The Rules of Constraint Modelling: An Overview

Alan M. Frisch and **Chris Jefferson** and **Bernadette Martínez Hernández**
Artificial Intelligence Group, Dept. of Computer Science, Univ. of York, UK
{frisch,caj,berna}@cs.york.ac.uk

Ian Miguel
School of Computer Science, Univ. of St Andrews, UK
ianm@dcs.st-and.ac.uk

Abstract

We address a major bottleneck in the use of constraint programming: modelling. Our system CONJURE automatically *refines* a specification of a problem in our abstract constraint specification language, ESSENCE, into a set of alternative constraint programs, thus automating an important part of the modelling process and helping to reduce the modelling bottleneck significantly.

Introduction

To employ constraint programming to solve a problem, it first must be *modelled*, by a set of constraints on decision variables that solutions must satisfy. Modelling is difficult and requires expertise, thus limiting widespread use of constraint technology. The vast majority of research on constraint modelling presents alternative models to a particular problem and evaluates them through analysis and/or experiment. The process by which the alternative models are generated is rarely discussed. Each constraint programmer must learn the art of modelling by forming generalisations from these studies.

We show that a set of rules can formalise the generation of alternative models. This requires a language in which to express the abstract problem structure of which models are a function. If all modelling choices are to be open to the rules, the language must have a level of abstraction above that at which modelling decisions are made. We have designed such a language and call it ESSENCE. The rules are embedded in the CONJURE system, which, given an ESSENCE specification, generates models of the type supported by existing constraint solvers. Hence, we refer to our rules as *refinement rules*. Our current focus is on generating a set of correct models that includes those that a human expert would generate. In future, we will focus on generating only *good* models.

A central task in modelling most combinatorial problems is choosing a representation of complex decision variables. Current finite-domain constraint solvers provide decision variables whose domains contain atomic elements. Furthermore, other work has extended constraint languages to support sets [5], multisets [7], functions [6] and relations [2] — but these are all on atomic objects. Combinatorial problems often require finding a more complex combinatorial structure. For example, the Sonet fibre-optic communication problem (see Fig. 1) requires finding a set of sets of nodes, subject to communication demand constraints. Modelling the Sonet problem requires deciding how to represent this decision variable as a constrained collection of atomic decision variables.

A Sonet communication network comprises a number of rings, each joining a number of nodes. A node is installed on a ring using an ADM and there is a capacity bound on the number of nodes that can be installed on a ring. Each node can be installed on more than one ring. Communication can be routed between a pair of nodes only if both are installed on a common ring. Given the capacity bound and a specification of which pairs of nodes must communicate, allocate a set of nodes to each ring so that the given communication demands are met. The objective is to minimise the number of ADMs used. (This is a common simplification of the full Sonet problem, as described in [3])

`given`	*nrings*:`nat`, *nnodes*:`nat`, *capacity*:`nat`
`letting`	*Nodes* be 1..*nnodes*
`given`	*demand*:`set (size` m`) of set (size` 2`) of` *Nodes*
`find`	*rings*: `mset (size` *nrings*`) of set (maxsize` *capacity*`) of` *Nodes*
`minimising`	$\sum_{r \in rings} \lvert r \rvert$
`such that`	$\forall pair \in demand.\ \exists r \in rings\ .\ pair \subseteq r$

Figure 1: Specification of the Sonet problem.

In concert with choosing a representation of complex decision variables is the task of representing the constraints of the problem. In its natural form, a combinatorial problem imposes constraints on the combinatorial structure that is sought. These constraints must be "translated" so that they are imposed on the representation of the decision variables. Our results show how these two central tasks of modelling can be formalised and automated, and in doing do so makes two principal contributions.

Contribution 1: ESSENCE

As our first contribution, we have designed a language, called ESSENCE, that enables combinatorial problems to be stated at a high level of abstraction. This level of abstraction is a consequence of three features *unique among constraint languages*. (1) The language supports a wide range of types (including sets, multisets, relations, functions, partitions) and decision variables can be of these types. (2) All types can be nested to arbitrary depth; for example, a decision variable can be of type set, set of sets, set of set of sets, and so forth. (3) Constraints can contain quantifiers that range over decision variables. For example, if a decision variable X is of type set of sets, a constraint can be of the form $\forall x \in X.\phi$.

As an illustrative example, Fig. 1 also presents an ESSENCE specification of the Sonet problem. Notice that the decision variable (denoted by the `find` keyword) is of type set of sets. Further, note the quantification over this decision variable in both the objective function and constraint.

Contribution 2: CONJURE

Our second major contribution is the formulation and automation of a set of rules that can refine constraints on complex variables in an ESSENCE specification into constraints on atomic and atomic set variables, the level of abstraction provided by existing constraint toolkits. To continue our example, Figure 2 gives two of the alternative models that CONJURE outputs for the Sonet specification. Initially we have restricted our attention to sets and multisets in order to focus on formulating the rules correctly. Having done so, we expect the range of types accommodated by the refinement rules to be easy to extend.

