Outline

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

1. Modelling Viewpoints
2. Implied Constraints
3. Redundant Variables & Channelling Constraints
4. Pre-Computation
Recap

**1 Modelling**: express problem in terms of
- Parameters,
- Decision variables,
- Constraints, and
- Objective.

**2 Solving**: solve with existing state-of-the-art solvers.
Example (Student Seating Problem)

Given:

- $n$ students, and
- $m$ chairs positioned around tables.

What are suitable decision variables for this problem?

A choice of decision variables represents a viewpoint.
Example (Student Seating Problem)

Given:

- \( n \) students, and
- \( m \) chairs positioned around tables.

Find a seating arrangement such that:

- Each table is either at least half full or empty.
- Each table has at least as many students as any table behind it.
- A maximum number of student neighbour preferences are satisfied.

What are suitable decision variables for this problem?

A choice of decision variables represents a viewpoint.

\( n = 15 \) students
\( m = 20 \) chairs
A viewpoint is a choice of decision variables.

**Example (Student Seating Problem)**

**Viewpoint 1:**
For each student, which chair is the student assigned to?
- Define variables $S_1, S_2, \ldots, S_{15} \in \{1, \ldots, 20\}$.
- A solution where $S_5 = 7$ represents that student 5 is sitting on chair 7.

**Viewpoint 2:**
For each chair, which student, if any, is sitting on it?
- Define variables $C_1, C_2, \ldots, C_{20} \in \{0, \ldots, 15\}$.
- A solution where $C_7 = 5$ represents that student 5 is sitting on chair 7, and $C_7 = 0$ that chair 7 is empty.

Let’s look at a generic problem to see how viewpoints differ when we start formulating constraints.
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 and distinguishing objects and shapes, 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
6 array[1..s] of var 0..c: Colour; % 0 is a dummy colour
7
8 % There are n objects:
9 constraint count(Colour, 0) = s-n;
10 % The numbers of objects of the used colours are distinct:
11 constraint
12     alldifferent_except_0 (global_cardinality(Colour, 1..c));
13 % The objects have distinct shapes:
14 % implied by lines 6 and 9
15 % ... add here the other constraints ...
16 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?

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] of var set of 1..s: Shapes;

6 % There are n objects:
7 constraint n = sum(colour in 1..c)(card(Shapes[colour]));
8 % The numbers of objects of the used colours are distinct:
9 constraint alldifferent_except_0(colour in 1..c)
   (card(Shapes[colour]));
10 % The objects have distinct shapes:
11 constraint n = card(array_union(Shapes));
12 % ... add here the other constraints ...
13 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
6 array[1..c,1..s] of var 0..1: NbrObjects;
7
8 % There are n objects:
9 constraint n = sum(colour in 1..c, shape in 1..s) (NbrObjects[colour,shape]);
10 % 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]));
12 % The objects have distinct shapes:
13 constraint forall(shape in 1..s)(1 >=
    sum(colour in 1..c)(NbrObjects[colour,shape]));
14 % ... 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?

```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
6 array[1..n] of var 1..s: Shape;
7 array[1..n] of var 1..c: Colour;
8
% There are n objects:
9 % implied by lines 6 and 7
10 % The numbers of objects of the used colours are distinct:
11 constraint
12   alldifferent_except_0(global_cardinality(Colour,1..c));
13 % The objects have distinct shapes:
14 constraint alldifferent(Shape);
15 % ... add here the other constraints ...
16 solve satisfy;
```

We have used two parallel arrays with the same index set but different domains in order to represent pair variables.
Which viewpoint is better in terms of:

- Search space explored

- Expressing constraints and objective

- Performance

- Readability
Which viewpoint is better in terms of:

- Search space explored
  - Viewpoint 1: $O((c + 1)^s)$
  - Viewpoint 2: $O(2^{s \cdot c})$
  - Viewpoint 3: $O(s^n \cdot c^n)$

- Expressing constraints and objective

- Performance

- Readability
Which viewpoint is better in terms of:

- Search space explored
  - Viewpoint 1: $O((c + 1)^s)$
  - Viewpoint 2: $O(2^{s \cdot c})$
  - Viewpoint 3: $O(s^n \cdot c^n)$

Does this actually matter?

- Expressing constraints and objective

- Performance

- Readability
Which viewpoint is better in terms of:

- Search space explored
  - Viewpoint 1: $O((c + 1)^s)$
  - Viewpoint 2: $O(2^{s \cdot c})$
  - Viewpoint 3: $O(s^n \cdot c^n)$

Does this actually matter?

- Expressing constraints and objective
  - Depends on the unstated constraints.
  - Ideally we want a viewpoint that allows global constraint predicates to be used.

- Performance

- Readability
Which viewpoint is better in terms of:

- **Search space explored**
  - Viewpoint 1: $O((c + 1)^s)$
  - Viewpoint 2: $O(2^{s \cdot c})$
  - Viewpoint 3: $O(s^n \cdot c^n)$

Does this actually matter?

- **Expressing constraints and objective**
  - Depends on the unstated constraints.
  - Ideally we want a viewpoint that allows global constraint predicates to be used.

- **Performance**
  - Hard to tell, we have to run experiments!

- **Readability**
Which viewpoint is better in terms of:

- **Search space explored**
  - Viewpoint 1: $O((c + 1)^s)$
  - Viewpoint 2: $O(2^{s \cdot c})$
  - Viewpoint 3: $O(s^n \cdot c^n)$

Does this actually matter?

- **Expressing constraints and objective**
  - Depends on the unstated constraints.
  - Ideally we want a viewpoint that allows global constraint predicates to be used.

- **Performance**
  - Hard to tell, we have to run experiments!

- **Readability**
  - Who is going to read our model?
  - What is their background?
Which viewpoint is better in terms of:

- Search space explored
  - Viewpoint 1: $O((c + 1)^s)$
  - Viewpoint 2: $O(2^{s \cdot c})$
  - Viewpoint 3: $O(s^n \cdot c^n)$

Does this actually matter?

- Expressing constraints and objective
  - Depends on the unstated constraints.
  - Ideally we want a viewpoint that allows global constraint predicates to be used.

- Performance
  - Hard to tell, we have to run experiments!

- Readability
  - Who is going to read our model?
  - What is their background?

*There is no correct answer here, we need to actually think about this and do experiments.*
Outline

1. Modelling Viewpoints

2. Implied Constraints

3. Redundant Variables & Channelling Constraints

4. Pre-Computation
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:** $\text{Magic} = \begin{bmatrix} 0 & 1 & \cdots & n-1 \\ \in 0..n & \in 0..n & \cdots & \in 0..n \end{bmatrix}$

**Constraint:**

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

or, logically equivalently but better:

forall($i$ in $I$)($\text{count}(\text{Magic}, i, \text{Magic}[i])$)

or, logically equivalently and even better:

$\text{global_cardinality_closed} (\text{Magic}, I, \text{Magic})$

**Implied Constraint:**

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

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):

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

Example

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

In Topic 5: 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
Redundant Decision Variables

Example (n-queens)

Use both the \( n^2 \) decision variables \( \text{Queen}[i, j] \text{ in } 0..1 \) and the \( n \) decision variables \( \text{Row}[q] \text{ in } 1..n \).

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 of some constraints, 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 7).
Outline

1. Modelling Viewpoints
2. Implied Constraints
3. Redundant Variables & Channelling Constraints
4. Pre-Computation
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 MiniZinc 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!
```