Outline

1. Motivation
2. alldifferent
3. nvalue
4. global_cardinality
5. element
6. bin_packing
7. knapsack
8. cumulative, disjunctive
9. circuit, subcircuit
10. lex_lesseq
11. regular, table
Outline

1. Motivation
2. alldifferent
3. nvalue
4. global_cardinality
5. element
6. bin_packing
7. knapsack
8. cumulative, disjunctive
9. circuit, subcircuit
10. lex_lesseq
11. regular, table
Examples

Let \( A \) be a 1D array of variables, say with indices in \( 1..n \): 

- The \texttt{alldifferent} \((A)\) constraint holds if and only if (iff) all the elements of \( A \) take different values: 
  \[
  \forall (i, j \in 1..n \text{ where } i < j)(A[i] \neq A[j]).
  \]

- The \texttt{at_least} \((m, A, v)\) constraint holds iff at least \( m \) elements of \( A \) take the value \( v \), where \( m \) is an integer: 
  \[
  m \leq \sum_{i \in 1..n} \text{bool2int}(A[i] = v).
  \]

The \texttt{count_leq} \((A, v, m)\) constraint has the same semantics, but \( v \) and \( m \) can even be variables.

Definition

A \textbf{definition} of a constraint predicate is its semantics, stated in MiniZinc in terms of usually simpler constraint predicates. Each use of a predicate is \textbf{decomposed} during flattening by inlining either its MiniZinc-provided default definition or an overriding backend-provided solver-specific definition.
Motivation:

+ More compact and intuitive models, because more expressive predicates are available: islands of common combinatorial structure are identified in declarative medium-level abstractions.

+ Faster solving, due to better inference and relaxation, enabled by more global information in the model, provided the predicate is a built-in of the used solver.

Enabling constraint-based modelling:

- Constraint predicates over any number of variables go by many names: global-constraint predicates, combinatorial-constraint predicates, ...

Outline

1. Motivation
2. alldifferent
3. nvalue
4. global_cardinality
5. element
6. bin_packing
7. knapsack
8. cumulative_disjunctive
9. circuit_subcircuit
10. lex_lesseq
11. regular_table
The **alldifferent** Predicate

### Definition (Laurière, 1978)

The **alldifferent** (`A`) constraint, where `A` is a 1D array of variables, say with indices in `1..n`, holds iff all the elements of `A` take different values.

Its default definition is a conjunction of `\frac{n \cdot (n-1)}{2}` disequality constraints:

\[
\forall (i, j \in 1..n \text{ where } i < j) (A[i] \neq A[j])
\]

### Example

- **n-queens problem**: see Topic 1: Introduction.
- **Photo problem**: see Topic 2: Basic Modelling.
Outline

1. Motivation
2. alldifferent
3. nvalue
4. global_cardinality
5. element
6. bin_packing
7. knapsack
8. cumulative, disjunctive
9. circuit, subcircuit
10. lex_lesseq
11. regular, table
The nvalue Predicate

Definition (Pachet and Roy, 1999)

The \texttt{nvalue}(m,A) constraint holds iff variable m takes the number of distinct values taken by the elements of the 1D array A of variables, say with indices in \(1 \ldots n\):

\[|\{A[1], \ldots, A[n]\}| = m\]

The \texttt{nvalue}(A) expression denotes the number of distinct values taken by the elements of the 1D array A of variables.

Note: \texttt{alldifferent}(A) iff \texttt{nvalue}(n,A), when A has size n: always use the most specific available predicate!

Example

Model 2 of the Warehouse Location Problem:

see Topic 6: Case Studies.
Outline

1. Motivation
2. alldifferent
3. nvalue
4. **global_cardinality**
5. element
6. **bin_packing**
7. knapsack
8. cumulative, disjunctive
9. circuit, subcircuit
10. **lex_lesseq**
11. **regular, table**
The **global_cardinality** Predicate

**Definition (Régis, 1996)**

The **global_cardinality** \((A, V, C)\) constraint holds iff each variable \(C[j]\) has the number of elements of the 1D array \(A\) of variables that take value \(V[j]\). It generalises:

- **alldifferent** \((A)\): set \(\text{dom}(C[j]) = \{0, 1\}\) for all \(j\) and \(V = \bigcup_{i=1}^{n} \text{dom}(A[i])\), if \(A\) has indices in \(1..n\).
- **count_leq** \((A, v, m)\):
  - set \(V = [v]\) and constrain \(m \leq C[1]\).
- **count_gt** \((A, v, m)\):
  - set \(V = [v]\) and constrain \(m > C[1]\).