In formulating the refinement rules, we have overcome two primary difficulties and many secondary ones. The first difficulty arises because expressions, particularly decision variables of non-atomic type, can usually, if not always, be refined in multiple ways. Furthermore, the refinement of an operator depends on how its operands are refined. The second major difficulty arises from arbitrary nesting of types. Suppose that A and B are sets of some deeply nested type and we wish to refine the constraint $A = B$. Such a constraint would involve all components of both A and B. Furthermore, we wish to produce refinements in which A and B do not have the same kind of representation.

```
given       nrings:nat, nnodes:nat, capacity:nat
letting     Nodes be 1..nnodes
given       demand′:matrix (indexed by 1..m, 1..2) of Nodes
find        rings′:matrix (indexed by 1..nrings, Nodes) of bool
minimising  sum_{j:1..nrings}(rings′[j])
such that   ∀j:1..nrings.(sum(rings′[j]) ≤ capacity)
            ∀j:1..m. ∃k:1..nrings. ∀a:1..2. rings′[k, demand′[j, a]]
```

```
given       nrings:nat, nnodes:nat, capacity:nat
letting     Nodes be 1..nnodes
given       demand′:matrix (indexed by 1..m, 1..2) of Nodes
find        rings′:matrix (indexed by 1..nrings, 1..capacity) of Nodes
            Switches:matrix (indexed by 1..nrings, 1..capacity) of bool
minimising  sum_{j:1..nrings}(Switches[j])
such that   ∀j:1..m ∃k:1..nrings ∀a:1..2 ∃b:1..capacity
              demand′[j, a] = rings′[k, b]) ∧ Switches[k, b]
            ∀j:1..m  ∀a:1..capacity  ∀b:1..a-1
              Switches[j,a] ∧ Switches[j,b] → rings′[j,a] ≠ rings′[j,b]
```

Figure 2: Two models of the Sonet specification.

Modelling in constraints involves more than just representing decision variables and problem constraints. Constraint models often contain many symmetries, often enormous numbers of them, which result in redundancies in the search space. Expert modellers are able to identify such symmetries and break them, either by introducing symmetry-breaking constraints or using a symmetry-aware search method. Another technique used by expert modellers is to represent a complex decision variable with multiple representations simultaneously and impose *channelling* constraints [1] to keep the representations consistent. This sometimes yields more propagation, and therefore less search, than a single representation. We have identified how channelling and symmetry detection can be supported by and introduced into our architecture for model generation.

We have identified 7 problems in the literature thus far that are specified in terms of nested sets and multisets. CONJURE has generated models for all of these, including such well known combinatorial problems as Balanced Incomplete Block Designs ([4] problem 28), the Social Golfers Problem ([4] problem 10), Steiner Triple Systems ([4] problem 44), and the Sonet problem itself.

Conclusion

Our results have made two major contributions towards realising the possibility that model generation can be formalised and automated. A rigorous account of model generation would be valuable in several ways. It could make our study of modelling more systematic, revealing gaps in our understanding. It could also guide the study of model selection by identifying the decision points and the set of alternatives available at each. Furthermore, it would be useful in teaching and presenting modelling and in constructing a catalogue of modelling constructs.

The automation of model generation offers further opportunities. CONJURE currently uses its rules to generate all possible models; but the rules could also be used within an interactive system, which, like an interactive theorem prover, allows the user to chose from among the alternatives available at a decision point. Our ultimate dream is that the automation of model generation takes us one step closer to automating the entire modelling process.

Further information on ESSENCE and CONJURE can be found at www.cs.york.ac.uk/aig/constraints/AutoModel/.

References

[1] B. M. W. Cheng, Jimmy Ho-Man Lee, and J. C. K. Wu. Speeding up constraint propagation by redundant modeling. *Proc. 2nd Int. Conf. on Princ. & Pract. of CP*, *LNCS* 1118, 91–103, 1996.

[2] P. Flener, J. Pearson, M. Agren. Introducing ESRA, a relational language for modelling combinatorial problems. *Proc. LOPSTR'03: Revised Selected Papers*, 214–232, LNCS 3018, 2004.

[3] A.M. Frisch, B. Hnich, I. Miguel, B.M. Smith, and T. Walsh. Transforming and refining abstract constraint specifications. *Proc. CSCLP04: ERCIM/Colognet Workshop on Constraint Solving & Constraint Logic Programming*, 2004.

[4] I. P. Gent and T. Walsh. CSPLib: A benchmark library for constraints. Technical Report APES-09-1999, APES, 1999.

[5] Carmen Gervet. Conjunto: constraint logic programming with finite set domains. *Proc. Int. Symposium in Logic Programming*, 339–358. The MIT press, 1994.

[6] B. Hnich. *Function Variables for Constraint Programming*. PhD thesis, Uppsala University, Dept. Information Science, 2003.

[7] Toby Walsh. Consistency and propagation with multiset constraints: A formal viewpoint. *Proc. 9th Int. Conf. on Princ. & Pract. of CP*, LNCS 2833, 2003.