Relational Modelling of CSPs and Tractability

ASTRA Research Group on the
Analysis, Synthesis, and Transformation / Reformulation
of Algorithms in constraint programming

Uppsala University
PierreF@csd.uu.se
http://www.csd.uu.se/~pierref/astra/

Currently sponsored by VR, under grant number 221-99-369
and by STINT, under institutional grant IG2001-67.

Members

- Dr Pierre Flener, Department of Information Technology, principal investigator
- Dr Justin Pearson, Department of Information Technology
- Mr Brahim Hnich, Department of Information Science, PhD student
- Ms Zeynep Kızıltan, Department of Information Science, PhD student
- Mr Simon Wrang, Department of Information Technology, exjobb student
- …

Projects & Sponsors

- OPL ASTRA project (VR): OPL Algorithm Synthesis and TRAnsformation
- OPL CORSA project (STINT): Constraint mOdelling, Reformulation, Solving, and Analysis
- CAMELOT proposal (VR): Constraint Analysis, ModELling, sOlving, and Tractability
- …

Partners

- Pascal van Hentenryck, Brown University, USA
- Toby Walsh, Cork Constraint Computation Centre, Ireland
- Alan Frisch and Ian Miguel, University of York, UK
- Peter Jeavons, Oxford University, UK
- Vincent Moulton, Linnaeus Centre for Bioinformatics, Uppsala University, Sweden
- …

Good Contacts

- Eugene Freuder, Cork Constraint Computation Centre, Ireland
- Edward Tsang, Essex University, UK
- Mark Wallace, IC Park, Imperial College, UK
- …

Grand Aim

Empower a wider range of programmers to unleash the proven benefits of constraint programming (CP) on a wider range of real-life constraint satisfaction problems (CSPs):

- Design of a next-generation **constraint modelling language** — called **ESRA** — for CSPs. Provision of datatypes for high-level concepts such as *relations*, (*multi-*)*sets*, and *sequences*. Formulation of models that are more natural, easier to maintain, and more amenable to analysis.

- Development of a **toolkit** that helps implement an ESRA model into a constraint program (in a current-generation language) that is not unlike what a human expert would have written, such that instances of the problem can be efficiently solved.

- Analysis of **tractability** at the ESRA level, so that properties, such as *symmetry*, can be automatically exploited during the implementation into a constraint program. Indeed, many CSPs are not tractable unless such properties are exploited.

1. Relational Modelling of CSPs

In constraint programming, much more effort has been directed at efficiently *solving* CSPs than at facilitating their *modelling*.

Claims

- A high-level abstract-datatype-based constraint modelling language opens the door to an automatable determination of how to 'best' represent the decision variables of a CSP.
- A first-order relational calculus with sets is a good candidate for such a language, as it gives rise to very natural and easy-to-maintain models of CSPs.

Modelling with Sets and Relations

- Formal Methods:

  Specification languages: B, VDM, Z, …
  Object modelling languages: ALLOY, OCL (for UML), …

- Databases:

  Entity-Relation-Attribute (ERA) data modelling languages.

- Constraints:

  Sets and set expressions: ECL^{i}PS^e, ILOG Solver, Oz, …
  Total functions and functional image: OPL (via matrices), …

Example: The Warehouse Location Problem

A company considers opening warehouses on some candidate locations
in order to supply its existing stores:

- Each candidate warehouse has the same maintenance cost.
- Each candidate warehouse has a supply capacity.
- The supply cost to a store depends on the warehouse.

The *objective* is to determine which warehouses to open,
and which of these warehouses should supply the various stores, such that:

- Each store must be supplied by exactly one opened warehouse.
- Each opened warehouse supplies at most a number of stores equal to its capacity.
- The sum of the actually incurred maintenance costs and supply costs is minimised.

