Outline

1. Modelling Viewpoints
2. Implied Constraints
3. Redundant Variables & Channelling Constraints
4. Pre-Computation
5. Over-Constrained Problems
Outline

1. Modelling Viewpoints

2. Implied Constraints

3. Redundant Variables & Channelling Constraints

4. Pre-Computation

5. Over-Constrained Problems
Example (Objects, Shapes, and Colours)

There are $n$ objects, $s$ shapes, and $c$ colours, with $s \geq n$. Assign a shape and a colour to each object such that:

1. the objects have distinct shapes;
2. the numbers of objects of the used colours are distinct;
3. other constraints, yielding NP-hardness, are satisfied.

This problem can be modelled from different viewpoints:

- Which colour, if any, does each shape have?
- Which shapes, if any, does each colour have?
- Which shape and colour does each object have?
- ...

Each viewpoint comes with benefits and drawbacks.
Example (Objects, Shapes, and Colours)

Viewpoint 1: Which colour, if any, does each shape have?

```plaintext
1 int: n; % number of objects
2 int: s; % number of shapes
3 int: c; % number of colours
4 constraint assert(s >= n, "Not enough shapes");

5 array[1..s] of var 0..c: Colour; % 0 is a dummy colour
6
7 % There are n objects:
8 constraint count(Colour, 0) = s-n;
9 % The numbers of objects of the used colours are distinct:
10 constraint
   alldifferent_except_0(global_cardinality(Colour, 1..c));
11 % The objects have distinct shapes:
12 % implied by lines 6 and 9
13 % ... add here the other constraints ...
14 solve satisfy;
```

Colour 0 is used when there is no object of the given shape. So what are the shape and colour of a particular object?! Map the objects onto the shapes with a non-0 colour!
Example (Objects, Shapes, and Colours)

Viewpoint 2: Which shapes, if any, does each colour have?

```plaintext
int: n; % number of objects
int: s; % number of shapes
int: c; % number of colours
constraint assert(s >= n, "Not enough shapes");
array[1..c] of var set of 1..s: Shapes;

% There are n objects:
constraint n = sum(colour in 1..c)(card(Shapes[colour]));
% The numbers of objects of the used colours are distinct:
constraint alldifferent_except_0(colour in 1..c)
   (card(Shapes[colour]));
% The objects have distinct shapes:
constraint n = card(array_union(Shapes));
% ... add here the other constraints ...
solve satisfy;
```

Post-process: map the objects onto actually used shapes.
Can we also model this viewpoint without set variables? Yes, see the next slide!
Example (Objects, Shapes, and Colours)

Viewpoint 2: Which shapes, if any, does each colour have?

```plaintext
1 int: n; % number of objects
2 int: s; % number of shapes
3 int: c; % number of colours
4 constraint assert(s >= n, "Not enough shapes");

5 array[1..c,1..s] of var 0..1: NbrObjects;

% There are n objects:
9 constraint n = sum(colour in 1..c, shape in 1..s) (NbrObjects[colour,shape]);

% The numbers of objects of the used colours are distinct:
11 constraint alldifferent_except_0(colour in 1..c) (sum(shape in 1..s)(NbrObjects[colour,shape]));

% The objects have distinct shapes:
13 constraint forall(shape in 1..s)(1 >= sum(colour in 1..c) (NbrObjects[colour,shape]));

% ... add here the other constraints ...
15 solve satisfy;
```

Which model for viewpoint 2 is clearer or better? Ask & try!
Example (Objects, Shapes, and Colours)

Viewpoint 3: Which shape & colour does each object have?

1  int: n; % number of objects
2  int: s; % number of shapes
3  int: c; % number of colours
4  constraint assert(s >= n, "Not enough shapes");
5
6  array[1..n] of var 1..s: Shape;
7  array[1..n] of var 1..c: Colour;
8
9  % There are n objects:
10  % implied by lines 6 and 7
11  % The numbers of objects of the used colours are distinct:
12  constraint
13      alldifferent_except_0(global_cardinality(Colour,1..c));
14  % The objects have distinct shapes:
15  constraint alldifferent(Shape);
16  % ... add here the other constraints ...
17  solve satisfy;

We have used two parallel arrays with the same index set but different domains in order to represent pair variables.
Outline

1. Modelling Viewpoints

2. Implied Constraints

3. Redundant Variables & Channelling Constraints

4. Pre-Computation

5. Over-Constrained Problems
Example (The Magic Series Problem)

The element at index $i$ in $I = 0..(n-1)$ is the number of occurrences of $i$. Solution: $\text{Magic} = [1, 2, 1, 0]$ for $n=4$.

