0% CPU, Infinite Lips
V = [9, _G11470, _G11473, _G11476, 1, 0, _G11485, _G11488],
_G11470 in 4..7,
all_different([_G11470, _G11473, _G11476, _G11485, _G11488, 0, 1, 9
1000*9+91*_G11470+ -90*_G11473+_G11476+ -9000*1+ -900*0+10*_G11485+
_G11473 in 5..8,
_G11476 in 2..8,
_G11485 in 2..8,
_G11488 in 2..8.

?- time(solve(V)).
% 10,088 inferences, 0.01 CPU in 0.00 seconds (308% CPU, 1008800 Li
V = [9, 5, 6, 7, 1, 0, 8, 2].
```

MiniZinc Coding I

```
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]);
```

MiniZinc Coding II

```
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"];
```

Presentation Outline

1. Introduction: An Example to Start
2. Constraint Satisfaction Problems: Examples
3. Constraint Satisfaction Problem
4. Introduction to MiniZinc
5. Example: SEND+MORE=MONEY
6. **Constraint Optimization Example**
7. Using Data files
8. Constraint Programming with Real Numbers
9. More Complex Structures: Arrays and Sets

Integer Programming

Two types of cakes

- two types of cakes: b = banana, c = chocolate; b,c - output variables,
- each of them uses specific amount of limited resources,
- each of them provides some profit,
- the goal is to maximize the profit.

MiniZinc Coding: Banana & Chocolate Cakes I

```
% Baking cakes for the school fete

var 0..100: b; % no. of banana cakes
var 0..100: c; % no. of chocolate cakes

% flour
constraint 250*b + 200*c <= 4000;
% bananas
constraint 2*b  <= 6;
% sugar
constraint 75*b + 150*c <= 2000;
% butter
constraint 100*b + 150*c <= 500;
% cocoa
constraint 75*c <= 500;

% maximize our profit
```

MiniZinc Coding: Banana & Chocolate Cakes II

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

output ["no. of banana cakes = ", show(b), "\n",
        "no. of chocolate cakes = ", show(c), "\n"];
```

Presentation Outline

1. Introduction: An Example to Start
2. Constraint Satisfaction Problems: Examples
3. Constraint Satisfaction Problem
4. Introduction to MiniZinc
5. Example: SEND+MORE=MONEY
6. Constraint Optimization Example
7. **Using Data files**
8. Constraint Programming with Real Numbers
9. More Complex Structures: Arrays and Sets

Using Data files

In order to change the parameters of the model it is convenient to specify them in a data-file (.dzn).

- a data-file contains a set of pre-declared parameters,
- there can be several files with different data,
- hence, the same model can be re-used in an easy way,
- it is reasonable to check the imported parameters,
- a check is done with the `assert(<condition>,<output>)` predicate (this is called *Defensive Programming*),
- `assert` acts as `Boolean` expression.

Example MiniZinc Code with Data-file I

```
% Baking cakes for the school fete (with data file)

int: flour;  %no. grams of flour available
int: banana; %no. of bananas available
int: sugar;  %no. grams of sugar available
int: butter; %no. grams of butter available
int: cocoa;  %no. grams of cocoa available

constraint assert(flour >= 0,"Invalid datafile: " ++
                  "Amount of flour is non-negative");
constraint assert(banana >= 0,"Invalid datafile: " ++
                  "Amount of banana is non-negative");
constraint assert(sugar >= 0,"Invalid datafile: " ++
                  "Amount of sugar is non-negative");
constraint assert(butter >= 0,"Invalid datafile: " ++
                  "Amount of butter is non-negative");
constraint assert(cocoa >= 0,"Invalid datafile: " ++
```

Example MiniZinc Code with Data-file II

```
                    "Amount of cocoa is non-negative");

var 0..100: b; % no. of banana cakes
var 0..100: c; % no. of chocolate cakes

% flour
constraint 250*b + 200*c <= flour;
% bananas
constraint 2*b  <= banana;
% sugar
constraint 75*b + 150*c <= sugar;
% butter
constraint 100*b + 150*c <= butter;
% cocoa
constraint 75*c <= cocoa;

% maximize our profit
solve maximize 400*b + 450*c;
```