Other variants exist: always use the most specific predicate!

**Example**

Model of the Magic Series problem: see Topic 4: Modelling.
Outline

1. Motivation
2. alldifferent
3. nvalue
4. global_cardinality
5. element
6. bin_packing
7. knapsack
8. cumulative, disjunctive
circuit, subcircuit
9. circuit, subcircuit
10. lex_lesseq
11. regular, table
The **element** Predicate

### Definition (Van Hentenryck and Carillon, 1988)

The *element*(i, A, e) constraint, where:
- A is an array of decision variables,
- i is an integer decision variable, and
- e is a decision variable,

holds if and only if \( A[i] = e \).

For better model readability, the *element* predicate should not be used, as the functional form \( A[\phi] \) is allowed, even if \( \phi \) is an integer expression involving at least one variable.

Example (Job allocation at minimal salary cost)

Given $n$ jobs and the salaries of $w$ work applicants, find a work applicant for each job such that some constraints (on the qualifications of the work applicants for the jobs, on workload distribution, etc) are satisfied and the total salary cost is minimal:

```plaintext
1 array[1..w] of int: Salary;
2 array[1..n] of var 1..w: Worker; % job j by Worker[j]
3 solve minimize sum(j in 1..n)(Salary[Worker[j]]);
4 constraint ...; % qualifications, workload, etc
```

We do not know at modelling time the worker of each job!
Outline

1. Motivation
2. alldifferent
3. nvalue
4. global_cardinality
5. element
6. bin_packing
7. knapsack
8. cumulative, disjunctive
9. circuit, subcircuit
10. lex_lesseq
11. regular, table
The `bin_packing` Predicate

**Definition**

Let object \( O_i \) have the given weight or volume \( V[i] \). Let variable \( B[i] \) denote the bin into which \( O_i \) is put. The `bin_packing(u, B, V)` constraint holds iff the sum of the volumes of the objects put into each bin is at most \( u \); there are \(|\bigcup_i \text{dom}(B[i])| \) bins. Variant predicates exist.

**Example (Balanced academic curriculum problem)**

Given \( n \) courses \( C_i \), each with a set \( P[i] \) of prerequisite courses and a workload \( W[i] \), find in which semester \( S[i] \) of \( 1..m \) each course \( C_i \) can be offered in order to satisfy all prerequisites under a completely balanced workload:

1. `bin_packing(sum(W) div m, S, W);`
2. `forall(i in 1..n, j in P[i])(S[i] > S[j]);`
Outline

1. Motivation
2. alldifferent
3. nvalue
4. global_cardinality
5. element
6. bin_packing
7. knapsack
8. cumulative, disjunctive
9. circuit, subcircuit
10. lex_leq
11. regular, table
The **knapsack** Predicate

### Definition

Let object $O_i$ have the given weight or volume $V[i]$. Let object $O_i$ have the given value or profit $P[i]$. Let the decision variable $X[i]$ denote the number of copies of $O_i$ that are put into a given knapsack. Let the decision variables $v$ and $p$ respectively denote the total volume and total profit of the knapsack. Given $n$ objects, the `knapsack(V, P, X, v, p)` constraint holds iff $\sum(i \text{ in } 1..n) (V[i] \times X[i]) = v$ and $\sum(i \text{ in } 1..n) (P[i] \times X[i]) = p$.

### Example

To model the Knapsack Problem for a knapsack of given capacity $c$, add $v \leq c$ and maximize $p$. 

