CONJURE: Automatic Generation of Constraint Models from Problem Specifications (Extended Abstract)*

Özgür Akgün1, Alan M. Frisch2, Ian P. Gent1, Christopher Jefferson1, Ian Miguel1, Peter Nightingale2

1School of Computer Science, University of St Andrews, UK
2Department of Computer Science, University of York, UK
{ozg.akgun, ian.gent, caj21, ijm}@st-andrews.ac.uk, {alan.frisch, peter.nightingale}@york.ac.uk

Abstract

When solving a combinatorial problem, the formulation or model of the problem is critical to the efficiency of the solver. Automating the modelling process has long been of interest given the expertise and time required to develop an effective model of a particular problem. We describe a method to automatically produce constraint models from a problem specification written in the abstract constraint specification language ESSENCE. Our approach is to incrementally refine the specification into a concrete model by applying a chosen refinement rule at each step. Any non-trivial specification may be refined in multiple ways, creating a diverse space of models to choose from.

The handling of symmetries is a particularly important aspect of automated modelling. We show how modelling symmetries may be broken automatically as they enter a model during refinement, removing the need for an expensive symmetry detection step following model formulation.

Our approach is implemented in a system called CONJURE. We compare the models produced by CONJURE to constraint models from the literature that are known to be effective. Our empirical results confirm that CONJURE can reproduce successfully the kernels of the constraint models of 42 benchmark problems found in the literature.

1 Introduction

Efficient decision-making is of central importance to a modern society, and it is natural to represent and reason about such decision-making problems in terms of constraints. Constraint programming [Rossi et al., 2006] offers a means by which solutions to such problems can be found automatically. Constraint solving of a given problem proceeds in two phases.

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.

There are typically many possible models for a given problem, and the model chosen can dramatically affect the efficiency of constraint solving. This presents a serious obstacle for non-expert users, who have difficulty in formulating a good (or even correct) model from among the many possible alternatives. Modelling is therefore a critical bottleneck in the process of constraint solving, considered to be one of the key challenges facing the constraints field [Freuder, 2018].

This paper presents the automated constraint modelling system CONJURE, which serves to demonstrate the efficacy of the refinement-based approach. A problem is input to CONJURE in ESSENCE, an abstract constraint specification language. ESSENCE’s support for abstract decision variables with types such as set, multiset, relation and function, as well as nested types, such as set of sets and multiset of relations, allows a problem to be specified without committing to constraint modelling decisions. To illustrate, consider the fragment of the ESSENCE specification of the Social Golfers Problem [Sellmann and Harvey, 2002] presented in Figure 1.

Figure 1: An ESSENCE problem specification of the Social Golfers Problem (Problem 10 on CSPLib). In a golf club there are a number of golfers who wish to play together in $g$ groups of size $s$. Find a schedule of play for $w$ weeks such that no pair of golfers play together more than once.

*The full version of this paper was published in Artificial Intelligence journal [Akgün et al., 2022].
ing a set of partitions of golfers subject to some constraints, which can be specified in ESSENCE via a single abstract decision variable, as presented in the figure where the variable is shaded.

The work presented in [Akgun et al., 2022] summarises and extends over fifteen years of our work on automated constraint modelling. Our earliest work on refinement-based automated constraint modelling appeared between 2002 and 2005 [Frisch et al., 2002; Frisch et al., 2003; Bakewell et al., 2003; Frisch et al., 2005b; Frisch et al., 2005c]. We introduced the ESSENCE language in 2005 [Frisch et al., 2005a; Frisch et al., 2007], which is the subject of a separate journal article [Frisch et al., 2008]. Following the presentation of initial prototypes [Frisch et al., 2005c; Akgun et al., 2010] the first full version of CONJURE was presented in 2011 [Akgun et al., 2011] then extended to handle automated symmetry breaking [Akgun et al., 2013; Akgun et al., 2014], and presented in detail in Akgun’s thesis [Akgun, 2014]. [Akgun et al., 2022] gives a complete overview of CONJURE, including the most recent advances.

1.1 Contributions

CONJURE provides class-level refinement of specifications containing arbitrarily nested types and expressions into efficient constraint models. Achieving this goal required several significant contributions and insights, which we summarise here:

- CONJURE is unique in refining problem class specifications to class-level constraint models.
- Multiple models are generated from one ESSENCE specification by following different rule application pathways.
- CONJURE is able to refine nested abstract types (for example, a set of sets of integers) without enumerating all possible values of the inner type (in this example, set of integers).
- Symmetry introduced during refinement is broken consistently and completely.
- CONJURE is able to generate channelled models by representing an abstract decision variable in more than one way, with an elegant mechanism for producing channelling constraints from a simple equality constraint.
- Model selection is achieved via the simple and lightweight COMPACT EEP heuristic, which is shown to select good models in many cases.
- The system is evaluated comprehensively on 42 problem classes from CSPLib [Jefferson et al., 1999], demonstrating that CONJURE is able to generate models similar to models in the literature produced by experts.