Example MiniZinc Code with Data-file III

```
output ["no. of banana cakes = ", show(b), "\n",
        "no. of chocolate cakes = ", show(c), "\n"];
```

Presentation Outline

1. Introduction: An Example to Start
2. Constraint Satisfaction Problems: Examples
3. Constraint Satisfaction Problem
4. Introduction to MiniZinc
5. Example: SEND+MORE=MONEY
6. Constraint Optimization Example
7. Using Data files
8. **Constraint Programming with Real Numbers**
9. More Complex Structures: Arrays and Sets

Model with Real Numbers

Some ideas concerning Constraint Programming and Optimization with real numbers (floats).

- the properties of the model change drastically,
- it may be necessary to use different solver! (in our case G12 MIP)
- if analytic model is accessible - try it!
- Linear Programming and Simplex may be a solution,
- Mixed Integer-Linear Programming models are hard,
- the same model can be used for answering different questions:

A Loan

P - amount borrowed, I – interest rate, R – rate (4 rates), B – balance

- given I, P, and R (rate), how much is the final balance?
- given I, P, and ensuring B4=0 (0 balance), what should be the rates?
- given I, R, and ensuring B4=0 (0 balance), how much can I borrow (P)?

- the model does not change; only the input parameters.

MiniZinc Coding – a Loan Example I

```
% 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;
```

MiniZinc Coding – a Loan Example II

```
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"
];
```

Presentation Outline

1. Introduction: An Example to Start
2. Constraint Satisfaction Problems: Examples
3. Constraint Satisfaction Problem
4. Introduction to MiniZinc
5. Example: SEND+MORE=MONEY
6. Constraint Optimization Example
7. Using Data files
8. Constraint Programming with Real Numbers
9. More Complex Structures: Arrays and Sets

Why Arrays and Sets?

- the number of variables can change with the size of the problem:
  - e.g. number of products (array: quantity of product),
  - e.g. number of rates to be paid (array: amount of rate),
  - e.g. number of components/blocks (array: component value),
- **array** can be of one, two, and many dimensions,
- notation `temp[3,7]`; a variable associated with the position `(3,7)` on a grid,
- one can define a single constraint over an array – all the variables!
- `array[1..5] = [1,3,5,7,11]` – one-dimensional array,
- `array[1..3,1..4] = [|1,2,3|,|4,5,6|,|7,8,9|,|10,11,12|]` – two-dimensional 4×3 array,
- `constraint forall(i in 1..w-1)(temp[i,h] = right);` – –example constraint over a border,
- list comprehension: special form of list specification, e.g. `forall( [a[i] != a[j] | i,j in 1..3 where i < j])`