---

Motivation

- alldifferent
- nvalue
- global_cardinality
- element
- bin_packing
- knapsack
- cumulative, disjunctive
- circuit, subcircuit
- lex_lesseq
- regular, table

M4CO
Example ([http://xkcd.com/287](http://xkcd.com/287))

A simplified version of the Knapsack Problem, but still NP-complete.

```plaintext
1 array[1..6] of int: Cost = [215, 275, 335, 355, 420, 580];
2 array[1..6] of int: Profit = [0, 0, 0, 0, 0, 0];
3 array[1..6] of var 0..10: Amount;
4 constraint knapsack(Cost, Profit, Amount, 1505, 0);
5 solve satisfy;
```

See this [interview](http://xkcd.com/287) for some interesting trivia.
Outline

1. Motivation
2. alldifferent
3. nvalue
4. global_cardinality
5. element
6. bin_packing
7. knapsack
8. cumulative, disjunctive
9. circuit, subcircuit
10. lex_lesseq
11. regular, table
Assume we want to schedule a set of tasks to be performed over a given period such that we have the **earliest** end.

**Definition**

A task $T_i$ is a triple $\langle S[i], D[i], R[i] \rangle$ of parameters or decision variables, where:

- $S[i]$ is the starting time of task $T_i$
- $D[i]$ is the duration of task $T_i$
- $R[i]$ is the quantity of a global resource needed by $T_i$

Tasks may be run in parallel if the global resource suffices.

![Sample schedule with parallel tasks and bounded resource](image)
**Definition**

A precedence constraint of task $T_1$ on task $T_2$ expresses that the performing of $T_1$ must finish before $T_2$ can start. We say that task $T_1$ precedes task $T_2$.

**Example (courtesy Magnus Ågren)**

Sample tasks (bubbles), durations (black numbers), resource requirements (blue numbers), and precedences (orange arrows). Task T7 is a dummy task, as we do not know which of tasks T5 and T6 will finish last.
Let us temporarily ignore the bounded global resource: If we have an unlimited global resource or each task has its own local resource, then the polynomial-time-solvable problem of finding the earliest ending time, under only the precedence constraints, for performing all the tasks can be modelled using linear inequalities.

Example (continued)

The precedence constraints indicated by the orange arrows on slide 22 are modelled as follows, based on the task durations indicated there in black:

1. constraint \( D = [2,1,4,2,3,1,0] \);
6. % add here the resource constraints of the next slide
7. solve minimize \( S[7] \);
The **cumulative** Predicate

**Definition (Aggoun and Beldiceanu, 1993)**

The `cumulative(S, D, R, u)` constraint, where each task $T_i$ has a starting time $S[i]$, a duration $D[i]$, and a resource requirement $R[i]$, ensures that the resource upper limit $u$ is never exceeded when performing the $T_i$.

`cumulative` does not ensure any precedence constraints between the tasks: these have to be stated separately.

**Example (end)**

To ensure that the global resource capacity of $u = 8$ units, say, is never exceeded under the resource requirements of the tasks indicated in blue on slide 22, add the following:

1. `constraint R = [1, 3, 3, 2, 4, 6, 0];`
2. `constraint cumulative(S, D, R, 8);`
The **disjunctive** Predicate

**Definition**

A non-overlap constraint between tasks $T_1$ and $T_2$ states that either $T_1$ precedes $T_2$ or $T_2$ precedes $T_1$, say because both tasks require a resource that is available only for one task at a time. We say that tasks $T_1$ and $T_2$ do not overlap.

**Definition (Carlier, 1982)**

The **disjunctive** $(S,D)$ constraint, where each task $T_i$ has a starting time $S[i]$ and a duration $D[i]$, ensures that no tasks $T_i$ and $T_j$ overlap. It has the following definitions:

- **forall** $(i, j \text{ in } 1..n \text{ where } i<j)$
  $$((S[i]+D[i]<=S[j]) \lor (S[j]+D[j]<=S[i]))$$

- **cumulative** $(S, D, [1 | i \text{ in } 1..n], 1)$

Always use the most specific available constraint predicate!
Outline

1. Motivation
2. alldifferent
3. nvalue
4. global_cardinality
5. element
6. bin_packing
7. knapsack
8. cumulative, disjunctive
9. circuit, subcircuit
10. lex_lesseq
11. regular, table
The **circuit** and **subcircuit** Predicates

Enabling the representation of a circuit in a digraph:

- Each vertex is mapped to an integer, say from 1 to n.
- Let variable $S[i]$ represent the successor of vertex $i$.
- The domain of $S[i]$ is the set of indices $j$ such that there is an arc from vertex $i$ to vertex $j$, plus $i$ itself.