2 Automated Modelling in Conjure

In this section we set the scene for automated modelling by describing CONJURE itself, the pipeline of tools it sits within, and the languages produced and consumed by CONJURE and the other tools.

![Figure 2: Automated Constraint Modelling Pipeline](image)

2.1 The Pipeline

Our modelling and solving pipeline is illustrated in Figure 2. An ESSENCE problem specification is given to CONJURE, which refines the specification into a set of concrete models in ESSENCE PRIME. Both the specification and the model typically relate to a problem class, i.e. they both have problem class parameters that need to be instantiated before instances of the class can be solved. CONJURE separately translates problem class parameters expressed in ESSENCE into ESSENCE PRIME using the representations selected when refining the problem specification. This allows the user to solve multiple instances of the same problem class while only performing refinement once.

SAVILE ROW [Nightingale et al., 2017] is the second tool in the pipeline. It takes as input the model and problem class parameters in ESSENCE PRIME, and produces output for a number of different solvers. SAVILE ROW instantiates the model and performs optimisations before translating the instance into the input language of a solver. Currently SAVILE ROW translates to CP solvers MINION [Gent et al., 2006] and Gecode [Schulte et al., 2023], the learning CP solver Chuffed [Chu et al., 2018], SAT solvers such as Glucose [Audemard and Simon, 2009], MaxSAT solvers such as Open-WBO [Martins et al., 2014], and SMT solvers such as Yices [Dutertre, 2014], Z3 [De Moura and Bjørner, 2008], and Boolector [Niemetz et al., 2014 published 2015].

Once a solution has been found SAVILE ROW translates the solution back into ESSENCE PRIME. CONJURE then translates the ESSENCE PRIME solution back into ESSENCE. Thus the user of CONJURE can specify a problem in terms of abstract types such as partition, and receive solutions in terms of the same types.
2.2 Summary of the ESSENCE Language

CONJURE takes as input an abstract problem specification written in ESSENCE and automatically generates ESSENCE PRIME models as output. ESSENCE is a high-level problem specification language providing a rich set of built-in domains and domain constructors (parameterised domains), such as multi-sets, functions, and partitions. Decision variables can have these domains so as to precisely encode what they mean, and to avoid the need to model these complex domains via multiple decision variables with simpler domains. ESSENCE domains that are not directly represented in ESSENCE PRIME are called abstract domains and domains that are shared between the two languages are called concrete domains (Boolean, int, and matrices of these). We also characterise domains as compound when they contain multiple elements (such as a tuple or matrix). Tuples and records contain a fixed number of fields. Fields in a tuple domain are identified by their position and fields in a record domain are identified by the field name. Variants are tagged unions: they contain a single value for one of the components, tagged by the name of the component. Domains and domain constructors may be nested arbitrarily, allowing for rich domains such as a partition of sets of integers.

For further details the reader is referred to the original journal paper describing ESSENCE [Frisch et al., 2008] and the frequently updated documentation accompanying the CONJURE release [Özgür Akgün et al., 2022].

3 Refinement Rules in CONJURE

CONJURE translates an abstract problem specification written in ESSENCE into a concrete model in ESSENCE PRIME via a series of transformations. These transformations are written as rules in CONJURE. There are two main kinds of rules: representation selection and expression refinement. Applying representation selection rules to each abstract variable in a specification corresponds to choosing a viewpoint for the problem. A viewpoint is a selection of variables with associated domains sufficient to characterise the solutions to the problem. Different viewpoints give rise to fundamentally different models of a problem [Law and Lee, 2002; Smith, 2006]. Multiple representation selection rules may be applied to the same abstract variable to create a channelled model [Cheng et al., 1996], in which a single abstract decision variable is refined in multiple ways. Expression refinement rules rewrite expressions to use one of the selected representations of an abstract variable. Thus the two types of rules correspond to modelling steps taken by human model-makers: selection of a viewpoint or viewpoints, and formulating the constraints.

Refinement rules in CONJURE encode known modelling transformations that are well established in the literature and are known to be correct. We do not formally prove the correctness of the refinement rules; a full and formal exposition of the rules together with proofs of correctness is out of the scope of this paper.

3.1 Representation Selection Rules and Symmetry Breaking

Representation selection rules operate on decision variables or parameters with abstract domains. When a representation selection rule is applied to a domain, it removes the outermost abstract type and replaces it with a concrete type such as a matrix. The output domain is not necessarily concrete, however a concrete domain can always be reached by repeated application of representation selection rules.

In some cases the output domain of a representation selection rule may have values in its domain that do not correspond to values of the input domain. In this case, structural constraints are needed to rule out these values.