A First OPL Model of the Warehouse Location Problem (published in the OPL book)

`Capacity:`

Berlin	London	Madrid	Paris	Rome
1	4	2	1	3

`SupplyCost:`

	Berlin	London	Madrid	Paris	Rome
Store 1	20	24	11	25	30
…	…	…	…	…	…

`Supply:`

Store 1	Store 2	…	Store 10
$\in$ Whs	$\in$ Whs	…	$\in$ Whs

`OpenWhs` ($\subseteq$ `Whs`):

Berlin	London	Madrid	Paris	Rome
$\in\{0,1\}$	$\in\{0,1\}$	$\in\{0,1\}$	$\in\{0,1\}$	$\in\{0,1\}$

$$\text{minimise} \quad \sum_{s \in \text{Stores}} \text{SupplyCost}[s,\text{Supply}[s]] + \sum_{w \in \text{Whs}} \text{OpenWhs}[w] \cdot \text{MaintCost}$$

$$\text{subject to} \quad \forall s \in \text{Stores} \cdot \text{OpenWhs}[\text{Supply}[s]] = 1$$

$$\text{and} \quad \forall w \in \text{Whs} \cdot \sum_{s \in \text{Stores}} (\text{Supply}[s]{=}w) \leq \text{Capacity}[w]$$

A Second OPL Model of the Warehouse Location Problem

Drop the redundant matrix `OpenWhs` of decision variables as well as the channelling constraint, and reformulate the second term of the cost function as follows:

$$\sum_{\mathtt{w} \in \mathtt{Whs}} \left(\sum_{\mathtt{s} \in \mathtt{Stores}} (\mathtt{Supply[s]=w}) > \mathtt{0} \right) \cdot \mathtt{MaintCost}$$

This model is more efficient, at least on the considered instance data.

Redundancy elimination may thus pay off, but it may just as well be the converse.
But this is hard to guess, as human intuition may be weak here.

Disadvantages of OPL Modelling

– *No* set variables: need for `0/1` - modelling of subsets.

– *No* relation variables: only total-function variables (as matrices of decision variables); hence awkward formulation of constraints referring to their inverses, which are relations.

– Frequent *need* for higher-order constraints (reification, type casting).

Problem Statement Revisited

In more mathematical terms:

- The sought supply relationship is a *total function* (or: *mapping*) from the set of stores into the set of candidate warehouses.
- The set of warehouses to be opened is the *range* of that function.

Design Decisions for a New Modelling Language

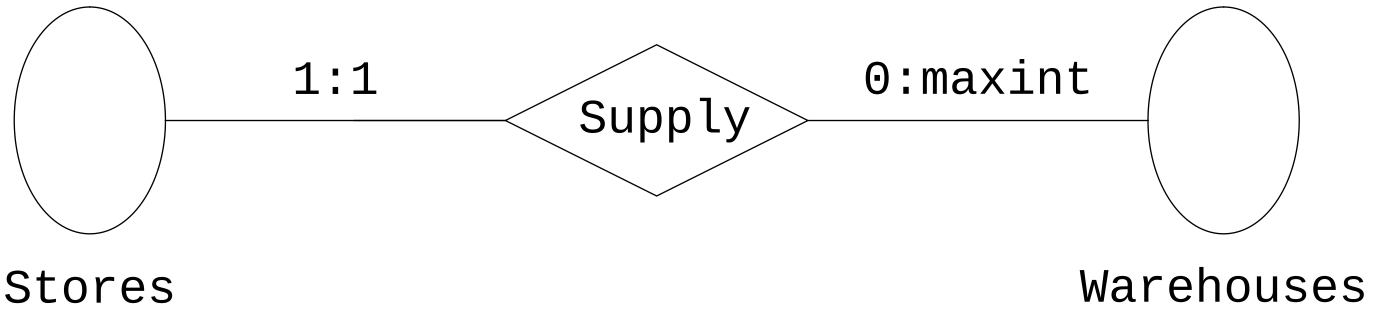
- Capture common modelling idioms in high-level abstract datatypes (ADTs), especially for *sets*, *relations*, fixed / bounded-length *sequences*, and *permutations*, to prevent premature commitment to concrete representations: design of a high-level, next-generation, solver-independent modelling language.
- Computational completeness is *not* aimed at, as long as the language is useful for elegantly modelling a large number of CSPs:

  + *No* higher-order constraints (no reification, no type casting).

  + *No* procedure calls, hence *no* recursion.

  + *Only* bounded quantification.
- Extension of a streamlined significant subset of OPL.

A Relational Model of the Warehouse Location Problem



$$\texttt{minimise} \quad \sum_{(\texttt{s},\texttt{w}) \in \texttt{Supply}} \texttt{SupplyCost[s,w]} + |\texttt{Stores.Supply}| \cdot \texttt{MaintCost}$$

$$\texttt{such that} \quad \forall \texttt{w} \in \texttt{Warehouses} \cdot |\texttt{w.\~{}Supply}| \le \texttt{Capacity[w]}$$

This relational model can be implemented into various CPs, not unlike the OPL models above.