**Definition (Lauri`ere, 1978)**

The circuit$(S)$ constraint holds iff the $n$ arcs $i \rightarrow S[i]$ form a Hamiltonian circuit: each vertex is visited once.

**Definition (Beldiceanu and Contejean, 1994)**

The subcircuit$(S)$ constraint holds iff circuit$(S')$ holds for exactly one possibly empty non-singleton subarray $S'$ of $S$, and $S[i] = i$ for all the other vertices.
### Example (Solutions and non-solutions)

Assume the successor variables in $S$ take these values:

- $[2, 3, 4, 1]$: circuit $1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 1$.
- $[1, 2, 3, 4]$: empty subcircuit, and $S[i]=i$ for all $i$.
- $[3, 4, 1, 2]$: not a solution, as there are two subcircuits, namely $1 \rightarrow 3 \rightarrow 1$ and $2 \rightarrow 4 \rightarrow 2$.

### Usage:

- **Travelling salesperson problem**: Visit all cities exactly once, under the smallest sum of travelled distances:
  ```
  constraint circuit(S);
  solve minimize sum(c in 1..n)(Distance[c,S[c]]);
  ```

- **Vehicle routing problems**

- **Requiring a path from vertex $i$ to vertex $j$ in $1..n$**:
  ```
  constraint subcircuit(S) /\ S[j] = i;
  upon adding $i$ to dom($S[j]$) if need be.
  ```
Outline

1. Motivation
2. alldifferent
3. nvalue
4. global_cardinality
5. element
6. bin_packing
7. knapsack
8. cumulative, disjunctive
9. circuit, subcircuit
10. lex_leq
11. regular, table
**The lex_lesseq Predicate**

**Example**

```
lex_lesseq([1, 2, 34, 5, 678], [1, 2, 36, 45, 78]),
because 34 < 36 among the natural numbers.
```

**Definition**

The `lex_lesseq(A, B)` constraint, where `A` and `B` are same-length 1D arrays of variables, say both with indices in `1..n`, holds iff `A` is lexicographically at most equal to `B`:

- either `n=0`, or `A[1]<B[1]`,

Variant predicates exist.

**Usage:** Exploit index symmetries in matrix models, where there are matrices of decision variables: see Topic 4: Modelling, and see Topic 7: Symmetry.
Outline

1. Motivation
2. alldifferent
3. nvalue
4. global_cardinality
5. element
6. bin_packing
7. knapsack
8. cumulative, disjunctive
9. circuit, subcircuit
10. lex_lesseq
11. regular, table
Regular Expressions

Example (Regular Expressions)

- \((0|1)^*0\) denotes the set of even binary numbers.
- \(1^*(011^*)^*(0|\epsilon)\) denotes the set of strings of zeros and ones without consecutive zeros.
- \((0|1)^*00(0|1)^*\) denotes the set of strings of zeros and ones with consecutive zeros.

Notation for strings:

- Let \(\epsilon\) denote the empty string.
- Let \(v \cdot w\) denote the concatenation of strings \(v\) and \(w\).
- Let \(w^i\) denote the concatenation of \(i\) copies of string \(w\).
Regular Expressions and Languages

Definition

Let $\Sigma$ be an alphabet, that is a finite set of symbols. Regular expressions over $\Sigma$ are defined as follows:

- $\emptyset$ is a regular expression: its language, $L(\emptyset)$, is $\emptyset$.
- $\epsilon$ is a regular expression: $L(\epsilon) = \{\epsilon\}$.
- If $\sigma \in \Sigma$, then $\sigma$ is a regular expression: $L(\sigma) = \{\sigma\}$.
- If $r$ and $s$ are regular expressions, then $rs$ is a regular expression: $L(rs) = \{v \cdot w \mid v \in L(r) \land w \in L(s)\}$.
- If $r$ and $s$ are regular expressions, then $r|s$ is a regular expression: $L(r|s) = L(r) \cup L(s)$.
- If $r$ is a regular expression, then $r^*$ is a regular expression: $L(r^*) = \{w^i \mid i \in \mathbb{N} \land w \in L(r)\}$.