3.2 Expression Refinement Rules

Expression refinement rules are the second kind of rules in CONJURE. They are used to translate ESSENCE expressions to equivalent ESSENCE PRIME expressions. They may or may not depend on the representations of decision variables and parameters. Rules that do not depend on representations are called horizontal rules, and those that do are called vertical rules. Horizontal rules do not change the representation of decision variables, they merely translate ESSENCE expressions to other ESSENCE expressions. Horizontal rules are representation independent, and they reduce the need for a very large number of representation-dependent vertical rules.

4 Model Selection with the COMPACTEP Heuristic

CONJURE is able to produce multiple models by enumerating all possible ways of selecting representations. If time is limited it is sensible to provide a rapid model selection method, avoiding both generating all models and training using instance data. In earlier work we proposed a method based on racing [Akgun et al., 2013] to select a subset of the models that perform well on a given set of training instances. Racing methods allow comparing alternative algorithms without necessarily having to run all algorithms on all instances. Racing for model selection can be very computationally expensive. The focus of this paper is on refinement within CONJURE so we omit model selection methods that are essentially external to CONJURE such as racing.

CONJURE contains greedy model selection heuristics that are used for making local decisions during model generation. These can be employed during both representation selection and expression refinement. The default heuristic is called COMPACTEP, which stands for “compact except parameters”, and it is a combination of the COMPACT heuristic and the SPARSE heuristic. We define these heuristics in the following.

The COMPACT heuristic favours transformations that produce simpler types of variables and smaller expressions at each point during refinement where multiple rules are applicable. We define the compact ordering on abstract types as follows: concrete domains (such as bool, matrix) are smaller than abstract domains; within concrete domains, bool is smaller than int and int is smaller than matrix. These rules are applied recursively, so that a
one-dimensional matrix of int is smaller than any two-dimensional matrix. Abstract type constructors have the ordering set < mset < sequence < function < relation < partition, which is also applied recursively. At each stage of representation selection, the COMPACTEP heuristic will select the smallest domain according to this order.

During expression refinement COMPACT chooses the rule that produces the most shallow abstract syntax tree (AST) directly following its application. The SPARSE heuristic is intended to enable small representations of parameter values. It employs a built-in ordering of representations that gives priority to those that take advantage of sparsity.

The default COMPACTEP heuristic is a combination of these two heuristics: during representation selection, CONJURE uses the SPARSE heuristic when representing problem class parameters and the COMPACT heuristic for everything else.

5 Evaluation: CONJURE Produces Kernels of Good Models

CONJURE provides full coverage of the ESSENCE language. It has at least one variable representation rule (typically several) for every abstract variable type, as well as horizontal and vertical expression refinement rules for every operators defined on them. In this section we test the hypothesis that the kernels of constraint models written by experts can be automatically generated by refining a problem’s abstract specification. For two CP models to have the same model kernel, they need to share the same viewpoint, the same representation of decision variables and the same formulation of the problem constraints, together with symmetry breaking. Expert models might have additional features such as implied constraints or dominance breaking [Beck and Prestwich, 2004] constraints, these are not considered to be in the kernel of the CP model for this evaluation. Some expert models contain global constraints that are not present in ESSENCE PRIME. In these cases, if CONJURE generates an equivalent decomposition then we consider the two models to have the same kernel.

In order to test this hypothesis, we took a diverse set of 42 benchmark problems drawn from the literature and refined them with CONJURE. Our main source for these problems is CSPLib [Jefferson et al., 1999]. We cover the entire CSPLib problem class collection (at the time of writing), except those problems that are naturally represented using only matrices of Booleans or integers, i.e. without the facilities that ESSENCE provides in addition to those of lower level constraint modelling languages.

In [Akgün et al., 2022], we present the set of problem classes and the abstract types of their decision variables in ESSENCE. Additionally, we cite the papers that contain a kernel that CONJURE is able to generate. We begin by noting the variety of decision variable types involved in the benchmark problems, representing further evidence that the current collection of rules, the rewrite rule mechanism, and the CONJURE system as a whole is capable of refining a wide variety of abstract problem specifications into concrete models. The number of models generated for a problem specification depends on the number of representation options for its decision variables.

6 Conclusion

In this extended abstract and in the full version of this paper [Akgun et al., 2022] we have presented the automated constraint modelling system CONJURE. It employs a set of refinement rules to transform the specification of a parameterised problem class in the abstract constraint specification language ESSENCE into a concrete constraint model. By varying the selection and application of these rules CONJURE can produce a set of alternative models. We have demonstrated on a large set of problem classes that, in the vast majority of cases, the set produced includes those formulated by human experts in the literature. Furthermore, we have presented a heuristic by which an effective model can be selected.

Acknowledgments

This work is supported by the EPSRC grants EP/P015638/1 and EP/P026842/1.

References