Relational models are more concise as well as easier to develop and read,
as there is a closer conceptual correspondence with the informal statement of the CSP.

Towards making ESRA a fully *declarative* language, no search procedures can be specified.

Time efficiency depends on the implementation of the model as a constraint program.

Premise: A good representation is often better than a good search procedure.

Model Implementation

Occurrences of the high-level ADTs and the constraints thereon can be represented in terms of the classical datatypes in CP, namely integers, lists, matrices, and sets.

This representation relationship is one-to-many.

Example: A total function $f : V \rightarrow W$ can be represented by matrices:

A:

$$\begin{array}{|c|c|c|c|} \multicolumn{1}{c}{v_1} & \multicolumn{1}{c}{v_2} & \multicolumn{1}{c}{\dots} & \multicolumn{1}{c}{v_\mathtt{m}} \\ \hline \in W & \in W & \dots & \in W \\ \hline \end{array}$$

$\mathtt{A[i]=j}$ whenever $f(\mathtt{i}) = \mathtt{j}$

B:

$$\begin{array}{c|c|c|c|c|l} & \multicolumn{1}{c}{w_1} & \multicolumn{1}{c}{w_2} & \multicolumn{1}{c}{\dots} & \multicolumn{1}{c}{w_\mathtt{n}} & \\ \cline{2-5} v_1 & \in\{\mathtt{0,1}\} & \in\{\mathtt{0,1}\} & \dots & \in\{\mathtt{0,1}\} & \sum_{\mathtt{j} \in \mathtt{1:n}} \mathtt{B[1,j]} = \mathtt{1} \\ \cline{2-5} \dots & \dots & \dots & \dots & \dots & \dots \\ \cline{2-5} v_\mathtt{m} & \in\{\mathtt{0,1}\} & \in\{\mathtt{0,1}\} & \dots & \in\{\mathtt{0,1}\} & \sum_{\mathtt{j} \in \mathtt{1:n}} \mathtt{B[m,j]} = \mathtt{1} \\ \cline{2-5} \end{array}$$

$\mathtt{B[i,j]=1}$ whenever $f(\mathtt{i}) = \mathtt{j}$

C:	w_1	w_2	…	w_n	
	$\subseteq V$	$\subseteq V$	…	$\subseteq V$	…

$$\mathtt{i} \in \mathtt{C[j]} \text{ whenever } f(\mathtt{i}) = \mathtt{j}$$

etc.

For different constraints on f,
different representations of f may lead to easier formulation or to better propagation.

Having several representations of f at the same time,
while linking them via *channelling constraints* such as

$$\mathtt{A[i]=j} \Leftrightarrow \mathtt{B[i,j]=1}$$

may pay off, despite the increased numbers of decision variables and constraints.

Expert modellers have a standard “bag of tricks” for high-level concepts,
resulting in recurring low-level idioms that can be captured.

This need *not* result in an efficiency loss while solving.

Efficiency of an Implementation

For a given solver and model,
it is impossible to figure out which is the best implementation,
as this varies with the instances of the problem.

It would be interesting to automatically generate alternative implementations,
also called *reformulations*, and to automatically figure out which one is the most suitable:

- *Heuristic approach*: Based on an analysis of the model constraints and on formalised insights about when a (possibly redundant) representation of a high-level ADT is preferred,
thereby automating current best practice and making it available to the non-expert (and 'lazy').
- *Empirical approach*: Based on training instances, so that the resulting program is only claimed to be the most suitable for instances within the distribution underlying the training instances.
In the extreme case where no training instances are provided,
such a system would be non-deterministic, leaving the final choice to the human.

Premise: Compilation time is irrelevant and will be recovered through many instance runs.

Very little has been done on automatic reformulation,
but we advocate that a high-level notation such as ESRA is a suitable level to do so.

We are developing a compiler that combines the two approaches.

Model Maintenance

- *Problem change*: maintenance is much easier with a high-level modelling language, as the model is not encumbered by ADT representation details and efficiency-related reformulations. Indeed, these issues are only dealt with during compilation into an implementation.
- *Instance distribution change*: maintenance is avoided by the envisaged compiler, namely by providing a new set of training instances.

All model maintenance will be assisted by re-compilation.

Model Analysis

Constraint models in a high-level language are more amenable to analysis, so that detected properties of the problem can be exploited during compilation (see Section 3).