A regular expression defines a regular language over $\Sigma$. 
Common abbreviations for regular expressions:

Let $r$ be a regular expression:

- $[\text{abcd}]$ denotes $a|b|c|d$
- $[\text{b-g}]$ denotes $[\text{bcdefg}]$
- $[\text{a-cA-C01}]$ denotes $[\text{abcABC01}]$
- $r^?$ denotes $r|\epsilon$
- $r^+$ denotes $rr^*$
- $r^4$ denotes $rrrr$

**Usage:** Regular expressions are good for the specification of regular languages, but not so good for reasoning on them, where one often uses finite automata instead.
**Example**

**Conventions:**
- **Start state:** A (marked by arc coming in from nowhere).
- **Accepting states:** D and E (marked by double circles).
- **Determinism:** There is one outgoing arc per symbol in alphabet $\Sigma = \{s, t\}$; missing arcs go to a non-accepting state that has self-loops on every symbol in $\Sigma$. 
The **regular** and **table** Predicates

**Definition (Pesant, 2004)**

The `regular(A, D)` constraint holds iff the values taken by the 1D array `A` of variables form a string that belongs to the regular language accepted by the DFA `D`.

If need be, use the transformation algorithms of automata theory to convert a regular expression into a (minimal) DFA.

**Definition**

The `table(A, T)` constraint holds iff the values taken by the 1D array `A` of variables form a row of the 2D array `T` of values.

The 2D array `T` gives an **extensional definition** of a new constraint predicate, as opposed to the **intensional definition** given so far for all other constraint predicates.
Example (The Nonogram Puzzle)

Each hint gives the sequence of lengths of **blue** blocks in its row or column, with at least one white cell between blocks, but possibly none before the first and after the last block.

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>1</th>
<th>2</th>
<th>2</th>
<th>1</th>
<th>2</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

2 1
Example (The Nonogram Puzzle)

Each hint gives the sequence of lengths of **blue** blocks in its row or column, with at least one white cell between blocks, but possibly none before the first and after the last block.

Solution:
Example (The Nonogram Puzzle, end)

Model:

- Decision variables: An integer variable for each cell, with value 0 if it is coloured white, and value 1 if it is coloured blue.

- Constraints: State a regular constraint for each hint. For example, for hint 2 3 1 on a row or column \( A \) of length \( n \geq 8 \), state regular \( (A, 0^*1^30+1^0)^* \), but replace the regular expression by a DFA for the same language.

See Survey of Paint-by-Number Puzzle Solvers: the straightforward model above fares well, at least with a CP solver, compared to hand-written problem-specific code.
Example (Nurse Rostering)

Each nurse is assigned each day to one of the following:

- n normal shift (this value is not available on Sundays)
- l long shift (this value is not available on Sundays)
- s Sunday shift (this value is only available on Sundays)
- o day off

The nurse labour union imposes the following regulations:

- Monday off after a Sunday shift
- No single long shifts
- One day off after two consecutive long shifts

For each nurse, state the following constraint over the scheduling horizon, say 17 weeks here:

\[
\text{regular}([\text{Sun1, Mon1, ..., Fri17, Sat17}], (s o|l|l|o|n|o)^*)
\]

Further, a hospital has constraints on nurse presence.
Example (The Kakuro Puzzle)

Fill in digits of $1 \ldots 9$ such that the digits of each word are pairwise distinct and add up to the number to the left (for horizontal words) or on top (for vertical words) of the word.
Example (The Kakuro Puzzle)

Fill in digits of 1..9 such that the digits of each word are pairwise distinct and add up to the number to the left (for horizontal words) or on top (for vertical words) of the word.

```
  11  4
  5   2   3
 14  10
  17
  9   5   1   2   3
  6   4   3   1
 10  3   1   4   2
  3   2   1
```
Example (The Kakuro Puzzle, continued)

**Model:**

- Decision variables: An integer variable for each cell, with domain 1..9.
- Constraints: For each hint $A[\alpha] + \cdots + A[\beta] = \sigma$, state `alldifferent(i in \alpha..\beta)(A[i]) \land sum(i in \alpha..\beta)(A[i]) = \sigma`.