Laplace: a Visualization

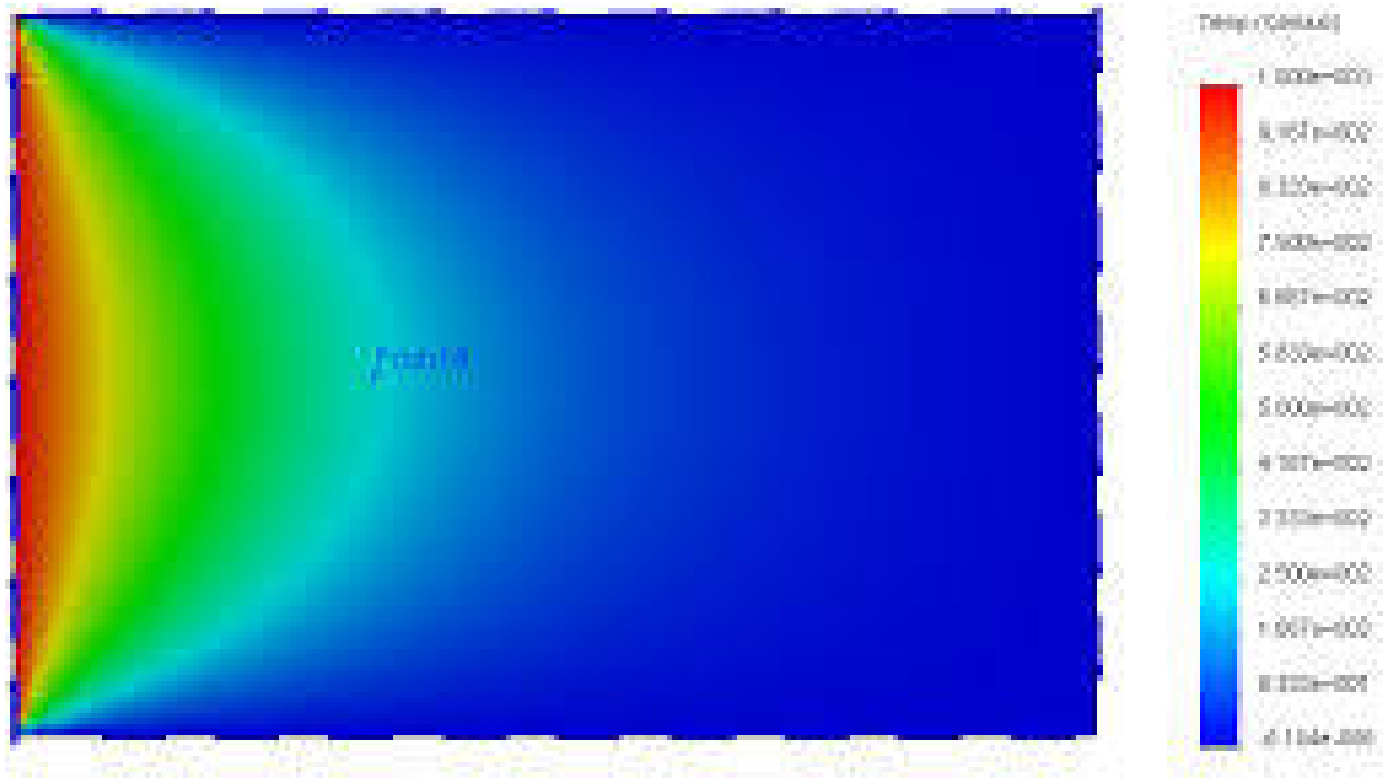


Figure : A Visualization of 2-D Temperature Distribution

MiniZinc Coding: Laplace I

```
int: w = 4;
int: h = 4;

% arraydec
array[0..w,0..h] of var float: t; % temperature at point (i,j)
var float: left;   % left   edge temperature
var float: right;  % right  edge temperature
var float: top;    % top    edge temperature
var float: bottom; % bottom edge temperature

% equation
% Laplace equation: each internal temp.
% is average of its neighbours
constraint forall(i in 1..w-1, j in 1..h-1)(
   4.0*t[i,j] = t[i-1,j] + t[i,j-1] + t[i+1,j] + t[i,j+1]);
% sides
% edge constraints
```

MiniZinc Coding: Laplace II

```
constraint forall(i in 1..w-1)(t[i,0] = left);
constraint forall(i in 1..w-1)(t[i,h] = right);
constraint forall(j in 1..h-1)(t[0,j] = top);
constraint forall(j in 1..h-1)(t[w,j] = bottom);
% corners
% corner constraints
constraint t[0,0]=0.0;
constraint t[0,h]=0.0;
constraint t[w,0]=0.0;
constraint t[w,h]=0.0;
left = 0.0;
right = 0.0;
top = 100.0;
bottom = 0.0;

solve satisfy;

output [ show_float(6, 2, t[i,j]) ++
```

MiniZinc Coding: Laplace III

```
        if j == h then "\n" else " " endif |
        i in 0..w, j in 0..h
];
```