2. Tractability

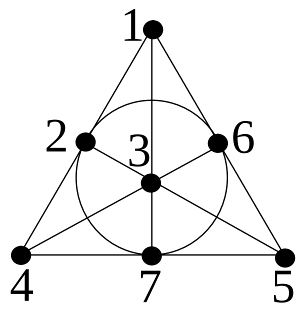
Symmetry-Breaking Constraints

Origin of Symmetries

Symmetries are often introduced *while formulating* a constraint program
from an informal statement of a CSP.

Example: Steiner Triple Systems

A *Steiner triple system* consists of a set X of points and a set B of 3-element subsets B_i of X, called *triples*, such that every pair of distinct elements of X appears in exactly one triple B_i of B.



Existing symmetries:

- Any permutation of X gives rise to a symmetry.

A matrix model and a solution for $|X| = |B| = 7$:

	B_1	B_2	B_3	B_4	B_5	B_6	B_7
Element 1	1	2	3	4	5	6	7
Element 2	2	3	4	5	6	7	1
Element 3	4	5	6	7	1	2	3

Introduced symmetries:

- The elements of each triple B_i are indistinguishable.
- The triples B_i are indistinguishable (column symmetry).

In our example, this introduces $3!^7 \cdot 7! = 1{,}410{,}877{,}440$ symmetries…

Two assignments are *equivalent* if some symmetry takes one to the other.

In each equivalence class (called *orbit* in group theory) of assignments,
it suffices to enumerate only one member during search.
This may make a problem “more tractable.”

Detection of Symmetries

Detecting symmetries is graph-isomorphism complete in general.

However, in a relational model of a CSP, many symmetries are absent,
and are only introduced during the compilation into a lower-level implementation,
such as the matrix model above.

Reduction of Symmetries

Breaking symmetries is also hard:
a super-exponential number of lexicographic ordering constraints is often necessary.

It does *not* always suffice to impose lexicographic ordering constraints on the rows *and* columns,
as many of the symmetries result from a composition of the row and column symmetries:

$$\begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 0 & 0 \end{bmatrix} \xleftrightarrow[\texttt{swap columns 1 \& 2}]{\texttt{swap rows 2 \& 3}} \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix} \xleftrightarrow[\texttt{swap columns 2 \& 3}]{\texttt{swap rows 1 \& 2}} \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 1 & 0 \end{bmatrix}$$

It is also possible to break symmetries *during* search.

Implied Constraints

Constraint propagation may be improved by adding logically redundant constraints, called *implied constraints* in the literature.

Example: Magic Series

`Series:`	0	1	$\ldots$	$n-1$
	$\in 0..n$	$\in 0..n$	$\ldots$	$\in 0..n$

$$\forall \mathtt{i} \in \mathtt{Range} \cdot \mathtt{Series[i]} = \sum_{\mathtt{j} \in \mathtt{Range}} (\mathtt{Series[j]=i})$$

$$\sum_{\mathtt{i} \in \mathtt{Range}} \mathtt{Series[i]} = \mathtt{n}$$

$$\sum_{\mathtt{i} \in \mathtt{Range}} \mathtt{Series[i]} \cdot \mathtt{i} = \mathtt{n}$$

Adding implied constraints is usually done *manually*.

Automatically adding implied constraints is rather straightforward,
with off-the-shelf automated theorem proving technology,
so the objective is to detect the ones leading to improved propagation.

3. Conclusion

Summary

The compilation of an ADT-based constraint modelling language can be based on:

- The few standard ways of possibly redundantly representing high-level concepts (relations, …).
- Theoretically or empirically gained insights about when to deploy each representation.
- Training instances of the problem.

Model formulation and model maintenance then become much easier,
because not encumbered by early — if not uninformed — commitments to representation choices.

A suitable first-order relational calculus with sets is a good candidate for such a modelling language.

Related Work

ALICE (Laurière), NP-SPEC (Cadoli *et al*.), OPL (Van Hentenryck), …:
Quest for a practical modelling language based on a strongly-typed first-order logic
(with the look of an imperative language), while dispensing with such hard-to-properly-implement
and rarely-necessary (in CP) 'luxuries' as recursion and unbounded quantification.

Our ESRA goes beyond, by advocating an abstract view of sets and relations:
the chosen representation of each decision variable of a high-level ADT depends on the constraints.