**Performance, using a CP solver:**

- 22 × 14 Kakuro with 114 hints: 9638 nodes, 160 s
- 90 × 124 Kakuro with 4558 hints: ? nodes, ? years

**Symptom:** The decomposition may give weak inference: for $x \neq y \land x+y=4$, CP inference gives $x, y$ in 1..3, not noticing that 2 should be pruned from both domains. We may need a custom predicate `alldifferent_sum`. 
Example (The Kakuro Puzzle, continued)

New model: Use the regular or table predicate for the 
alldifferent and sum-based constraints of each hint?

- For the hint $x+y=4$, state regular([x, y], 13|31).
- For the hint $y+z=3$, state regular([y, z], 12|21).
- In MiniZinc, one cannot provide a regular expression, 
  but one must convert it into a (minimised) DFA. If, as 
  above, only concatenation ($rs$) and alternation ($r|s$) are 
  used, then one can also use table instead:

  \[
  \begin{align*}
  \text{table} & ([x, y], [1, 3|3, 1]) \ \text{}/\text{}/ \\
  \text{table} & ([y, z], [1, 2|2, 1]).
  \end{align*}
  \]

- What about the hint $\sum A[\alpha] + \cdots + A[\alpha+8] = 45$? 
  There are $9! = 362,880$ solutions...
Example (The Kakuro Puzzle, end)

**New model (end):**

- For the hint $A[\alpha] + \cdots + A[\alpha+8] = 45$, it suffices to state `alldifferent([A[i]|i in \alpha..\alpha+8])`, as the sum of 9 distinct non-0 digits is necessarily 45.
- For the hint $A[\alpha] + \cdots + A[\alpha+7] = \sigma$, it suffices to state `alldifferent([A[i]|i in \alpha..\alpha+7|+[45-\sigma])`.
- For the hint $A[\alpha] = \sigma$, it suffices to state $A[\alpha] = \sigma$.

Other opportunities for improvement exist.

**New performance, using a CP solver:**

- 22 × 14 Kakuro with 114 hints: 0 search nodes, 28 ms!
- 90 × 124 Kakuro with 4558 hints: 0 nodes, 345 ms!

The Kakuro story is based on material by Christian Schulte.
When to Use These Predicates?

Rapid prototyping of a new constraint predicate:
The regular and table predicates are very useful in the following conjunctive situation:

- A needed constraint predicate $\gamma$ on a 1D array of variables is not a built-in of the used solver.
- A definition of $\gamma$ in terms of built-in predicates is not obvious to the modeller, or it has turned out that its inference is too expensive or weak.
- The modeller does not have the time or skill to design an inference algorithm for $\gamma$, or deems $\gamma$ not reusable.
- The complexity and strength of an inference algorithm for $\gamma$ are not deemed crucial for the time being.
Important Modelling Device

Example (Encoding a small function)

The constraint $x \times x = y$, where there is exactly one $y$ for every $x$, may yield poor inference: for $x$ in 1..6, say, try $element(x, [1, 4, 9, 16, 25, 36], y)$, that is $[1, 4, 9, 16, 25, 36][x] = y$, for better inference.

The element predicate is a specialisation of regular and table, just like a function is a special case of a relation.

Example (Encoding a small relation)

The constraint $x \times x = \text{abs}(y)$, where there can be more than one $y$ for every $x$, and vice-versa, may yield poor inference: for $x$ in 0..3, say, try the less readable $table([x, y], [|0,0|1,-1|1,1|2,-4|2,4|3,-9|3,9|])$ for better inference (maybe not with a MIP solver).