An Automated Constraint Modelling and Solving Toolchain

Ozgur Akgun[1] Alan M. Frisch[2] Ian Gent[1] Bilal Syed Hussain[1]
Christopher Jefferson[1] Lars Kotthoff[3] Ian Miguel[1] Peter Nightingale[1]

[1] University of St Andrews
{ozgur.akgun,Ian.Gent,bh246,caj21,ijm,pwn1}@st-andrews.ac.uk
[2] University of York alan.frisch@york.ac.uk
[3] University College Cork larsko@4c.ucc.ie

1 Introduction

Constraint Programming (CP) is a powerful technique for solving a wide range of combinatorial problems. However, non-experts have difficulty in formulating the good constraint models of problems necessary for a constraint solver to obtain solutions effectively. Hence, it is desirable to automate constraint modelling. Our approach, embodied in our CONJURE system, is to refine constraint models from their abstract specifications in our ESSENCE constraint specification language. This allows the user to specify a problem in terms of familiar concepts such as sets, functions and relations, without making detailed modelling decisions.

This paper presents two improvements to CONJURE that provide further progress towards the goal of fully automated constraint modelling. The first "closes the loop", mapping solutions to constraint models back to the ESSENCE specifications from which the models are refined. This allows a user to work solely at the ESSENCE specification level, rather than taking the models CONJURE produces, manually inputting them to a constraint solver, and interpreting the results. The second is a heuristic to select among the many possible models CONJURE can typically produce from a given ESSENCE specification. As noted, the model chosen has a significant impact on subsequent solver performance and model selection is difficult for a non-expert, so our model selection heuristic, although preliminary, removes a significant burden from the user. Together, these two advances enable us to provide a first complete automated constraint modelling and solving toolkit.

2 Background

Solving a problem using CP proceeds in two steps. First, the problem is modelled as a set of decision variables, and a set of constraints on those variables that a solution must satisfy. A decision variable represents a choice that must be made in order to solve the problem. The domain of potential values associated with each decision variable corresponds to the options for that choice. The second step consists of using a constraint solver to find solutions to the model: assignments of values to decision variables satisfying all constraints.

Typically, there are many possible models of a given problem, and the model chosen has a substantial impact on the subsequent performance of a constraint solver. Without help, it is very difficult for a novice user to formulate an effective (or even correct) model of a given problem. This is considered to be one of the key challenges facing the field [7] and drives research in automated constraint modelling. This challenge has received a considerable amount of attention in the literature recently [6], where a variety of approaches have been taken to automate aspects of constraint modelling, including: machine learning, case-based reasoning, theorem proving, automated transformation of medium-level solver-independent constraint models[5, 8], and refinement of abstract constraint specifications [1, 3] in languages such as ESSENCE [2], and Zinc [4].

```
language Essence 1.3

given k, n : int(1..)
letting D be domain int(1..n*k)
letting R be domain int(1..n)

find seq: function(total, surjective) D → R

such that
 forAll m : R .
  exists f : function (total, injective) int(1..k) → D .
   (forAll i : D . i in range(f) ↔ i in preImage(seq,m))
    /\
   (forAll j : int(1..k-1) . f(j+1) - f(j) = m + 1)
```

Figure 1: ESSENCE Spec: Langford's Number Problem

Our approach to this problem is to provide the user with an abstract constraint specification language, ESSENCE, in which a problem can be described above the level at which modelling decisions are made. ESSENCE supports abstract decision variables that correspond to familiar concepts such as sets, multisets, functions and relations. Furthermore, it supports the arbitrary nesting of these objects, such as sets of sets of functions, which allows very concise problem specification. An example is given in Figure 1.

Since existing constraint solvers do not support these abstract decision variables directly, abstract constraint specifications must be refined into concrete constraint models. In addition, once solutions are obtained to the refined models, they must be mapped back onto the original ESSENCE specification for presentation to the user. This is the responsibility of the rest of our toolchain, which is described below.

3 An Automated Constraint Modelling and Solving Toolchain

Our toolchain is summarised in Figure 2[1]. An ESSENCE specification is input to our CONJURE system, which em-

[1] These tools are available for download.
CONJURE: http://conjure.cs.st-andrews.ac.uk
SAVILEROW: http://savilerow.cs.st-andrews.ac.uk
MINION: http://minion.sourceforge.net

