

## Problem-specific constraint solver generation

Constraints are a powerful, natural means of knowledge representation and inference in many disparate fields. Simple illustrative examples include: adjacent countries on the map cannot be coloured the same, the orders assigned to a slab of steel must not exceed the slab's capacity, and a warehouse should only be opened if it serves at least one store. This generality underpins the success with which constraint programming has been applied to a wide variety of disciplines, such as scheduling, industrial design, and combinatorial mathematics, to name but a few examples [Pug95][Wal96].

The problems constraint programming attempts to solve are ubiquitous and easily understood. Despite their intuitiveness, solving this kind of problem programmatically and developing general-purpose algorithms which can be applied to any constraint problem remains one of the major challenges of computer science. Although the basics of constraint programming have been well-known for decades [JS79], general-purpose constraint solvers which scale to real-life problems are still an area of active research [Bar99].

Constraint solving of a combinatorial problem usually proceeds in two phases. First, the problem is modelled by a set of constraints on decision variables that its solutions must satisfy. This is already one of the major challenges of constraint programming, as there is almost always more than one way to model a particular problem. There is work underway to simplify this process [FHJ+08].

The second phase is the actual solution of the problem. A constraint solver uses the information given in the model to automatically search for solutions. This process works without user intervention, which limits the ways in which a badly performing model can be detected and corrected. The most important factor affecting the performance of a model, the choice of algorithms, is fixed by the implementation of the solver.

Current constraint solvers are capable of handling large problems, if modelled correctly [GJM06]. The state of the art is that the models are fine-tuned by experts who have an in-depth understanding of the solver algorithms to yield the best performance. In the case of complex, practical problems, this is often the only way of making finding a solution feasible.

This proposal addresses a major challenge – to maintain the natural representational advantages of constraints while delivering a substantial increase in the efficiency and scalability of constraint solvers without the need of expert intervention [Pug04].

To meet this challenge, a constraint solver generator will be developed that, given a constraint model, will produce a highly-optimised constraint solver tailored to that model alone. This will enable a fine-grained optimisation not possible for a general solver and hence allow the solution of much larger, more difficult problems without resorting to manual fine-tuning. It will bring new applications from both industry and academia within the reach of far less experienced users, substantially widening access to this powerful technology.

Constraint solving will proceed in three instead of the usual two phases. First, the constraint problem is modelled. The second phase is to let the constraint solver generator analyse the model to determine which algorithms and techniques should be employed to solve it. The result of this analysis is used to generate a constraint solver best fit for the particular model of the problem. In

the final phase the problem is solved.

Not only constraint problem solving, but also modelling benefits from a constraint solver generator. The model can focus on the abstract aspects of the problem without having to consider which ways of modelling would interact best with the solver.

The proposed system should be able to generate constraint solvers for large problem instances reliably and efficiently. It should not only enable non-expert users to benefit from constraint programming, but also empower developers and researchers to implement new solving algorithms and techniques transparently, without requiring the problem to be remodelled. The constraint solver generator can simply analyse the model again and create a new solver which uses the new algorithms.

A universal constraint solver generator would represent a significant step forward in constraint programming research.

## References

- [Bar99] Roman Barták. Constraint programming: In pursuit of the holy grail. In *Proceedings of Workshop of Doctoral Students '99*, 1999. <http://kti.ms.mff.cuni.cz/~bartak/html/publications.html>.
- [FHJ<sup>+</sup>08] Alan M. Frisch, Warwick Harvey, Chris Jefferson, Bernadette M. Hernandez, and Ian Miguel. Essence: A constraint language for specifying combinatorial problems. *Constraints Journal*, 13(3), July 2008. To appear.
- [GJM06] Ian P. Gent, Chris Jefferson, and Ian Miguel. MINION: A fast scalable constraint solver. In *Proceedings of the Seventeenth European Conference on Artificial Intelligence*, pages 98–102, 2006.
- [JS79] Guy L. Steele Jr. and Gerald J. Sussman. Constraints. In *APL '79: Proceedings of the international conference on APL: part 1*, pages 208–225, New York, NY, USA, 1979. ACM Press.
- [Pug95] Jean-François Puget. Applications of constraint programming. In *CP '95: Proceedings of the First International Conference on Principles and Practice of Constraint Programming*, pages 647–650, London, UK, 1995. Springer-Verlag.
- [Pug04] Jean-François Puget. Constraint programming next challenge: Simplicity of use. In *Principles and Practice of Constraint Programming*, pages 5–8, 2004.
- [Wal96] Mark Wallace. Practical applications of constraint programming. *Constraints Journal*, 1(1):139–168, September 1996.