Finite Domain Modeling and Solving in the Monadic Constraint Programming Framework

P. Wuille
K.U.Leuven, Departement of Computer Science
pieter.wuille@cs.kuleuven.be

T. Schrijvers
K.U.Leuven, Departement of Computer Science
tom.schrijvers@cs.kuleuven.be

1 Introduction

High-level Constraint Programming (CP) languages are an active topic of research. We subdivide existing languages in two classes:

- Modeling languages: stand-alone languages whose programs are mapped to one or more constraint solver backends. They are usually highly declarative and allow convenient model transformations. Because designing and developing a new (modeling) language from scratch is a significant effort, existing modeling languages often run behind on the state of the art in programming language features such as static typing or abstraction mechanisms. Examples include the Zinc family of languages [3].
- A programming language API for writing and solving constraint models, tied to a particular constraint solver. This API can be an integral part of the language (an embedded domain-specific language, or EDSL), or a set of library functions—or a combination of both. This approach is often Turing-complete and offers the abstraction mechanisms of the host language, which are sometimes more imperative in nature. Examples are Gecode [2] or FaCiLe [1].

The Monadic Constraint Programming (MCP) framework [5] — including its Finite Domain (FD) subsystem — combines the advantages of both. While it is a library for the existing programming language Haskell [4], it uses its built-in features for extending the functionality of existing operators and functions, as well as the ability to add custom operators, to provide a clean and declarative syntax for specifying constraint problems. The result is an embedded domain specific language (EDSL) for CP in Haskell.

2 Design

MCP has a layered design, consisting of high-level interfaces for writing models, a core layer that maintains search trees, and multiple backends to do the actual propagation of constraints. The search tree is made up of the constraint model, that is extended with labeling nodes at the leaves. During the search, these leaves are expanded recursively, until solutions or inconsistencies are found. This search system is independent from the solvers and labeling strategies, and works for generic search trees.

The framework contains an FD modeling subsystem that allows FD problems to be specified using simple expressions and operators over integers, booleans and arrays. It provides powerful higher-order abstractions such as map and fold (reduce) over arrays, as well as the higher-order functionality provided by Haskell itself. It encapsulates the particulars of any given FD solver, including the range of primitives. It is implemented as a wrapper-solver around underlying solvers [6], allowing constraint problems to be written using high-level boolean expressions over integer and boolean variables, and arrays of these. The high-level expressions are broken down and translated to (lower-level) constraints supported by the underlying solvers.

Supported solvers include a primitive Haskell-only solver, a runtime solver using the Gecode C++ library, and a staged solver that generates C++ code.

References

[1] N. Barnier. *Application de la programmation par contraintes à des problèmes de gestion du trafic aérien.* PhD thesis, Institut National Polytechnique de Toulouse, December 2002. `http://www.recherche.enac.fr/opti/papers/thesis/`.

[2] Gecode Team. Gecode: Generic constraint development environment, 2006. Available from `http://www.gecode.org`.

[3] K. Marriott et al. The design of the Zinc modelling language. *Constraints*, 13(3):229–267, 2008.

[4] S. Peyton Jones et al. The Haskell 98 language and libraries: The revised report. *Journal of Functional Programming*, 13(1):0–255, Jan 2003.

[5] T. Schrijvers, P. Stuckey, and P. Wadler. Monadic Constraint Programming. *J. Func. Prog.*, 19(6):663–697, 2009.

[6] P. Wuille and T. Schrijvers. Monadic Constraint Programming with Gecode. In *Proceedings of the 8th International Workshop on Constraint Modelling and Reformulation*, pages 171–185, 2009.