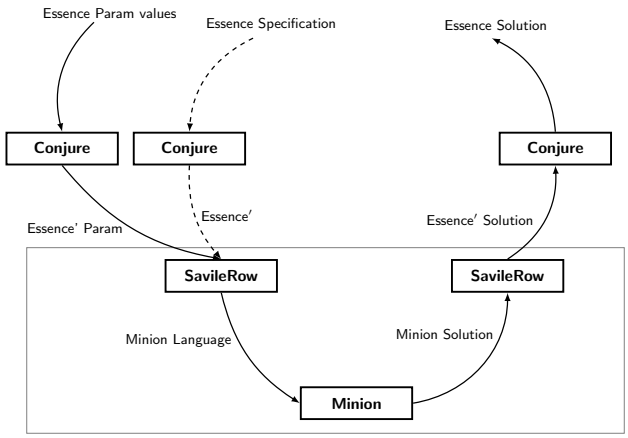

Figure 2: Our Automated Constraint Modelling Toolkit

ploys a system of rules to refine the specification into a constraint model in the ESSENCE′ language, a solver-independent constraint modelling language that is a subset of ESSENCE. A constraint model typically describes a parameterised problem *class*. An instance of the class is obtained for input to the constraint solver by giving values for the parameters. ESSENCE provides the same facility, allowing the specification of problem classes. The refinement of problem specification and parameter values is separated in our toolchain, as shown in the figure. This allows the user to solve multiple instances from the same problem class while only performing model refinement once.

The SAVILEROW system accepts an ESSENCE′ model and corresponding parameter values. It instantiates the model and transforms it into the input suitable for the MINION constraint solver (SAVILEROW is able to produce output suitable for other constraint solvers, but this is beyond the scope of this abstract). After MINION has solved the problem instance, SAVILEROW translates the solution back into ESSENCE′. CONJURE then translates the ESSENCE′ solution into a solution to the original ESSENCE problem for presentation to the user.

CONJURE can generate multiple models for a given problem, and is able to produce all or a random selection of them as required. Selecting a good model from the available candidates remains a challenging task. We have developed a simple heuristic, which works reasonably well in practice. It works by always choosing the *smallest* candidate each time there are multiple alternative reformulations during automated modelling. This decision is local, hence it is subject to the usual pitfalls of local search techniques. CONJURE can pick an alternative which is smaller at the time of one transformation, but a later transformation can cause a great increase in the size of the generated model. Moreover, CP models that are smaller in size aren't necessarily *better*. Despite these shortcomings, the "smallest" heuristic provides a starting point for further research on model selection.

4 Case Study: Langford's Problem

We use Langford's Number Problem[2] as an illustrative example. The ESSENCE specification in Figure 1 uses a decision variable with a *function* domain to succinctly represent the permutation in the problem. The domain of this function variable is further constrained by the use of attributes: *total* and *surjective*. These attributes have similar meanings to their meanings in discrete mathematics.

CONJURE can model this problem in several different ways. One important source of variety is choosing a representation for each abstract domain in the original problem specification. For this example, either a one-dimensional matrix containing integer decision variables, or a two-dimensional matrix containing boolean decision variables can be used to model the decision variable *seq*. Both options require posting additional constraints to satisfy the invariants required by the original domain. Furthermore, each expression involving *seq* needs to be reformulated depending on this choice. The "smallest" heuristic will pick the one-dimensional model because it generates a *smaller* domain and fewer additional constraints.

5 Future Work

We have given an overview of a complete automated constraint modelling and solving toolchain, which allows a user to specify a problem, and receive solutions, in familiar terms without having to make difficult constraint modelling decisions. A principal item of future work is in developing more sophisticated methods of model selection, both for individual models and to construct model portfolios to be solved in a modern multi-core setting.

References

[1] Ozgur Akgun, Ian Miguel, Chris Jefferson, Alan Frisch, and Brahim Hnich. Extensible automated constraint modelling. In *AAAI Conference on AI*, 2011.

[2] Alan M. Frisch, Warwick Harvey, Chris Jefferson, Bernadette Martínez-Hernández, and Ian Miguel. Essence: A constraint language for specifying combinatorial problems. *Constraints 13(3)*, pages 268–306, 2008.

[3] Alan M. Frisch, Christopher Jefferson, Bernadette Martínez Hernández, and Ian Miguel. The rules of constraint modelling. In *IJCAI-05, Proc of the Nineteenth Int Joint Conf on AI, Scotland*, pages 109–116, 2005.

[4] Kim Marriott, Nicholas Nethercote, Reza Rafeh, Peter J. Stuckey, Maria Garcia de la Banda, and Mark Wallace. The design of the zinc modelling language. *Constraints*, 2008.

[5] N. Nethercote, P. J. Stuckey, R. Becket, S. Brand, G. J. Duck, and G. Tack. Minizinc: Towards a standard CP modelling language. In *Proc. of CP 2007*, pages 529–543, 2007.

[6] Barry O'Sullivan. Automated modelling and solving in constraint programming. In *Proceedings of the Twenty-Fourth AAAI Conference on AI, USA*. AAAI Press, 2010.

[7] Jean-Francois Puget. Constraint programming next challenge: Simplicity of use. In *Principles and Practice of Constraint Programming - CP 2004*, pages 5–8, 2004.

[8] Andrea Rendl. Thesis: Effective compilation of constraint models. 2010.

[2] $L(k, n)$ is problem the of ordering k sets of the numbers $1...n$ such that for each pair of the number i is i places apart.