**Variables:** $\text{Magic} = 0 \ 1 \ \cdots \ n-1$

$\in 0..n \ 0..n \ \cdots \ 0..n$

**Constraint:**

forall$(i \in I)(\text{Magic}[i] = \sum (j \in I)(\text{bool2int}(\text{Magic}[j]=i)))$

or, logically equivalently:

global_cardinality_closed(Magic, I, Magic)

**Implied Constraint:**

$\sum (\text{Magic})=n \ \lor \ \sum (i \in I)(\text{Magic}[i] \times i)=n$

For $n=80$, using a CP solver: only 7 search nodes are explored instead of 302; the solving is 1,000 times faster.
Definition

An implied constraint, also called a redundant constraint, is a constraint that logically follows from other constraints.

Benefit:
Solving may be faster, without losing any solutions. However, not all implied constraints accelerate the solving.

Good practice in MiniZinc:
Flag implied constraints using the `implied_constraint` predicate. This allows backends to handle them differently, if wanted (see Topic 9: Modelling for CBLS):

```
predicate implied_constraint(var bool: c) = c;  // VS
predicate implied_constraint(var bool: c) = true;
```

Example

```
constraint implied_constraint(sum(Magic) = n);
```

In Topic 7: Symmetry, we will see the equally recommended `symmetry_breaking_constraint` predicate.
Outline

1. Modelling Viewpoints
2. Implied Constraints
3. Redundant Variables & Channelling Constraints
4. Pre-Computation
5. Over-Constrained Problems
Redundant Decision Variables

Example (n-queens)

Use both the $n^2$ decision variables $Queen[i, j]$ in 0..1 and the $n$ decision variables $Row[q]$ in 1..n.

Definition

A redundant decision variable is a decision variable that represents information that is already represented by some other decision variables. It reflects a different viewpoint.

Benefit: Easier modelling, or faster solving, or both.

Examples (see Topic 6: Case Studies)

- Model of Black-Hole Patience
- Models 1 & 3 of Warehouse Location Problem
Channelling Constraints

Example (n-queens)

Channelling between the n decision variables Row[i] in 1..n and the n^2 decision variables Queen[i,j] in 0..1:

forall(i in 1..n)(Row[i] = sum(j in 1..n)(j * Queen[i,j]))

Definition

A channelling constraint establishes the coherence of the values of mutually redundant decision variables.

Examples (see Topic 6: Case Studies)

- Model of Black-Hole Patience
- Models 1 & 3 of Warehouse Location Problem
- Experiment with channelling between the viewpoints for the Objects, Shapes, and Colours problem (slide 4).
Outline

1. Modelling Viewpoints
2. Implied Constraints
3. Redundant Variables & Channelling Constraints
4. Pre-Computation
5. Over-Constrained Problems
Example (Prize-Pool Division)

Consider a maximisation problem where the objective function is the division of an unknown prize pool by an unknown number of winners:

```plaintext
... 
array[1..5] of int: Pools = [1000,5000,15000,20000,25000];
var 1..5: x;
var 1..500: numWinners;
...
solve maximize Pools[x] div numWinners; % implicit: element!
```

**Observation:** We should avoid using the `div` function on decision variables, because:

- It yields weak inference, at least in CP & LCG solvers.
- Its inference takes unnecessary time and memory.
- It is not supported by all backends.

**Idea:** We can pre-compute all possible objective values.
Idea: We can pre-compute all possible objective values.

Example (Prize-Pool Division, revisited)

Pre-compute a 2D array, indexed by $1..5$ and $1..500$, for each possible value pair of $x$ and $\text{numWinners}$:

```plaintext
... 
array[1..5] of int: Pools = [1000,5000,15000,20000,25000];
var 1..5: x;
var 1..500: numWinners;
...
array[1..5,1..500] of int: objFun = array2d(1..5,1..500, 
    [Pools[p] div n | p in 1..5, n in 1..500]);
solve maximize objFun[x,numWinners]; % implicit: 2D-element!
```
Outline

1. Modelling Viewpoints
2. Implied Constraints
3. Redundant Variables & Channelling Constraints
4. Pre-Computation
5. Over-Constrained Problems
Over-Constrained Problems

Definition

A problem is over-constrained when its correct formalisation leads to an inconsistent model (that is, with no solutions).

Remedies, unless the problem cannot be modified:

- Drop some constraints.
- Relax the requirement that all constraints be satisfied:
  - Keep some of the constraints as hard constraints, which must be satisfied.
  - Consider the other constraints as soft constraints, which need not be satisfied.
Soft Constraints

Implementation:
Define the possibly additional objective function regarding the soft constraints.

- For example: Maximise the number of the actually satisfied soft constraints. This can be done by reifying the soft constraints and maximising the sum of the introduced Boolean decision variables. See the Photo Problem in Topic 2: Basic Modelling.

- More generally: Assign numeric utilities to the soft constraints, and maximise the sum of the utilities of the actually satisfied soft constraints.